SIMULATION OF A TURING MACHINE ON A DIGITAL COMPUTER

Robert W. Coffin, Harry E. Goheen,** and Walter R. Stahl****

I. INTRODUCTION

The theory of algorithms relies heavily upon the conceptual and theoretical usefulness of the Turing Machine.[1] Recent work by Trakhenbrot[2] has given further support to the hypothesis that "all algorithms can be given in the form of functional matrices and executed by the corresponding Turing Machines." Such a statement does not immediately suggest that all problems should be reduced to their equivalent Turing Machine, but the implication is clear that if certain problems, recognizable as algorithms, do not lend themselves to a solution in a formal logic structure, they may be reduced to a Turing scheme using a suitable strategy. Examples of such problems are revealed in the work by Lusted and Stahl[3] in the problem of medical diagnosis and by Stahl and Goheen[4] in simulation of the operation of biological cell systems.

For years the practical value of the Turing Machine has been discounted because there is widespread belief among mathematicians that a Turing Machine always requires a tape of infinite length and that there is no assurance that a particular algorithm will achieve a stable solution in a finite length of time. These misconceptions probably stem from the fact that the Turing Machine invariably enters the discussion of complex, self-organizing automata in which these conditions might exist. Furthermore, the work of A. M. Turing[5] preceded the growth of modern electronic technology by several years, consequently to implement a Turing Machine as a physical device, at that time, would have been an impractical and costly undertaking.

The authors and their colleagues have found that simulation of the Turing Machine on a Digital Computer is a useful and practical tool not only for problem solving and validation of algorithms but also for teaching students the fundamentals of programming. This paper will describe the digital program by which a generalized simulation of the Turing Machine has been accomplished.

II. PRELIMINARY CONSIDERATIONS

If there are M configurations corresponding to the Q-levels of the Turing Machine each requiring m bits, and if the Turing Machine requires an alphabet of size N, each symbol requiring n bits, we shall need a list of MN strings which we call machine quintuples. A machine quintuple will consist of:

(a) m bits identifying a Q-level,

(b) n bits identifying a symbol of the alphabet,

(c) m bits designating the next Q-level,

*Chief Programmer, Dept. of Automatic Data Processing and Biomathematics, Oregon Regional Primate Research Center, Beaverton, Oregon.

**Professor of Mathematics, Dept. of Mathematics, Oregon State University, Corvallis, Oregon.

***Associate Scientist, Dept. of Automatic Data Processing and Biomathematics, Oregon Regional Primate Research Center, Beaverton, Oregon, and Associate Professor, Dept. of Mathematics, Oregon State University, Corvallis, Oregon.

(d) 2 bits designating the next motion (either Left, Right, or Place),

(e) n bits designating the symbol to replace the symbol represented by (b) at the current location in the Turing tape.

In the descriptive language of the Turing Table, the last (m + 2 + n) bits represent the (i, j) entry of the Turing Table, where i and j are given in (a) and (b) of the quintuple. The list of quintuples will require a maximum of 2MN (m + n + 1) bits. As a practical matter, Turing Tables are rarely "saturated," i.e., the entire matrix filled with non-zero entries, however the maximum bit requirement:

$$2MN\ (m + n + 1) = A \qquad (1)$$

is an important program design criterion.

To provide for rapid and intelligible interpretation of a processed Turing tape, it is desirable to limit the magnitude of the Turing alphabet to those alphabets of available input-output devices. It is important to note that this restriction is one of convenience since, in binary form, a symbol is limited only by the allowable n. If a Turing Machine requires a large alphabet it would be necessary to numerically precode the alphabet, or alternatively, use a two-for-one notation.

In addition to (1) a portion of the total available memory must be used to store the Turing tape. If this memory allocation is b bits, then the maximum possible length of the Turing tape will be the parameter:

$$[b/n] = B \qquad (2)$$

If the total number of bits in the memory of a specific computer is S, and the simulation program requires s bits, we have the following inequality which defines the maximum combination of the Turing Machine and tape that can be processed.

$$A + B < S - s \qquad (3)$$

In addition to (3), there is another restriction imposed upon the simulation program. We denote by r the time used in passing from one quintuple to the next. If an algorithm requires T quintuple cycles for solution, we have the following inequality in which C denotes the time available for the processing of the algorithm.

$$r\,T < C \qquad (4)$$

The inequalities (3) and (4) are viewed as general restrictions upon the magnitude of Turing Machine which can be simulated on a particular computer.

III. STRATEGY OF SIMULATION

The simulation of a machine in a digital computer requires two major routines that are logically distinct from each other.[6] The first of these routines serves to receive and interpret external notation and generate the "compiled logic" of the device to be simulated. Whether or not this routine is a true compiler, in the sense of current usage of the word, depends upon the logical complexity of the simulated device and the corresponding notation required to fully describe its function. The logic of the Turing Machine is quite simple and completely embodied in the construction of the quintuple which in turn is determined by the author of the Turing Machine. Therefore the builder is required only to make the notational translation of externally coded quintuple statements into machine language. To distinguish between a true compiler and the routine that generates the "compiled logic", we shall henceforth refer to the latter as the "builder" routine.

The essential steps of the builder routine are outlined below.

1. Read quintuple card.
2. HALT card? Yes, go to 14.
3. STOP quintuple? Yes, go to 12.
4. Extract (Q, S, Q, M, S).
5. Convert Q-s to binary.
6. Assign movement code.
7. Form function: (Q, M, S).
8. Form identifier (Q, S).
9. Store identifier and function.
10. Update storage.
11. Go to 1.
12. Insert stop code for function.
13. Go to 8.
14. Exit Builder.

Each quintuplet is punched on a separate card in the format

Q S Q M S for example
10 A 11 R B

where 10 A is the Turing Matrix identification pair and 11 R B is the function triplet.

The only exception to this format is the special stop card with the format:

Q S STOP

We have chosen to use a special stop code for three reasons. First, the special stop code makes it possible to distinglish between a normal halt and an error in the Turing Machine. Secondly, it is far more economical of computer time to make a simple, one-step test rather than go through three, multi-step tests to determine the (Qi, Sj, Qi, P, Sj) quintuple. Finally, the special stop code and the natural stop quintuple are theoretically equivalent.

The quintuple deck is terminated with the halt card:

HALT

The dimensions of the Turing Table are determined by the builder. The row dimension is equal to the largest defined value of Q, and the column dimension is equal to the number of different symbols defined in the quintuple list. It must be noted that the quintuple notation makes the determination of the dimensions unnecessary except for purposes of arraying the matrix on a line printer or other device.

Since the speed of the builder routine is limited by the speed of the card reader there is time for nonessential checking for mispunched cards. The inclusion of three such checking features has proved to be very helpful in ridding quintuple decks of gross errors. The first and most valuable check is dependent upon an alphabet definition card that is read immediately prior to the first quintuple card. A short routine generates a check list consisting of all of the defined symbols which can legally occur in the quintuple deck. Then as the quintuple cards are read, each symbol is checked against the defined alphabet. In the event an undefined symbol appears, the illegal symbol and the card containing the error are typed out with an error message. The error routine keeps a list of the addresses where the incorrect cards should be located, and the builder proceeds to the next card. After the HALT card is detected the program stops and allows the operator to insert the corrected cards in the card hopper. Upon restarting, the program will translate the corrected cards and store the corresponding entries in their original order. Similar checks are made to detect illegal movement and non-numeric symbols in the Q state, with identical procedures for error recovery.

After the building is complete, before any simulation can take place, a Turing tape must be loaded into memory. The Turing tape is punched on one or more cards, and each symbol is checked against the original alphabet list. If an illegal symbol occurs in the Turing tape, an error message is typed and the reader routine halts. Upon restarting, the corrected tape is read, checked and stored.

The second part of the program is called the "driver" routine which operates on an element of compiled logic, provides functional continuity between elements, and hence forces the simulation. In the ideal case, no part of the driving routine performs any function except to operate on an element or provide continuity between elements, however, the timing restriction will certainly have to be included in the driver routine and is considered to be an allowable artifact. Other artifacts may be justified for purposes of demonstration or instruction but are undesirable burdens on the simulation since time is at a premium and each artifact must consume time on every quintuple cycle. Clearly, any artifact that causes the driver routine to amend itself or alter an element of compiled logic is unallowable.

The following outline shows the essential steps of the driver routine.

1. Extract symbol from Turing tape.
2. Merge symbol with previous Q-level (from step 7).
3. Find function corresponding to (Q, S).
4. Is function a STOP? Yes, go to 14.
5. Substitute function symbol in Turing tape.
6. Move Turing tape "window".
7. Extract next Q-level.
8. Has time been exceeded? Yes, go to 10.
9. Go to 1.
10. Type time warning.
11. Halt.

12. Insert new time restriction.
13. Go to 1.
14. Output final Turing tape.
15. Halt.

The step which consumes the largest segment of time in the "Driver" is the table lookup to find the next function triplet, (step 3). The time required for this search can be greatly reduced if we take advantage of the natural tendency of Turing Machine authors to group important Q-levels together. Thus, the probability of finding the next quintuple in a close proximity to the current quintuple is much higher than finding it at a great distance. After a check is made against the current quintuple, a flutter search is used in which the search begins with the next entry forward of the current quintuple and then shifts to the last entry preceding then to the second entry forward and so forth until the required quintuple is located. It is important to note that the order in which the quintuples occur is important only to the efficiency of the algorithm but does not influence its logical operation.

IV. SAMPLE ALGORITHMS

The first problem is to detect the change of pattern in a strip of contrasting white and black segments as shown in Fig. 1(a).

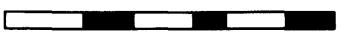

Figure 1(a).

Figure 1(b).

Figure 1(c).

To code the pattern as a string of symbols we simply assign the letter A to the white segments and the letter B to the black segments, and in both cases make the number of letters proportional to the length of each segment as shown in Fig. 1(b). The problem is to write an algorithm in Turing notation that will operate on this coded string in such a way that only the points at which the pattern changes are marked. The desired solution is shown in Fig. 1(c). This is logically equivalent to generating a unit pulse based upon a criterion of wave height or frequency.

The Turing Table for this algorithm is shown in Fig. 2, and is a simple 3 × 3 matrix with 2 null entries.

	*	A	B
Q1	2R*	—	—
Q2	2P*	2RA	3RB
Q3	3P*	2RB	3RA

Figure 2.

The strategy is to proceed from the left * to the right passing from Q1 to Q2 when either an A or B is encountered. If an A is seen, control stops in Q2 until a B is seen. The first B is retained and control goes to Q3 where successive B's are replaced by A's until an A is seen. The first A is replaced by a B and control passes back to Q2 which begins the algorithm again. If a B is seen as the first symbol, the same type of procedure is followed except control goes immediately to state Q3.

Fig. 3 shows the complete input deck as it appears before the start of a run.

```
*AB
1 *  2 R A
2 *  STOP
2 A  2 R A
2 B  3 R B
3 *  STOP
3 A  2 R B
3 B  3 R A
HALT
*AAAABBBAAABBAAABBB*
END TEST
```

Figure 3.

The first card is the alphabet definition card consisting of the three legal symbols: *, A, and B. Following the alphabet card is the seven-card Turing program which is terminated with a HALT card. The coded test string shown in Fig. 1(b) is next, followed by an END OF TEST card.

Fig. 4 shows the sequential operation of the Turing Machine. The current quintuple is shown at the right side of the tape. The delta symbol under each line shows the next symbol to be viewed by the "window".

```
*AAAABBBAAABBAAABBB*    [001,*] [002,R,*]
 Δ
*AAAABBBAAABBAAABBB*    [002,A] [002,R,A]
  Δ
*AAAABBBAAABBAAABBB*    [002,A] [002,R,A]
   Δ
*AAAABBBAAABBAAABBB*    [002,A] [002,R,A]
    Δ
*AAAABBBAAABBAAABBB*    [002,A] [002,R,A]
     Δ
*AAAABBBAAABBAAABBB*    [002,B] [003,R,B]
      Δ
*AAAABABAAABBAAABBB*    [003,B] [003,R,A]
       Δ
*AAAABAAAAABBAAABBB*    [003,B] [003,R,A]
        Δ
*AAAABAABAABBAAABBB*    [003,A] [002,R,B]
         Δ
*AAAABAABAABBAAABBB*    [002,A] [002,R,A]
          Δ
*AAAABAABAABBAAABBB*    [002,A] [002,R,A]
           Δ
*AAAABAABAABBAAABBB*    [002,B] [003,R,B]
            Δ
*AAAABAABAABAAAABBB*    [003,B] [003,R,A]
             Δ
*AAAABAABAABABAABBB*    [003,A] [002,R,B]
              Δ
*AAAABAABAABABAABBB*    [002,A] [002,R,A]
               Δ
*AAAABAABAABABAABBB*    [002,A] [002,R,A]
                Δ
*AAAABAABAABABAABBB*    [002,B] [003,R,B]
                 Δ
*AAAABAABAABABAABAB*    [003,B] [003,R,A]
                  Δ
*AAAABAABAABABAABAA*    [003,B] [003,R,A]
                   Δ
*AAAABAABAABABAABAA*    [003,*]  STOP
                   Δ
RUN COMPLETE
```

Figure 4.

In this example the Turing tape is printed at the completion of every quintuple cycle consequently the "window" is shown to move only one symbol per line of output. Such an extensive output is desirable only for short, illustrative examples since an algorithm may involve many thousands of cycles for solution.

The second algorithm is a popular child's problem in which a number of boys and soldiers are to be transported across a river in a row boat. The boat is capable of holding one or two boys, or one soldier, but not one boy and one soldier. The basic solution strategy is a simple five step algorithm as follows:

1. two boys across river
2. one boy back
3. boy out, soldier in, soldier across.
4. boy back
5. go to 1.

When all of the soldiers are across the river, the algorithm reduces to a back and forth shuttle involving only boys with a net gain of one boy per two river crossings. The problem can be coded in an interesting manner with symbols representing the various objects involved.

G is a grassy spot on the river bank,
W is the water,
F is a seat in the row boat,
B is a boy,
S is a soldier,
% is the left bank of the river,
)=(is the right bank of the river
≠ is the tape end marker symbol
— is an intermediate symbol used to denote movement of the boat.

A sample configuration is shown in Fig. 5(a) with the solution configuration in Fig. 5(b).

At the beginning all of the boys and soldiers are on the right side of the river with the boat at the right bank. In the final string, all of the boys and soldiers have been transported to the left side and the boat is at the left bank.

Figure (6) shows the Turing program in quintuple form required to process any such input configuration.

#GGGGG%WWFF□SBBSSG#

Figure 5(a).

#BSSSB%FFWW□GGGGGG#

Figure 5(b).

1 # 1 R #	1 G 2 R G	2 % 3 R %
2 G 2 R G	3 = 4 R =	3 F 3 R F
3 W 3 R W	3 # 4 R #	4 # STOP
4 G 4 R G	4 S 4 R S	4 B 5 R G
5 S 5 R S	5 B 6 L G	5 G 5 R G
5 # 24 L #	6 F 7 L B	6 S 6 L S
6 = 6 L =	6 G 6 L G	6 # 6 L #
7 S 8 L S	7 B 8 P B	7 F 7 L B
7 W 8 R W	7 G 8 P B	8 W 8 R W
8 # 13 R #	8 % 13 R %	8 S 10 L -
8 F 11 L -	8 B 9 L -	9 F 8 L F
9 W 12 R B	9 - 9 L B	9 B 8 L B
9 S 8 L S	10 S 8 L S	10 - 10 L S
10 W 12 R S	10 F 8 L F	10 B 8 L B
11 S 8 L S	11 - 11 L F	11 F 8 L F
11 B 8 L B	11 W 12 R F	12 S 10 L W
12 B 9 L W	12 F 11 L W	12 - 12 R -
13 S 15 L F	13 F 13 R F	13 B 14 L F
14 B 14 L B	14 S 14 L S	14 F 14 L F
14 % 14 L %	14 G 16 R B	14 # 31 L %
15 G 17 R S	15 F 15 L F	15 B 15 L B
15 S 15 L S	15 % 15 L %	16 F 16 R F
16 W 19 L W	16 % 16 R %	16 B 16 R B
16 S 16 R S	17 G 17 R G	17 B 18 R G
17 S 17 R S	18 % 18 R %	18 F 16 R B
18 G 18 R G	18 S 18 R S	18 B 18 R B
19 B 20 R -	19 - 19 R -	19 = 23 L =
19 F 21 R -	19 # 23 L #	19 W 19 L W
20 F 19 R F	20 B 19 R B	20 - 20 R B
20 W 22 L B	21 F 19 R F	21 W 22 L F
21 B 19 R B	21 - 21 R F	22 B 20 R W
22 F 21 R W	22 - 22 L -	23 B 24 R F
23 F 23 L F	24 # 24 R #	24 G 25 L B
24 F 24 R F	24 = 24 R =	24 B 24 R B
24 S 24 R S	25 = 26 R #	25 G 25 L G
25 # 3 P =	25 S 25 L S	25 B 25 L B
26 S 27 L G	26 # 28 L #	26 G 26 R G
26 B 26 R B	26 F 26 L F	26 W 8 R W
27 F 26 L S	27 # 27 L #	27 B 27 L B
27 G 27 L G	28 G 28 L G	28 F 29 L B
28 = 28 L =	28 B 29 L B	29 B 25 L B
29 F 29 L F	29 = 30 R =	29 # 31 L =
29 G 29 L G	29 W 8 R W	30 = 30 R =
30 B 28 L G	30 F 30 R F	30 G 30 R G
30 W 30 R W	31 S 31 L S	31 B 31 L B
31 F 31 L F	31 W 31 L W	31 G 32 L B
31 % 30 R #	32 # STOP	32 G 32 L G
32 B 32 L B	32 S 32 L S	

Figure 6.

In addition to the standard solution algorithm, all contingencies are anticipated such as the presence of only one boy, or the absence of soldiers.

Because of the number of quintuple cycles involved it is not practical to show the test string after each cycle, however, figure 7 shows every second cycle.

Note that the "window" appears to take a two-symbol jump between lines.

V. DISCUSSION

The sample algorithms were run on a machine with a cycle time of 8 microseconds (SDS 920). The limiting factor, as mentioned above, is the time involved in finding the next quintuple in the total list. Since the efficiency is influenced by the order in which the quintuple deck is organized, the question arises whether or not it is possible to obtain an optimum ordering. Generally, this would be quite unlikely since an optimum order is dependent on the input string which may be composed of a random group of symbols. However, if quintuples, containing symbols which are used in conjunction with each other, are placed together in the input deck, the efficiency can probably be improved.

In the Boys and Soldiers algorithm we achieved a speed of 3,000 quintuple cycles/second by ordering the deck row-by-row as it appears in a Turing Functional Matrix. By rearranging some of the cards, particularly in Q-levels which deal with the left and right movement of the boat, we were able to increase the speed to about 3,300 quintuple cycles/seconds. In the opposite direction, we tried to arrange the deck in such a way that we produced a very inefficient operation and were able to reduce the speed to slightly less than 1,000 quintuple cycles/second. By shuffling the cards like a bridge deck, we obtained about 2,100 quintuple cycles/second.

It is not surprising that the row-by-row ordering is quite efficient since a Turing programmer normally adopts a row-by-row strategy in solving a problem, consequently the probability is high that significant quintuples will be grouped together. Experience with a

```
#GGG%WWFF□BSB#  [001,G] [002,R,G]
      △
#GGG%WWFF□BSB#  [002,G] [002,R,G]
     △
#GGG%WWFF□BSB#  [003,W] [003,R,W]
      △
#GGG%WWFF□BSB#  [003,F] [003,R,F]
       △
#GGG%WWFF□BSB#  [003,□] [004,R,□]
        △
#GGG%WWFF□GSB#  [005,S] [005,R,S]
         △
#GGG%WWFF□GSG#  [006,S] [006,L,S]
        △
#GGG%WWFF□GSG#  [006,□] [006,L,□]
       △
#GGG%WWBB□GSG#  [007,F] [007,L,B]
      △
#GGG%WW-B□GSG#  [008,B] [009,L,-]
     △
#GGG%WB-B□GSG#  [012,-] [012,R,-]
      △
#GGG%WBBW□GSG#  [009,-] [009,L,B]
     △
#GGG%WBBW□GSG#  [008,W] [008,R,W]
    △
#GGG%B-BW□GSG#  [009,W] [012,R,B]
     △
#GGG%B-WW□GSG#  [012,B] [009,L,W]
    △
#GGG%BBWW□GSG#  [009,B] [008,L,B]
   △
#GGG%FBWW□GSG#  [013,B] [014,L,F]
    △
#GGB%FBWW□GSG#  [014,G] [016,R,B]
   △
#GGB%FBWW□GSG#  [016,F] [016,R,F]
    △
#GGB%FBWW□GSG#  [016,W] [019,L,W]
     △
#GGB%F-BW□GSG#  [020,W] [022,L,B]
    △
#GGB%W-BW□GSG#  [022,F] [021,R,W]
     △
#GGB%WFBW□GSG#  [021,B] [019,R,B]
      △
#GGB%WF-W□GSG#  [019,B] [020,R,-]
       △
```

Figure 7.1.

```
#GSG%WB-W#BGG#  [019,F] [021,R,-]
      △
#GSG%WB-F#BGG#  [022,-] [022,L,-]
     △
#GSG%WWBF#BGG#  [020,-] [020,R,B]
      △
#GSG%WWBF#BGG#  [019,#] [023,L,#]
     △
#GSG%WWFF#BGG#  [023,B] [024,R,F]
      △
#GSG%WWFF#BGG#  [024,#] [024,R,#]
     △
#GSG%WWFF#BBG#  [024,G] [025,L,B]
      △
#GSG%WWFF□BBG#  [025,#] [003,P,□]
     △
#GSG%WWFF□GBG#  [004,B] [005,R,G]
      △
#GSG%WWFF□GGG#  [006,G] [006,L,G]
     △
#GSG%WWFB□GGG#  [006,F] [007,L,B]
    △
#GSG%WWBB□GGG#  [007,W] [008,R,W]
     △
#GSG%WB-B□GGG#  [009,W] [012,R,B]
    △
#GSG%WB-W□GGG#  [012,B] [009,L,W]
     △
#GSG%WBBW□GGG#  [009,B] [008,L,B]
    △
#GSG%W-BW□GGG#  [008,B] [009,L,-]
     △
#GSG%B-BW□GGG#  [012,-] [012,R,-]
      △
#GSG%BBWW□GGG#  [009,-] [009,L,B]
     △
#GSG%BBWW□GGG#  [008,%] [013,R,%]
    △
#GSG%FBWW□GGG#  [014,%] [014,L,%]
   △
#GSB%FBWW□GGG#  [016,%] [016,R,%]
    △
#GSB%FBWW□GGG#  [016,B] [016,R,B]
     △
#GSB%F-WW□GGG#  [019,B] [020,R,-]
      △
#GSB%F-BW□GGG#  [022,-] [022,L,-]
       △
```

Figure 7.2.

```
#GGB%WF-BHGSG#  [022,-] [022,L,-]
      Δ
#GGB%WWFBHGSG#  [021,-] [021,R,F]
      Δ
#GGB%WWFBHGSG#  [019,H] [023,L,H]
      Δ
#GGB%WWFFHGSG#  [024,H] [024,R,H]
      Δ
#GGB%WWFF#BSG#  [025,H] [026,R,#]
      Δ
#GGB%WWFF#BGG#  [026,S] [027,L,G]
      Δ
#GGB%WWFF#BGG#  [027,#] [027,L,#]
      Δ
#GGB%WWFS#BGG#  [026,F] [026,L,F]
      Δ
#GGB%WW-S#BGG#  [008,F] [011,L,-]
      Δ
#GGB%WF-S#BGG#  [012,-] [012,R,-]
      Δ
#GGB%WFSW#BGG#  [010,-] [010,L,S]
      Δ
#GGB%WFSW#BGG#  [008,W] [008,R,W]
      Δ
#GGB%F-SW#BGG#  [011,W] [012,R,F]
      Δ
#GGB%F-WW#BGG#  [012,S] [010,L,W]
      Δ
#GGB%FSWW#BGG#  [010,F] [008,L,F]
    Δ
#GGB%FSWW#BGG#  [013,F] [013,R,F]
      Δ
#GGB%FFWW#BGG#  [015,F] [015,L,F]
    Δ
#GGB%FFWW#BGG#  [015,B] [015,L,B]
  Δ
#GSG%FFWW#BGG#  [017,B] [018,R,G]
    Δ
#GSG%BFWW#BGG#  [018,F] [016,R,B]
      Δ
#GSG%BFWW#BGG#  [016,W] [019,L,W]
      Δ
#GSG%B-FW#BGG#  [021,W] [022,L,F]
      Δ
#GSG%W-FW#BGG#  [022,B] [020,R,W]
      Δ
#GSG%WBFW#BGG#  [020,F] [019,R,F]
        Δ
```

Figure 7.3.

```
#GSB%WFBWHGGG#  [021,-] [021,R,F]
      Δ
#GSB%WFBWHGGG#  [019,W] [019,L,W]
      Δ
#GSB%WF-BHGGG#  [020,W] [022,L,B]
      Δ
#GSB%WW-BHGGG#  [022,F] [021,R,W]
      Δ
#GSB%WWFBHGGG#  [021,B] [019,R,B]
      Δ
#GSB%WWFFHGGG#  [023,B] [024,R,F]
      Δ
#GSB%WWFFHBGG#  [024,G] [025,L,B]
      Δ
#GSB%WWFF#BGG#  [026,B] [026,R,B]
      Δ
#GSB%WWFF#BGG#  [026,G] [026,R,G]
        Δ
#GSB%WWFF#BGG#  [028,G] [028,L,G]
      Δ
#GSB%WWFF#BGG#  [028,B] [029,L,B]
      Δ
#GSB%WWFFHBGG#  [031,F] [031,L,F]
      Δ
#GSB%WWFFHBGG#  [031,W] [031,L,W]
      Δ
#GSB#WWFFHBGG#  [031,%] [030,R,#]
      Δ
#GSB#WWFFHBGG#  [030,W] [030,R,W]
      Δ
#GSB#WWFFHBGG#  [030,F] [030,R,F]
      Δ
#GSB#WWFFHGGG#  [030,B] [028,L,G]
      Δ
#GSB#WWFBHGGG#  [028,F] [029,L,B]
      Δ
#GSB#WWFBHGGG#  [029,W] [008,R,W]
      Δ
#GSB#WF-BHGGG#  [011,W] [012,R,F]
      Δ
#GSB#WF-WHGGG#  [012,B] [009,L,W]
      Δ
#GSB#WFBWHGGG#  [009,F] [008,L,F]
      Δ
#GSB#W-BWHGGG#  [008,F] [011,L,-]
      Δ
#GSB#F-BWHGGG#  [012,-] [012,R,-]
        Δ
```

Figure 7.4.

```
#GSB#FBWWIGGG#   [009,-] [009,L,B]
      Δ
#GSB#FBWWIGGG#   [008,#] [013,R,#]
      Δ
#GSB#FFWWIGGG#   [013,B] [014,L,F]
      Δ
#GSB%FFWWIGGG#   [014,#] [031,L,%]
    Δ
#GSB%FFWWIGGG#   [031,S] [031,L,S]
  Δ
#BSB%FFWWIGGG#   [032,#]  STOP
Δ
RUN COMPLETE
```

Figure 7.5.

particular algorithm would certainly indicate an efficient input order. However, it is an interesting characteristic of a Turing program that it will run to solution regardless of the input order of the instructions. The authors know of no other programming language in which this is true.

VI. CONCLUSION

The value of the simulation of a Turing Machine is two-fold. First, the simulation represents a practical means of solving algorithms in their most fundamental form, that is, in a basic language which is independent of both computer and computer language. Secondly, for non-numeric problems the processing rates are found to be competitive with the more conventional computer languages.

REFERENCES

1. McNAUGHTON, R. "The Theory of Automata, a Survey." *Advances in Computers,* vol. 2, pp. 379-421, edited by F. L. Alt, Academic Press, New York-London (1961).
2. TRAKHTENBROT, B. A. *Algorithms and the Machine Solution of Problems,* (In Russian, 1960). Available under the title, *Algorithms and Automatic Computing Machines,* D. C. Heath and Co., Boston (1963).
3. LUSTED, L. B., and W. R. STAHL, 1963. "Conceptual Basis of Medical Diagnosis." In press, *Proc. of the Conference on the Diagnostic Problem,* University of Michigan, May 1963.
4. STAHL, W. R., and H. E. GOHEEN. "Molecular Algorithms." In press, *J. Theoret. Biology* (1963).
5. TURING, A. M. "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proc. London Mathematical Society,* Series 2, vol. 42 (1936-37), pp. 230-65.
6. AMDAHL, L. Handbook of *Automation Computation and Control,* edited by E. M. Grabbe, E. Ramo, and D. E. Wooldridge, vol. 2, chapter 17, pp. 40-41, John Wiley and Sons, Inc., New York (1959).

7. The authors gratefully acknowledge the advice of Dr. L. B. Lusted, Senior Scientist, Division of Biophysical Sciences, Oregon Regional Primate Research Center.