starts the chain over again, every $(r + 1)$st draw will be scored. This circuit is used for raising matrices to powers without employing control cards.

ASCHER OPLER

Dow Chemical Co.
Pittsburg, Calif.

[1] This is actually sampling without replacement and therefore does not strictly conform to the MARKOV process originally described. However, if the ratio of the number of punched cards to the order of the matrix is very large, nonreplacement has negligible effect on the remaining transition probabilities until the exhaustion of the deck approaches.

[2] H. HOTELLING, "Some new methods in matrix calculation," *Ann. Math. Stat.* v. 14, 1943, p. 1–34.

[3] The IBM 101 statistical machine should be able to invert matrices up to the 30th order because of its selector capacity.

[4] Random numbers on punched cards are available from the RAND Corp., Santa Monica, Calif.

Computing Logical Truth with the California Digital Computer

1. Introduction. A problem which occasionally arises and in quite surprisingly diverse fields of endeavor is the computation of a truth table for a sentence built up of simple sentences connected by the simple sentential connectives. The problem is to find the truth value of the complete sentence for all combinations of truth values for the component sentences. If the complete sentence is a combination of n two-valued sentences, then it itself is two-valued and 2^n possible conditions must be considered. Thus if the number of different component sentences is at all large, an extremely tedious computation is necessary.

Since the computation itself is performed in a routine manner, the possibility of using a high-speed, automatic computer to carry through the details suggests itself. Indeed, a machine for this purpose (the Kalin-Burkhart Logical-Truth Calculator) has already been built.[1] It seems desirable, however, to investigate the possibility of solving these problems using general-purpose, digital computers as many of these will be in operation within the next few years. With this in mind, a program for truth-table calculation is worked out in the following for the California Digital Computer [MTAC, v. 5, p. 57–61]. The general plan should be applicable to any digital computer.

Most problem-solving programs require three basic routines. First the solution of the problem with a given set of data, second storing the result at the proper location in the memory, and third altering the data as required for the next computation. These will be discussed in subsequent sections.

Since the details of a computation program depend to a considerable extent on the characteristics of the computer for which it is developed, a brief description of the operations needed and available in the California Digital Computer (commonly called Caldic) will be presented. Other operations than those described are also available, but the discussion will not be complicated by describing them.

The program and data are given to the machine in the form of holes punched in a standard teletype tape. Results are printed out of the machine on a similar tape. Numerical data and results are in the form of decimal numbers which are used by the machine in a binary-coded form. The fact that numbers available to the operator are decimal is of importance in programming problems. This is particularly true of problems of the sort considered here which are essentially binary in nature. The numbers contain ten decimal digits and algebraic sign.

Before computation starts, data and orders are all transferred from the tape to the internal memory of the machine. This transfer takes approximately 18 milliseconds per number. The internal memory is a magnetic drum with a capacity of 10000, ten digit numbers. Each memory position is identified by a four digit, decimal number. No differentiation is made between orders and numbers as far as the memory is concerned. The number stored in a memory box is not altered by any operation except that of putting another number in the box.

The first two digits in an order identify the operation to be performed. (The operator sees and uses them as letters, the machine as numbers.) The next four digits usually are the address of the operand in the memory. Exceptions will be discussed with the particular operations concerned. The final four digits are zeros.

The results of computations appear in the A register. It takes a special order to transfer them to the memory if this is necessary before another operation is to be performed.

The other register of interest is the order counter. The last entry on the input tape is a four digit number which is stored in the order counter and is the address of the first order. Thereafter as each order is completed the number in the order counter is increased by one to give the address of the next order. An exception is when a special order is used to change the number in the order counter.

The operations of interest are described in the following. The time for the operation should be understood to be the average time for a large number of random repetitions of the operation.

1. ad M. Average time 17.6 milliseconds. M is a four digit number. Add the number in the memory box M to the number in the A register. This operation takes proper account of the algebraic signs of the two numbers to be added. In case the operation causes the A register to overflow, the machine stops unless the next order is the cp order. (See below.) At the completion of the ad operation the sum stands in the A register but without the overflowing digit if such exists.
2. su M. Average time 17.6 milliseconds. Subtract the number in the memory box M from the number in the A register. At the end of the operation the difference stands in the A register. The remarks about sign and overflow under ad apply here also.
3. mh M. Average time 32.4 milliseconds. Multiply the number in memory box M by the number in the A register. This operation takes proper account of the algebraic sign of the two factors. It proceeds as if the decimal point is just to the left of each of the two factors. Therefore no overflow can occur.
4. tm M. Average time 17.3 milliseconds. Transfer the number in the A register to the memory box M. This operation leaves the A register clear.
5. sl n. Average time 9.1 milliseconds. n is a four digit number with the first three digits always zero. Shift the number in the A register n

columns to the left. The first n digits after the sign are lost and n zeros are added at the right.

6. sr n. Average time 8.8 milliseconds. n same as in the sl operation. Shift the number in the A register n columns to the right. The n right hand digits are lost and n zeros are inserted after the sign.

7. cp n. Average time 8.8 milliseconds. n is any four digit number. If an overflow occurred during the preceding operation, replace the number in the order counter by n. Otherwise ignore this order.

8. sp n. Average time 8.8 milliseconds. n is any four digit number. Replace the number in the order counter by n.

9. ca M. Average time 17.5 milliseconds. Clear the address part of the order in the memory box M and add the remainder of the order to the number in the A register.

10. po M. Average time 1117.2 milliseconds. Print out the number in memory box M.

In general some of the above orders would also involve an additional register, the R register. However, in all the computations discussed in this paper the R register will hold nothing but zeros so may be ignored.

2. Problem Solution. In the following discussion marks used to distinguish a thing from its name will be omitted unless necessary to avoid ambiguities.

The operations on or between sentences to be considered are: negation, conjunction, disjunction, material implication, and equivalence. For purposes of the following discussion the word, combination, will represent any one of the latter four.

It will be convenient to make use of the following conventional terms in describing the problem solution. A schema is the result of writing in a row the following symbols in any order: (,), p_j, and the signs for negation and the various combinations. The sign, p_j, represents the various component sentences, j taking on various integral values to distinguish between them. Only well-formed schemas, i.e. schemas in which the parentheses are used to give meaningful groupings, will be used.

The various p_j's are, by definition, schemas of zero order. A schema of order n + 1 is constructed either by putting a negation sign before a schema of order n (which itself must be set off by parentheses) or by forming a combination of two schemas (each set off by parentheses) one of which is of order n and the other of which is of order not greater than n.

It is assumed that the truth values of all schemas of zero order are available. Computation proceeds first by computing the truth values of schemas of order one. Then from the values of schemas of order zero and of order one the truth values of all schemas of order two can be calculated. This process continues until the complete schema has been evaluated. The result is transferred to the memory, the p_j's are altered to the next set of values to be used, and the process is repeated.

This may be illustrated by the following example. Suppose the fifth order schema

$$((p_1 \rightarrow p_0) \wedge (((p_0 \wedge p_1) \wedge \sim p_3) \rightarrow p_2)) \wedge ((p_2 \rightarrow p_1) \wedge (\sim p_0 \rightarrow \sim p_3))$$

is to be evaluated. (Parentheses around individual variables and their negations are omitted.) Let a_j represent schemas of order one, b_j schemas of order two, and so on. Then

$$\begin{array}{ll}
a_1 \leftrightarrow (p_1 \rightarrow p_0) & b_1 \leftrightarrow (a_5 \wedge a_2) \\
a_2 \leftrightarrow (\sim p_3) & b_2 \leftrightarrow (a_4 \rightarrow a_2) \\
a_3 \leftrightarrow (p_2 \rightarrow p_1) & c_1 \leftrightarrow (b_2 \wedge a_3) \\
a_4 \leftrightarrow (\sim p_0) & c_2 \leftrightarrow (b_1 \rightarrow p_2) \\
a_5 \leftrightarrow (p_0 \wedge p_1) & d_1 \leftrightarrow (c_2 \wedge a_1) \\
 & e_1 \leftrightarrow (d_1 \wedge c_1)
\end{array}$$

Within any order-group any order of calculation may be used. The order shown is convenient since a_5, b_2, c_2, and d_1 are used only in the succeeding computation and thus need not be transferred to the memory. This possibility should always be exploited since in complicated problems it can lead to appreciable savings in computation time.

It remains to work out the evaluation of the negation and combinations. Many procedures have been tried, and those tabulated below require less computation time than any others considered. The times given include transfer from and to memory which may sometimes be avoided as discussed above. The operations to be performed follow from left to right in sequence. The symbols p and q are used for the component sentences. The results of the operations for the various cases are tabulated below the operation. The symbol (x) indicates "the address in the memory of the number x." The symbol C(M) indicates "the contents of the memory box M." The symbol 0^x1 indicates "x zero's followed by the digit 1."

Negation (52.5 milliseconds)

p	ad(01)	su(p)	tm M	C(M)
00	01	01	00	01
01	01	00	00	00

Conjunction (76.4 milliseconds)

p	q	ad(p)	mh(q)	sl0002	tm M	C(M)
00	00	00	0000	00	00	00
00	01	00	0000	00	00	00
01	00	01	0000	00	00	00
01	01	01	0001	01	00	01

Disjunction (88.0 milliseconds)

p	q	ad(p)	ad(q)	ad(09)	sr0009	sl0008	tm M	C(M)
00	00	00	00	09	0^90	00	00	00
00	01	00	01	10	0^91	01	00	01
01	00	01	01	10	0^91	01	00	01
01	01	01	02	11	0^91	01	00	01

Material Implication (88.0 milliseconds)

p	q	ad(q)	su(p)	ad(10)	sr0009	sl0008	tm M	C(M)
00	00	00	00	10	0^91	01	00	01
00	01	01	01	11	0^91	01	00	01
01	00	00	−01	09	0^91	00	00	00
01	01	01	00	10	0^91	01	00	01

Equivalence (109.8 milliseconds)

p	q	ad(p)	ad(q)	ad(98)	cp0008	ad(01)	cp0009	sl0002	ad(01)	tm M	C(M)
00	00	00	00	98	98	99	99	00	01	00	01
00	01	00	01	99	99	1.00	00			00	00
01	00	01	01	99	99	1.00	00			00	00
01	01	01	02	1.00	00				01	00	01
Order counter		0001	0002	0003	0004	0005	0006	0007	0008	0009	

It will be noticed that 01 is the truth value of a true sentence and 00 is the truth value of a false sentence. The zero in the first place simplifies the computation of the implication and has no effect on other computations.

3. Handling of Results. Since results consist of a single digit, it is possible to store ten of them in one memory box. Thus the first answer may be put in the left hand digit position, the second in the next to the left and so on until the memory box is filled. Then the contents of the box may be printed out, the box cleared, and the process repeated.

Storing ten answers in each box reduces the number of print operations by a factor of ten which is desirable since printing is by far the slowest of the machine operations.

Printing during computations makes the results available during computation which may be convenient. Also if the machine breaks down during computation, it will not be necessary to recalculate completely.

4. Altering Data. In order to make use of the advantages of automatic machine calculation, the machine itself should compute all but the initial data. In this problem the data are in the form of (binary) truth values and it is desirable to carry out the computation for all possible combinations of truth values.

Let the truth value (0 or 1) of each component sentence be p_j where j ranges over the values from 0 to $n - 1$, n being the number of component sentences to be considered. Construct a binary number k the left hand (or most significant) digit being the result of writing the value of p_{n-1}, the next most significant digit being the result of writing the value of p_{n-2}, and so on. The least significant digit will be the result of writing the value of p_0. All combinations are covered if k starts as zero and then is increased by unity for each computation until every digit of k is one.

In machines using binary arithmetic this procedure can be worked out quite readily. The machine being considered here uses decimal arithmetic which complicates the problem somewhat. The procedure finally adopted makes use of the theorem[2] which follows. For economy in writing, "p_j" will be considered equivalent to "the result of writing the value of p_j."

Definition. A binary number k is defined as the array

$$p_{n-1}p_{n-2} \cdots p_1p_0.$$

Theorem. The number k is replaced by $k + 1$ if and only if for every j (if $0 \leq j \leq n - 1$, then p_j is replaced by $1 - p_j$ if and only if for every m (if $0 \leq m < j$ then $p_m = 1$)). Or more compactly

$$k(\mathrm{B})k + 1 \leftrightarrow_j (0 \leq j \leq n - 1 \rightarrow (p_j(\mathrm{B})1 - p_j \leftrightarrow_m (0 \leq m < j \rightarrow p_m = 1)))$$

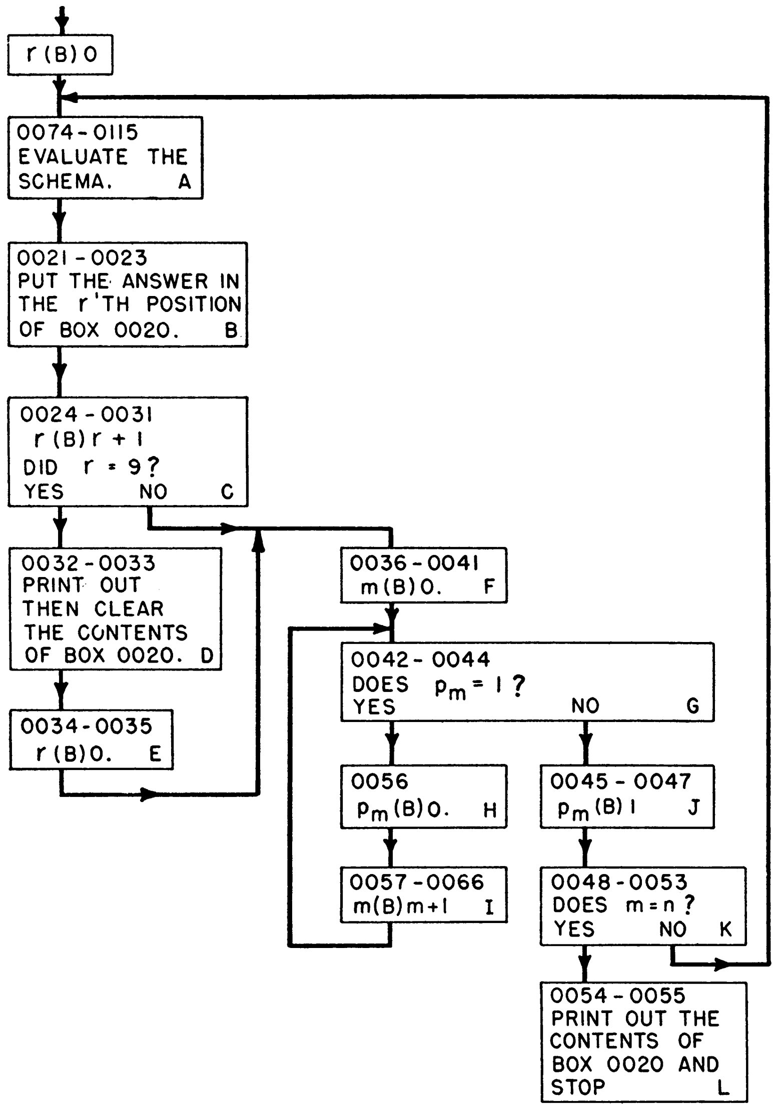
(It is understood that p_j remains unchanged unless it is replaced by $1 - p_j$ by the above rule, and that at every step p_0 is replaced by $1 - p_0$.)

This procedure is programmed quite readily.

5. Terminating the Problem. The machine will not stop, of course, unless it has definite instructions to do so. These may be obtained readily by adding another digit p_n to the left of k. This is zero throughout the computation but will change to one when a one is added to the last value of k to be considered. Thus in the preceding section when the largest value of j satisfying the last equivalence equals n, the computation is finished and the machine may be instructed to stop.

The situation may occur in which all possible combinations of truth values need not be considered. For instance, only true or only false values of a particular sentence may be of interest. In this case this value will be held as a constant (and not stored in the memory with the other p's) and n will be one less than the number of component sentences.

6. The Program. The operations discussed in the four preceding sections are embodied in the flow chart, figure 1, and together make up the program. When the flow chart is worked out in sufficient detail, the program is readily



r(B)0
0074-0115 EVALUATE THE SCHEMA. A
0021-0023 PUT THE ANSWER IN THE r'TH POSITION OF BOX 0020. B
0024-0031 r(B)r+1 DID r = 9? YES NO C
0032-0033 PRINT OUT THEN CLEAR THE CONTENTS OF BOX 0020. D
0034-0035 r(B)0. E
0036-0041 m(B)0. F
0042-0044 DOES p_m = 1? YES NO G
0056 p_m(B)0. H
0057-0066 m(B)m+1 I
0045-0047 p_m(B)1 J
0048-0053 DOES m=n? YES NO K
0054-0055 PRINT OUT THE CONTENTS OF BOX 0020 AND STOP L

FIGURE 1

FLOW CHART FOR THE PROGRAM

coded or translated into a language to which the machine will respond. A code for this problem is given in Appendix I.

Within the rectangles of the flow chart are first the memory boxes in which the instructions to perform the operation demanded in the rectangle are to be stored (for the particular code given in Appendix I). Secondly the operation to be performed is given. In case the operation to be performed is in the form of a question, the alternative answers to the question indicate the paths by which the computation will proceed. The capital letter in the rectangle identifies the operation for the discussion below.

The expression $x(\mathrm{B})y$ means that at this point y is to be substituted for x wherever x appears in the program. It may be read "x becomes y."

Operations D and E must be performed a number of times equal to the largest digit which is less than 0.1×2^n. Operations H and I must be performed a number of times equal to $2^{n+1} - (n + 2)$. Operation L must be performed once. All other operations must be performed 2^n times.

Operation A is self-explanatory. Operations B, C, D, and E are for handling the results. Operations F, G, H, I, and J take care of altering the data. Operations K and L are for terminating the problem.

7. Conclusion. The program given here was developed and coded with the idea in mind of minimizing the changes necessary in going from one problem to another. It was thought that 19 variables is beyond the limits of practicality (for this machine, at least, since 19 variables would take over six days of machine time). As long as no more than 19 variables are used the instructions in memory boxes 0000 to 0072 inclusive will apply to any problem. Memory box 0073 will contain the number $1.00–0.01n$ (decimal point to the left of the most significant figure of numbers in the memory). This will be used to terminate the computation. Starting with memory box 0074 the schema is evaluated as discussed in section 2. The schema given there as an example is the one coded in Appendix I. The truth value of the schema is finally left in the left hand digit position of the A register. (Note that the routines of section 2 leave the value in the position next to the left hand one so that the last operation must be altered accordingly.) Then the command sp0021 is used to go to operation B of figure 1.

In preparation for working these problems a master tape going only through 0072 will be prepared. For a particular problem this will be duplicated and the additional orders needed will be added on the duplicate. Then the entire contents of the tape will be fed into the machine, the A and R registers cleared, and the machine started with the order counter at 0074.

Using the average times given in section I the problem time is

$$T = 0.91 \times 2^n - 0.18n + 0.75 + 0.077c \times 2^n \text{ seconds,}$$

where c is the number of connectives in the schema. It should be emphasized that this is a rough estimate and depends on the assumption that on the average it will take half a drum revolution to find the memory box requested. In the example given here $n = 4$ and $c = 11$ ($\sim p_3$ need be calculated only once) yielding $T = 26$ seconds. (It is understood, of course, that this short example would be worked more economically manually.)

Available data on the Kalin-Burkhart machine are rather sparse, but it would be interesting to compare its operation with that of the program developed here. It would appear that the Kalin-Burkhart machine will compute the problem a little faster than will the Caldic. The Caldic, however, is rather slow as electronic computers go. Problem preparation time should be about the same. On the other hand the Kalin-Burkhart machine is limited to problems with 12 component sentences and 11 connectives. With the Caldic the number of component sentences is limited only by the time for which the machine is available for the particular problem and there appears to be no practical limitation on the number of connectives.

Appendix I. The Coded Program. These quantities will be fed in sequence into the memory the first into box 0000, the second into box 0001, and so on until the list is exhausted. It is understood that the machine will treat letters as digits and that each number will be filled out with zeros to make ten digits. Every quantity has a plus sign attached. Under each memory box number is given its contents.

```
0000     0001     0002     0003     0004     0005     0006     0007
0        0        0        0        0        0        0        0

0008     0009     0010     0011     0012     0013     0014     0015
0        0        0        0        0        0        0        0

0016     0017     0018     0019     0020     0021     0022     0023
0        0        0        0        0        sr0000   ad0020   tm0020

0024     0025     0026     0027     0028     0029     0030     0031
ad0021   sl0005   ad0069   cp0032   sr0005   ca0021   tm0021   sp0036

0032     0033     0034     0035     0036     0037     0038     0039
po0020   tm0020   ca0021   tm0021   ca0043   tm0042   ca0047   tm0047

0040     0041     0042     0043     0044     0045     0046     0047
ca0056   tm0056   ad0000   ad0071   cp0056   sl0002   ad0067   tm0000

0048     0049     0050     0051     0052     0053     0054     0055
ad0042   sl0004   ad0073   cp0054   sl0002   sp0074   po0020   STOP

0056     0057     0058     0059     0060     0061     0062     0063
tm0000   ad0042   ad0072   tm0042   ad0047   ad0072   tm0047   ad0056

0064     0065     0066     0067     0068     0069     0070     0071
ad0072   tm0056   sp0042   01       09       10       98       99

0072     0073     0074     0075     0076     0077     0078     0079
000001   96       ad0000   su0001   ad0069   sr0009   sl0008   tm0116

0080     0081     0082     0083     0084     0085     0086     0087
ad0067   su0003   tm0117   ad0001   su0002   ad0069   sr0009   sl0008

0088     0089     0090     0091     0092     0093     0094     0095
tm0118   ad0067   su0000   tm0119   ad0000   mh0001   sl0002   mh0117

0096     0097     0098     0099     0100     0101     0102     0103
sl0002   tm0120   ad0117   su0119   ad0069   sr0009   sl0008   mh0118

0104     0105     0106     0107     0108     0109     0110     0111
sl0002   tm0121   ad0002   su0120   ad0069   sr0009   sl0008   mh0116

0112     0113     0114     0115
sl0002   mh0121   sl0003   sp0021
```

Appendix II. Glossary.

n	the number of component sentences in the complete schema.
p_j	the jth component sentence.
a_j, b_j, etc.	the jth component schema of order 1, 2, etc.
c	the number of connectives in the schema.
T	estimated computing time in seconds.
1 (or 01)	in the proper context the truth value of a true sentence.
0 (or 00)	in the proper context the truth value of a false sentence.
x(B)y	y is to replace x wherever x appears.
(h)	address in the memory of the number h.
C(M)	contents of the memory box M.
$\sim$	It is not the case that . . .
$\wedge$	. . . and . . .
$\vee$	. . . or . . .
$\rightarrow$	If . . . then . . .
$\leftrightarrow$	. . . if and only if . . .
$_x$	For every x . . .

WILTON R. ABBOTT

Univ. of California
Berkeley

[1] EDMUND C. BERKELEY, *Giant Brains*, New York, 1949, Ch. 9.
[2] GEORGE W. PATTERSON, *Logical Syntax and Transformation Rules*. Moore School of Electrical Engineering, Research Division Report 50–8, Univ. of Pennsylvania, Philadelphia, 1949.

On the Accuracy of Runge-Kutta's Method

1. Introduction. While the accuracy of the most frequently used methods of integrating differential equations is fairly well known, that of the Runge-Kutta method does not seem to be too well established; except for a formula in Bieberbach's text[1] on differential equations there are no references pertaining to the error inherent in the Runge-Kutta method to be found in the standard textbooks on this subject.

Since this method may be employed quite advantageously in many cases of practical interest it is important to have on hand an estimate of the error. The purpose of the following sections is to provide such an estimate. As a comparison shows, the bound derived for this error seems to be somewhat better than the one cited by Bieberbach.

2. Runge-Kutta's Fourth Order Method. In trying to find that solution of the differential equation

$$dy/dx = f(x, y), \qquad y(x_0) = y_0, \tag{1}$$

at $x_1 = x_0 + h$, which agrees with the exact Taylor expansion about x_0:

$$y(x_1) = y_0 + hy_0' + h^2(y_0''/2) + h^3(y_0'''/6) + h^4(y_0^{\text{iv}}/24) + h^5(y_0^{\text{v}}/120) + \cdots \tag{2}$$