MINSKY'S SMALL UNIVERSAL TURING MACHINE

RAPHAEL M. ROBINSON
Department of Mathematics
University of California
Berkeley, CA 94720
USA

Received 15 February 1991

Marvin L. Minsky constructed a 4-symbol 7-state universal Turing machine in 1962. It was first announced in a postscript to [2] and is also described in [3, Sec. 14.8]. This paper contains everything that is needed for an understanding of his machine, including a complete description of its operation.

Minsky's machine remains one of the minimal known universal Turing machines. That is, there is no known such machine which decreases one parameter without increasing the other. However, Rogozhin [6], [7] has constructed seven universal machines with the following parameters:

(2 symbols, 24 states), (3 symbols, 11 states),

(4 symbols, 7 states), (5 symbols, 5 states), (6 symbols, 4 states),

(10 symbols, 3 states), (21 symbols, 2 states).

His 4-symbol 7-state machine is somewhat different from Minsky's, but all of his machines use a construction similar to that used by Minsky. The following corrections should be noted: First machine, for $q_6 00 L q_1$ read $q_6 00 L q_7$; second machine, for $q_4 11 R q_4$ read $q_4 11 R q_{10}$; last machine, for $q_2 b_2 b L q_2$ read $q_2 b_2 \vec{b} L q_2$.

A generalized Turing machine with 4 symbols and 7 states, closely related to Minsky's, was constructed and used in [5].

1. Introduction

A Turing machine has a tape, which we take to be infinite in both directions. The tape is divided into squares, each of which contains one symbol from a finite alphabet $s_0, s_1, s_2, \ldots$, of which s_0 is the blank. All but a finite number of the squares are blank. The machine is always in one of a finite number of states $q_0, q_1, q_2, \ldots$, of which q_0 is the initial state. There is a reading and writing head, which always scans one square of the tape. The action of the machine is defined by a finite number of quintuples $q_i s_j s_k R q_l$ and $q_i s_j s_k L q_l$, which indicate that if the machine is in state q_i and the head is scanning s_j, then it will overprint s_k, move right or left to the next square on the tape, and the machine will go into state q_l. There is at most one quintuple for each pair $q_i s_j$. If there is no such quintuple, then the machine halts.

The configuration of the machine at any time consists of the state q_i and the content of the tape, taking the origin at the head. Thus, although we speak of the head moving right or left along the tape, it would be more accurate to say that the tape moves left or right under the head. To take a trivial example, if a Turing machine has one state

International Journal of Mathematics Vol. 2, No. 5 (1991) 551–562

q_0, one symbol s_0, and one quintuple $q_0 s_0 s_0 R q_0$, then the configuration never changes. Although the head is described as always moving to the right, it does not disappear in the distance.

It is generally agreed that Turing machines furnish a suitable definition of computability: a function $f(n)$, from and to the natural numbers $(0, 1, 2, 3, \ldots)$, is computable if it can computed by a Turing machine. There are various possible ways of encoding the input and decoding the output. One possibility is to start the machine with a tape which is blank except for having the symbol s_1 in the first n squares from the scanned square to the right. If the machine halts, then $f(n)$ is taken to be number of squares from the scanned square to the right containing s_1 before reaching a square containing some other symbol. The Turing machine defines a function if it halts for every n. We do not know which Turing machines define functions. Is this a flaw in the definition?

Let us call a class of computable functions certain if every allowed rule is certain to lead to the computation of a function defined for all arguments. We can then enumerate all allowed rules, and hence construct a list $f_0(n)$, $f_1(n)$, $f_2(n)$, ..., which includes each function of the class at least once. We see that $f_n(n) + 1$ is a computable function not belonging to the class. Thus no certain class can include all computable functions. So the uncertainty about which Turing machines define a function is a necessary part of an adequate definition of computability. This idea is developed to prove the undecidability of various forms of the halting problem; see, for example, Minsky [3, Chapter 8].

Computable functions are also called general recursive, or just recursive. The primitive recursive functions form an important certain class of computable functions. They are discussed in many places, including [4].

A Turing machine U is called universal if it can simulate each Turing machine T. We are to give U a code for the structure of T and a code for the initial tape of T. The machine U must halt if T would have halted, and the final tape of U must include a code for the output of T.

In constructing a small universal machine, such as Minsky's, we must allow rather elaborate methods of encoding the input and decoding the output. How can we be sure that we are not cheating, by doing the real work in the encoding and decoding? This can be avoided if we insist that the encoding and deconding processes always belong to certain classes. Then the bulk of a difficult calculation will remain with the Turing machine.

The construction of the small universal Turing machine will proceed in three stages, in the next three sections. In Sec. 2, we show how a Turing machine using many symbols can be simulated by a machine using just two symbols. This reduction is intuitively obvious. We can consider a block of squares on the tape of the new machine as corresponding to a single square on the old tape. Enough new states can be introduced so that the new machine can examine all the squares of the block and remember the symbols there, thus identifying the corresponding symbol on the old tape. It can then take appropriate action. Still, carrying out the details is not altogether simple. A proof was given by Shannon [8], who also showed that the number of states can be reduced to two by increasing the number of symbols. A simplified proof of the reduction to

two symbols is given in Sec. 2. The simplification is at the expense of obtaining a very poor bound for the number of states needed.

To describe the results of Sec. 3, we must first discuss Post productions. They are considered in some detail in Minsky [3, Chapters 12 and 13]. Emil Post studied various transformations of words, where a word is simply a sequence of letters from a given finite alphabet. These transformations were called productions. We shall be interested in a special type of productions called normal productions. Here specified letters are deleted from the beginning of a word, and specified letters added at the end. An example of a normal production is $maW \rightarrow Win$, where W is any word, possibly empty. This would transform *mask* into *skin*, *mad* into *din*, and *ma* into *in*.

Still more special are the tag systems of productions. Here the number of letters deleted from the beginning is the same for all productions, and the letters added at the end depend only on the first deleted letter. The number of letters deleted is called the deletion number. The only productions used in Sec. 3 are tag productions with deletion number 2. For these, we shall use an abbreviated notation, showing just the first deleted letter and the added letters. For example, $l \rightarrow art$ would transform *last* into *start*, *lip* into *part*, and *lo* into *art*, but could not be applied to *l*. If we are using the alphabet $a, b, \ldots, z$, then this would actually correspond to 26 normal productions: $laW \rightarrow Wart, lbW \rightarrow Wart, \ldots, lzW \rightarrow Wart$.

For each initial letter, there should be at most one production of the new type. If there is no such production, then the letter is called a halting letter. The tag productions will halt only if we reach a tag word beginning with a halting letter, or a tag word of length less than 2. For a given class of admissible starting words, we may classify halting letters as actual or virtual. Those that we know can never appear as the first letter of a tag word will be called virtual halting letters, the others actual halting letters.

In Sec. 3, we show that the action of any Turing machine using two symbols can be simulated by a system of tag productions with deletion number 2. This result was due to Cocke and Minsky [1], and may also be found in Minsky [3, Sec. 14.6].

Finally, in Sec. 4, we construct a single 4-symbol 7-state Turing machine which can simulate all of the tag systems constructed in Sec. 3. But these simulated all 2-symbol Turing machines, which in turn simulated all Turning machines. So the machine constructed is indeed a universal Turing machine.

The machine table used is the same as in Minsky's original version [2], except for changing one quintuple. (One quintuple was also changed in [3], but it was a different one.) However, I found it convenient to change the notation. A weak point in Minsky's description was the halting procedure. It was rather complicated, and had the disadvantage that it mutilated the final output of the machine. Here a simpler procedure is used which preserves the output, and leaves the answer immediately to the right of the head.

Rogozhin's version of the 4-symbol 7-state machine has some advantages and some disadvantages as compared to the one used here. It uses only 26 quintuples, as compared with Minsky's 27, and hence may be considered as a simplification. The change does not seem to have been made for that purpose, but to provide a halting

procedure which does not mutilate the answer. His procedure does lose the first two letters of the final tag word, but this is not serious, since in the tag systems which are needed these can be taken as known. Also, his machine stops at the extreme left of the printed tape, and one has to search for an unknown distance to the right in order to find the answer. I do not know whether it is possible to reduce the number of quintuples to 26, and also stop with the answer close at hand.

2. Reduction to Two Symbols

We want to show that the action of a Turing machine using many symbols can be simulated by a Turing machine using just two symbols. To simplify the proof, we will make the reduction gradually, by showing that the number of symbols may be reduced by one if we start with more than two symbols.

Suppose that the given machine uses symbols $s_0, s_1, \ldots, s_n, s_{n+1}$, where $n \geqq 1$. We shall construct a new machine using only the symbols $s_0, s_1, \ldots, s_n$ which can copy the work of the old machine. On the tape of the new machine, major squares and minor squares will alternate. Major squares may contain any symbol $s_0, s_1, \ldots, s_n$, but minor squares can contain only s_0 or s_1. A single square on the old tape will correspond to a pair of squares on the new tape, a major square and the following minor square. If $k \leqq n$, the s_k on the old tape will become $s_k s_0$ on the new tape, but s_{n+1} will be replaced by $s_n s_1$.

We must increase the number of states. Indeed, to each state q_i of the old machine we will make correspond nine state of the new machine, namely $q_i(B)$, $q_i(T)$, $q_i(T_0)$, $q_i(T_1)$, $q_i(R)$, $q_i(L)$, $q_i(CR)$, $q_i(CL)$, and $q_i(LL)$. The letters used are intended to suggest the words Begin, Test, Right, Left, and Change. Every configuration of the old machine will correspond to a configuration of the new machine with the state q_i replaced by $q_i(B)$. The scanned square will be the major square of the pair corresponding to the old scanned square. But the new machine will also have configurations not corresponding to configurations of the old machine.

To each quintuple for the old machine there will be a unique quintuple for the new machine. There will also be a number of fixed quintuples, not dependent on the action of the old machine. Indeed, to each state q_i of the old machine, we will make correspond $n + 12$ quintuples of the new machine, namely

$$q_i(B)s_n s_n R q_i(T), \quad q_i(T)s_y s_y L q_i(T_y), \quad q_i(D)s_y s_y D q_i(B),$$

$$q_i(CR)s_y s_{1-y} R q_i(B), \quad q_i(CL)s_y s_{1-y} L q_i(LL), \quad q_i(LL)s_k s_k L q_i(L),$$

where y is 0 or 1, the direction D is R or L, and $0 \leqq k \leqq n$.

We must now tell how the quintuples for the old machine are replaced. The old quintuple may be taken as

$$q_i s_j s_k D q_l,$$

where D is R or L. The replacement will depend on the values of j and k:

$$\text{If } j < n, \quad k \leqq n, \quad \text{use} \quad q_i(B)s_j s_k D q_l(D).$$

$$\text{If } j < n, \quad k = n+1, \quad \text{use} \quad q_i(B)s_j s_n R q_l(CD).$$

$$\text{If } j = n, \quad k \leqq n, \quad \text{use} \quad q_i(T_0)s_n s_k D q_l(D).$$

$$\text{If } j = n, \quad k = n+1, \quad \text{use} \quad q_i(T_0)s_n s_n R q_l(CD).$$

$$\text{If } j = n+1, \quad k \leqq n, \quad \text{use} \quad q_i(T_1)s_n s_k R q_l(CD).$$

$$\text{If } j = n+1, \quad k = n+1, \quad \text{use} \quad q_i(T_1)s_n s_n D q_l(D).$$

Notice that three of the new quintuples contain R no matter whether D is R or L.

Each step of the computation of the old machine will correspond to 2, 4, or 6 steps for the new machine. Besides the main quintuple corrresponding to the old quintuple, some of the fixed quintuples will be used. If $j < n$, then we are ready for the main quintuple. But if $j = n$ or $j = n+1$, then the new machine is scanning s_n. To determine whether $j = n$ or $j = n+1$, the minor square to the right must be tested. This requires two steps. After the main quintuple is used, at least one more step is needed to bring the head to a major square. But if the minor square must be altered and $D = L$, then three left moves are needed. The procedure should be clear if we follow through two examples, one requiring only two steps, the other requiring six steps. In each case, the quintuples used are shown, and a portion of the resulting tape. The scanned symbol is bracketed.

Example 1. Replace $q_i s_j s_k R q_l$, where $j < n$, $k \leqq n$.

Start $q_i(B)$	$[s_j]s_0 s_x s_y$
$q_i(B)s_j s_k R q_l(R)$	$s_k[s_0]s_x s_y$
$q_l(R)s_0 s_0 R q_l(B)$	$s_k s_0[s_x]s_y$

Example 2. Replace $q_i s_{n+1} s_k L q_l$, where $k \leqq n$.

Start $q_i(B)$	$s_x s_y[s_n]s_1$
$q_i(B)s_n s_n R q_i(T)$	$s_x s_y s_n[s_1]$
$q_i(T)s_1 s_1 L q_i(T_1)$	$s_x s_y[s_n]s_1$
$q_i(T_1)s_n s_k R q_l(CL)$	$s_x s_y s_k[s_1]$

$$q_l(CL)s_1 s_0 L q_l(LL) \qquad s_x s_y [s_k] s_0$$

$$q_l(LL)s_k s_k L q_l(L) \qquad s_x [s_y] s_k s_0$$

$$q_l(L)s_y s_y L q_l(B) \qquad [s_x] s_y s_k s_0$$

3. Using Tag Productions

We shall now show how the action of a Turing machine using two symbols may be simulated by a system of tag productions with deletion number 2.

Let any Turing machine using two symbols $s_0 = 0$ and $s_1 = 1$ be given, where 0 also serves as the blank. Suppose that at a certain instant the machine is in state q_i and the tape reads

$$\ldots c_{-3}c_{-2}c_{-1}c_0c_1c_2c_3\ldots,$$

where c_0 is being scanned. Let

$$j = c_0, \quad m = c_{-1} + 2c_{-2} + 4c_{-3} + 8c_{-4} + \cdots,$$

$$n = c_1 + 2c_2 + 4c_3 + 8c_4 + \cdots.$$

The sums m and n are finite, since all but a finite portion of the tape is blank. The configuration is completely determined by the four numbers (i, j, m, n).

We need to know how two successive configurations (i, j, m, n) and (i', j', m', n') are related. Suppose that the quintuple used is $q_i s_j s_k D q_l$. We see that

$$\text{If } D = R, \quad \text{then} \quad i' = l, \quad m' = 2m + k, \quad n = 2n' + j'.$$

$$\text{If } D = L, \quad \text{then} \quad i' = l, \quad m = 2m' + j', \quad n' = 2n + k.$$

In either case, the new configuration (i', j', m', n') is determined.

As an alphabet for the tag system, we introduce letters A_{ij}, a_{ij}, B_{ij}, b_{ij}, C_{ij}, c_{ij}, D_{ij}, d_{ij}, E_{ij}, e_{ij}, F_{ij}, f_{ij} for each state-symbol pair $q_i s_j$ of the Turing machine. (Some of these may not be needed.) Corresponding to the configuration defined by (i, j, m, n), we use the tag word

$$A_{ij}A_{ij}(a_{ij}a_{ij})^m B_{ij}B_{ij}(b_{ij}b_{ij})^n,$$

from which the four numbers (i, j, m, n) can be recovered. For example, if the Turing machine is started on a blank tape, then the initial configuration will correspond to the tag word $A_{00}A_{00}B_{00}B_{00}$.

The tag productions used will add 1, 2, 3, or 4 letters to the end of the tag word. The productions will be divided into the following four classes.

1. Productions corresponding to $q_i s_j s_k R q_l$:

$$A_{ij} \to C_{ij} C_{ij} (c_{ij} c_{ij})^k, \qquad a_{ij} \to (c_{ij} c_{ij})^2,$$

$$B_{ij} \to D_{ij}, \qquad b_{ij} \to d_{ij}.$$

2. Productions corresponding to $q_i s_j s_k L q_l$:

$$A_{ij} \to C_{ij}, \qquad a_{ij} \to c_{ij},$$

$$B_{ij} \to D_{ij} D_{ij} (d_{ij} d_{ij})^k, \qquad b_{ij} \to (d_{ij} d_{ij})^2.$$

3. Productions corresponding to either of the above:

$$C_{ij} \to E_{l1} E_{l0}, \qquad c_{ij} \to e_{l1} e_{l0},$$

$$D_{ij} \to F_{l1} F_{l0}, \qquad d_{ij} \to f_{l1} f_{l0}.$$

4. Productions independent of the computer action:

$$E_{i0} \to A_{i0} A_{i0} A_{i0},$$

$$E_{i1} \to A_{i1} A_{i1}, \qquad e_{ij} \to a_{ij} a_{ij},$$

$$F_{ij} \to B_{ij} B_{ij}, \qquad f_{ij} \to b_{ij} b_{ij}.$$

We now look at the results of applying the tag productions needed to carry out the step of the Turing machine corresponding to the quintuple $q_i s_j s_k D q_l$. To save writing, the following convention will be used: Unsubscripted letters will be understood to have the subscripts ij. Letters with a single subscript 0 or 1 will be understood to have the subscripts $l0$ or $l1$.

The computation will be divided into two parts. The first part depends on whether $D = R$ or $D = L$, the second part on whether $j' = 1$ or $j' = 0$.

Part 1 for $D = R$.

Start with			$AA(aa)^m BB(bb)^n$.
$A \to CC(cc)^k$,	$a \to (cc)^2$	produce	$BB(bb)^n CC(cc)^{2m+k}$.
$B \to D$,	$b \to d$	produce	$CC(cc)^{m'} Dd^n$.
$C \to E_1 E_0$,	$c \to e_1 e_0$	produce	$Dd^{2n'+j'} E_1 E_0 (e_1 e_0)^{m'}$.
$D \to F_1 F_0$,	$d \to f_1 f_0$	produce	$E_1{}^{j'} E_0 (e_1 e_0)^{m'} F_1 F_0 (f_1 f_0)^{n'}$.

Part 1 for $D = L$.

Start with			$AA(aa)^m BB(bb)^n$.
$A \to C$,	$a \to c$	produce	$BB(bb)^n Cc^m$.
$B \to DD(dd)^k$,	$b \to (dd)^2$	produce	$Cc^{2m'+j'} DD(dd)^{2n+k}$.
$C \to E_1 E_0$,	$c \to e_1 e_0$	produce	$D^{1+j'}(dd)^{n'} E_1 E_0 (e_1 e_0)^{m'}$.
$D \to F_1 F_0$,	$d \to f_1 f_0$	produce	$E_1{}^{j'} E_0 (e_1 e_0)^{m'} F_1 F_0 (f_1 f_0)^{n'}$.

Part 2 for $j' = 1$.

Start with			$E_1 E_0 (e_1 e_0)^{m'} F_1 F_0 (f_1 f_0)^{n'}$.
$E_1 \to A_1 A_1$,	$e_1 \to a_1 a_1$	produce	$F_1 F_0 (f_1 f_0)^{n'} A_1 A_1 (a_1 a_1)^{m'}$.
$F_1 \to B_1 B_1$,	$f_1 \to b_1 b_1$	produce	$A_1 A_1 (a_1 a_1)^{m'} B_1 B_1 (b_1 b_1)^{n'}$.

Part 2 for $j' = 0$.

Start with			$E_0 (e_1 e_0)^{m'} F_1 F_0 (f_1 f_0)^{n'}$.
$E_0 \to A_0 A_0 A_0$,	$e_0 \to a_0 a_0$	produce	$F_0 (f_1 f_0)^{n'} A_0 A_0 A_0 (a_0 a_0)^{m'}$.
$F_0 \to B_0 B_0$,	$f_0 \to b_0 b_0$	produce	$A_0 A_0 (a_0 a_0)^{m'} B_0 B_0 (b_0 b_0)^{n'}$.

Since $i' = l$, in both cases we end with

$$A_{i'j'} A_{i'j'} (a_{i'j'} a_{i'j'})^{m'} B_{i'j'} B_{i'j'} (b_{i'j'} b_{i'j'})^{n'},$$

which represents the next configuration of the Turing machine.

The tag words corresponding to a Turing machine configuration have length at least 4, and intermediate tag words have length at least 3. If the tag system is started with a word corresponding to a machine configuration, then the productions can terminate only by reaching a word which begins with a halting letter. The letters to which no productions are assigned are A_{ij}, a_{ij}, B_{ij}, b_{ij}, C_{ij}, c_{ij}, D_{ij}, d_{ij}, when $q_i s_j$ does not begin a quintuple. Of these, C_{ij}, c_{ij}, D_{ij}, d_{ij} will never appear, hence may be dropped from the alphabet. Of the remaining letters for any such $q_i s_j$, only A_{ij} is an actual halting letter, whereas a_{ij}, B_{ij}, b_{ij} are virtual halting letters. We could eliminate the virtual halting letters by assigning arbitrary productions to them, but there seems to be no advantage in doing this.

For the construction of the small universal Turing machine in Sec. 4, it will be useful

to have just one actual halting letter. So for all $q_i s_j$ which do not begin quintuples, we should drop the subscripts from the corresponding A_{ij}. We may also do this for the a_{ij} and the b_{ij}, but not for the B_{ij}. With this agreement, A is the only actual halting letter. If the tag productions halt, the final word will always have the form

$$AA(aa)^m B_{ij} B_{ij} (bb)^n.$$

Notice that this determines (i, j, m, n), and hence the final configuration of the Turing machine.

4. The Small Universal Turing Machine

We now construct a 4-symbol 7-state Turing machine which can simulate the action of any set of tag productions with deletion number 2, subject to some mild restrictions. Since these restrictions are all satisfied by the tag systems constructed in Sec. 3, this means that we can simulate the action of any Turing machine. Hence the machine constructed is universal.

Assume that the tag system has the alphabets $a_0, a_1, a_2, \ldots, a_p$ (which is a different notation than that used in Sec. 3). Certain words in this alphabet will be admissible starting words. Assume that, when started with an admissible word, the tag productions can halt only be reaching a word starting with a_0. Thus a_0 will be the only actual halting letter, and we can never reach a word of length less than 2. For $1 \leqq i \leqq m$, we assume that a production

$$a_i \to a_{i1} a_{i2} \ldots a_{in_i}$$

is given, where $n_i > 0$. Each a_{ij} is assumed to be some a_k. If $m < p$, then $a_{m+1}, \ldots, a_p$ will be virtual halting letters. If a_i is a halting letter, then we put $n_i = 0$. For $0 \leqq i \leqq p$, we define

$$N_i = (n_0 + 1) + (n_1 + 1) + \cdots + (n_{i-1} + 1),$$

so that $N_0 = 0$ and $N_1 = 1$.

The Turing machine will use symbols 0, 1, 2, 3, where 0 is the blank, and have states $Q0, Q1, \ldots, Q6$, where $Q0$ is the initial state. The tag word $a_r a_s \ldots a_z$ will be coded as

$$S = 2^{N_r} 3\, 2^{N_s} 3 \ldots 3\, 2^{N_z}.$$

Notice that a_0 is coded as an empty string. Thus the tag word $a_0 a_0 a_1 a_1$ would be coded as 33232.

The right side of the i-th production will be coded as

$$P_i = 1\ \ 0^{N_{in_i}} 01 \ldots 01\ \ 0^{N_{i2}} 01\ \ 0^{N_{i1}} 01,$$

where $N_{ij} = N_k$ if $a_{ij} = a_k$. The initial tape of the Turing machine will be taken as

$$\ldots 0\ 0\ 1\ P_m \ldots P_2\ P_1\ 1\ 0 \ldots 0\ S\ 0\ 0 \ldots,$$

where S codes the initial word of the tag system, and there are one or more 0's preceding S. The machine is started in state $Q0$ scanning the first symbol of S.

In the machine table, we show for each state-symbol pair the new symbol and the direction of the motion. The number of the new state is shown only when the new state is different from the old state. For example, if the machine is in state $Q0$ scanning 2, then it overprints 0, moves left, and goes into state $Q1$.

MACHINE TABLE

	0	1	2	3
$Q0$	$2R$	$3R$	$0L1$	$2L4$
$Q1$	$0L$	$3R0$	$0L$	$1L$
$Q2$	$2L4$	$3R$	$2R$	$1R$
$Q3$	$3L4$	$3R$	$2R$	$1R$
$Q4$	—	$3L$	$2L$	$1L5$
$Q5$	$2R2$	$1L6$	$2L$	$1L$
$Q6$	$2R3$	$1R$	$0R$	$0R0$

Almost all of the operation of the Turing machine will consist in running the short shuttle and the long shuttle. The short shuttle locates the production corresponding to the initial letter of the tag word, then the long shuttle copies the required letters from the production to the end of the tag word. Their operation will be described in detail below. We start by listing the relevant parts of the machine table.

Short shuttle right	$Q0(0-2)$
Short shuttle left	$Q1$
Transition	$Q0(3)$
Long shuttle right	$Q2, Q3$
Long shuttle left	$Q4(1-3), Q5, Q6(0)$
Completion	$Q6(1-3)$
Halting	$Q4(0)$

At any time, the tape of the Turing machine, excluding the blank tails, will be divided into three parts: left, middle, and right. The short shuttle runs back and forth across the middle, whereas the long shuttle runs back and forth across the middle and right.

At each stage, each of the three segments uses just two of the four symbols. Here are the symbols used at certain times.

Before short shuttle right	$01 + 01 - 23$
Before short shuttle left	$01 - 23 + 23$
Before long shuttle right	$01 + 21 - 23$
Before long shuttle left	$01 - 23 - 21 +$

The three terms indicate the symbols used in the three segments. We may think of 2 and 3 as disguised forms of 0 and 1. The plus sign indicates the position of the head.

Assume that the initial tag word is $a_r a_s \ldots a_z$, where $1 \leqq r \leqq m$. We need to locate P_r. Notice that there are exactly N_r ones between S and P_r. We coded a_r as 2^{N_r} so that the information would be at hand to locate P_r. The machine starts in state $Q0$ scanning the first symbol of S, which is 2. It is changed to 0, and the machine goes into state $Q1$. We are ready for the short shuttle left. To begin with, the middle is empty. The shuttle finds the rightmost 1, changes it to 3, goes into state $Q0$, and starts the short shuttle right. On the first trip, it changes 0's into 2's, thrusting them into the middle. In general, the short shuttle left changes 2's and 3's into 0's and 1's, and the short shuttle right changes them back. Notice that this enables the short shuttle to recognize the end of the middle when going in either direction. The shuttle goes back and forth N_r times, enlarging the middle at both ends each time.

After N_r round trips, the production P_r which is needed is at the extreme right of the left segment. The short shuttle reaches the first 3 in S, changes it to 2, and the machine goes into state $Q4$ for the long shuttle left. The first trip is truncated, since it starts at the right end of the middle. There will be at least one 3 in the middle, and when it is reached the machine goes into state $Q5$. The half trip converts $01 - 23 + 23$ into $01 + 21 - 23$. When the 1 at the end of P_r is reached, the machine goes into state $Q6$, reaches the 0 to the left, changes it to 2, and goes into state $Q3$ to start the long shuttle right, which will write 3 at the extreme right. Each trip of the long shuttle left starts in state $Q4$ and finishes in state $Q5$. Each trip of the long shuttle in either direction interchanges 1's and 3's.

On the next N_{r1} trips of the long shuttle left, the head will encounter 0 in P_r. This is changed to 2, and the machine goes into state $Q2$, which causes 2 to be written at the extreme right. In this way, $2^{N_{r1}}$ is written at the end. On the next trip left, the symbol 1 is reached in P_r. If $n_r > 1$, then the next symbol to the left will be 0, and as before this causes 3 to be written at the end. After all n_r letters have been copied, the long shuttle left will reach the 1 at the left end of P_r, and the machine goes into state $Q6$. Since there is another 1 just to the left of P_r, the completion phase is started. In this phase, the head moves right, changing 2's to 0's. This pushes the middle back into the left, thus restoring the productions to their original form. In addition, when the right is reached, the first two letters of the tag word are erased. They were originally coded as $2^{N_r}\,3\,2^{N_s}$,

but this was changed to $2^{N_r}2\,2^{N_s}$. The next symbol is 3, whether it was there to begin with or not, and it is also erased. The blank squares created are thrust into the left, leaving the middle empty. The machine goes into state $Q0$, scanning the first symbol of the new S. One tag production has been completed.

The reason for the long shuttle interchanging 1's and 3's on both left and right trips may seem obscure. But notice that this provides exactly the tape which is needed when the last run of the long shuttle has been finished, and the completion phase is to start.

Finally, we must consider what happens when the tag word begins with a_0. In this case, the first symbol of S is 3. The machine begins acting on this word in state $Q0$, scanning this 3. The is changed to 2, the head moves left, and the machine goes into state $Q4$. Since the head is now scanning 0, the machine halts. We need only change the initial symbol of S back to 3, and decode this S, to find the final word of the tag productions. So we have simulated the action of the tag system.

References

1. John Cocke and Marvin Minsky, *Universality of tag systems with* $P = 2$, J. Assoc. Comput. Mach. **11** (1964), 15–20.
2. M. L. Minsky, *Size and structure of universal Turing machines using tag systems*, Recursive Function Theory, Proceedings of Symposia in Pure Mathematics, vol. V, American Mathematical Society, Providence, R. I., 1962, pp. 229–238.
3. Marvin L. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, N. J., 1967.
4. Raphael M. Robinson, *Primitive recursive functions*, Bull. Amer. Math. Soc. **53** (1947), 925–942.
5. Raphael M. Robinson, *Undecidability and nonperiodicity for tilings of the plane*, Invent. Math. **12** (1971), 177–209.
6. Yu. V. Rogozhin, *Seven universal Turing machines* (Russian), abstract, Fifth All-Union Conference on Mathematical Logic, Akad. Nauk SSSR Sibirsk. Otdel., Inst. Mat., Novosibirsk, 1979, p. 127.
7. Yu. V. Rogozhin, *Seven universal Turing machines* (Russian), Systems and Theoretical Programming, Mat. Issled. no. 69, Akademiya Nauk Moldavskoi SSSR, Kishinev, 1982, pp. 76–90.
8. Claude E. Shannon, *A universal Turing machine with two internal states*, Automata Studies, Ann. of Math. Stud. **34** (1956), 157–165.