Universality of (p=2) Tag systems and a 4 symbol 7 state Universal Turing Machine

M. L. Minsky

§0. Introduction.

This report describes (1) an improvement and great simplification of the proof that the "Tag" systems of Post can represent any computable process, and (2) a Universal Turing machine with just four symbols and seven states--the smallest yet reported.

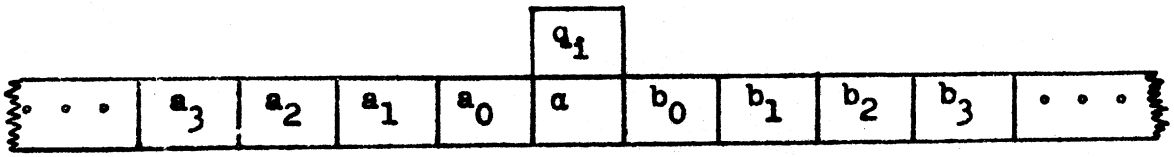
Part §1 of the report gives a very simple proof of the basic theorem II of ref. [1], that any Turing machine has a representation as a Tag system. The result is that we obtain systems with deletion number $p = 2$ (see §1 below), an improvement over the result $p = 6$ in ref. [1]. The simplification over the complicated methods of [1] is due in part to a better representation of the Turing machine, suggested by Dana Scott, and in part to a system of Post productions which yields the $p = 2$ result directly. The result of §1 is due jointly to John Cocke and the writer.

The simpler structure of the production rules enabled us to simplify both the structure of, and the encoding for, the 6-state, 6-symbol Universal Turing machine reported in [2]. This is described in part §2 of the report.

Part §3 is just a note simplifying (?) the basis for recursive functions developed in [1].

§1. Turing Machines and p = 2 Tag systems.

An instantaneous description of a Turing machine computation includes the contents of the Tape, the location of the machine on the tape, and the machine's internal state. The situation can be represented as



Following a suggestion of Dana Scott, we can regard the description as composed of four numbers; a, q_i, $m = \sum a_i 2^i$, and $n = \sum b_i 2^i$. We assume that the Machine is binary, so that the a_i's and b_i's are all ones and zeros. All but a finite number of the a_i's and b_i's are zero, so the summations are defined.

A Turing machine T is a set of quintuples $q_i, s_j : q_{ij}, s_{ij}, d_{ij}$. Suppose that d_{ij} happens to mean "move right". Then this quintuple has the following effect on $\{a, q_i, m, n\}$

1. change m to $2m + s_{ij}$
2. if $n = 2p + b_0$, change n to p and remember b_0 which is the next symbol to be read. P is $[\frac{n}{2}]$.
3. control is transferred to the quintuple starting with $\{q_{ij}, b_0\}$.

That is

$$\{q_i, s_j, m, n\} \rightarrow \{q_{ij}, n-2\left[\frac{n}{2}\right], 2m + s_{ij}, \left[\frac{n}{2}\right]\}$$

where $[x]$ is the largest integer in x. If d_{ij} is "move left" then the roles of m and n are interchanged.

The Turing Machine is thus equivalent to a programmed computer which executes instructions I_{ij} which perform the above operation on m and n. Each instruction is followed by transfer of control conditional on the value (0 or 1) of the symbol read in phase 2 above. This computer has 2 (infinite) data registers, for m and n, and an instruction location register to remember the pair q_i, s_j.

It is quite easy to represent this computer in a "bi-deletion" ($p = 2$) Post Tag system. See [1]. The system operates on a string of symbols which contains unary representations of the numbers m and n. The instruction location is represented by the choice of letters composing that string. For each instruction we will employ a different alphabet, distinguished by subscripts.

At the start of the k-th instruction I_k corresponding to $\{q_i, s_j\}$ the working string will have the form

$$A_k\overline{\Phi}(x_k\overline{\Phi})^m\ B_k\overline{\Phi}(y_k\overline{\Phi})^n$$

where the $\overline{\Phi}$'s represent symbols which can never affect the process. An operation of phase (1) is represented by Tag productions of the form ($P = 2$)

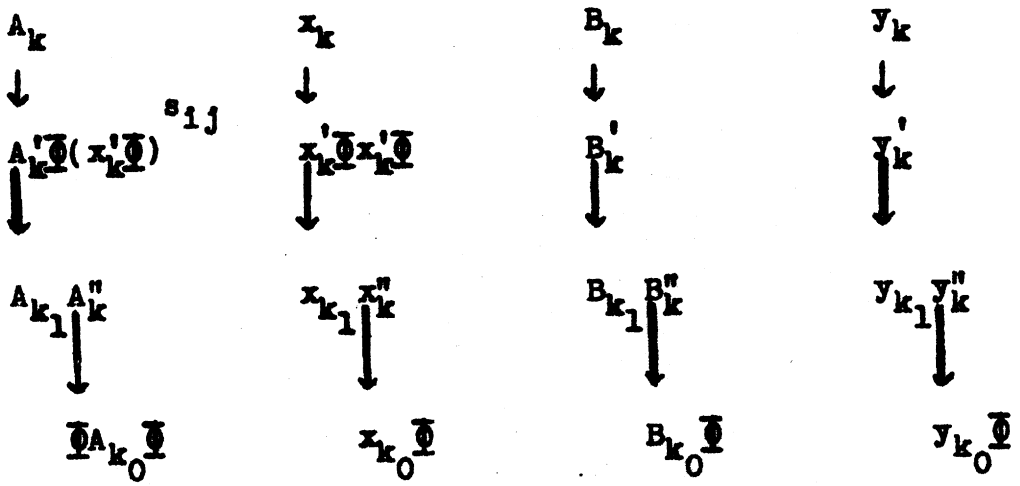
$$\begin{array}{cccc} A_k & & x_k & B_k & y_k \\ \downarrow & & \downarrow & \downarrow & \downarrow \\ A_k'\overline{\Phi}(x_k\overline{\Phi})^{s_{ij}} & & x_k'\overline{\Phi}x_k\overline{\Phi} & B_k' & y_k' \end{array}$$

so that

$$A_k\overline{\Phi}(x_k\overline{\Phi})^m\ B_k\overline{\Phi}(y_k\overline{\Phi})^n \rightarrow A_k'\overline{\Phi}(x_k'\overline{\Phi})^{2m+s_{ij}}\ B_k'(y_k')^n \quad .$$

Note that we have already begun the division of n by 2. This is the beginning of phase (1).

The outcome of phase (2) has to depend on whether n was even or odd, i.e., on whether the new symbol b_0 is zero or one. Suppose that for n even we are to transfer to the k_0-th instruction and for n odd to the k_1-th instruction*. Then



is a set of productions which does what we want. To check this we trace through two cases; for simplicity let the s_{ij} associated with this quintuple be zero. If n is odd then $n = 2p + 1$:

$$A\Phi(x\Phi)^m\ B\Phi(y\Phi)^{2p+1}$$

$$\rightarrow A'\Phi(x'\Phi)^{2m}\ B'(y')^{2p+1}$$

$$\rightarrow A_{k_1}A''_k(x_{k_1}x''_k)^{2m}\ B_{k_1}B''_k(y_{k_1}y''_k)^p$$

Since the string is composed of even-length blocks, the double-primed letters can never affect the process and the string is equivalent to

$$A_{k_1}\Phi(x_{k_1}\Phi)^{2m}\ B_{k_1}\Phi(y_{k_1}\Phi)^p$$

---

* If $k = \{q_i, s_j\}$ then $k_0 = \{q_{ij}, 0\}$ and $k_1 = \{q_{ij}, 1\}$

and control is transferred to the productions governing the k_1-th alphabet.

For even n , we have

$$A\overline{\Phi}(x\overline{\Phi})^m\ B\overline{\Phi}(y\overline{\Phi})^{2p}$$

$$\rightarrow A'\overline{\Phi}(x'\overline{\Phi})^{2m}\ B'(y')^{2p}$$

$$\rightarrow A_{k_1}\ A''(x_k\ x'')^{2m}\ B_{k_1}B''(y_{k_1}\ y'')^{p\overline{\Phi}}\ \underline{A_{k_0}\overline{\Phi}(x_{k_0}\overline{\Phi})^{2m}\ B_{k_0}\overline{\Phi}(y_{k_0}\overline{\Phi})^{p}}$$

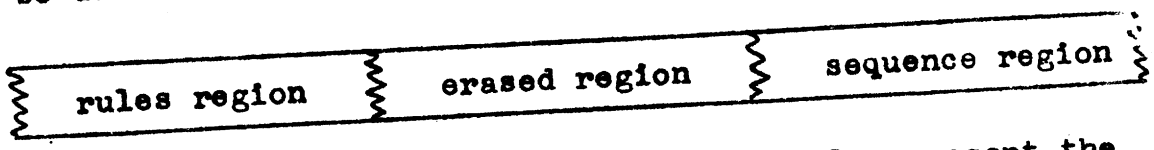
and the correct new values of m and n are delivered to the productions governing the k_0-th alphabet.(1)

The representation of a Turing machine as a 2-register computer is a variant of that in Dana Scott's sketch(2) of an elegant proof of Theorem I of [1]. In that paper we used a kind of second-order arithmetization so that the Turing machine condition could be represented in the prime factorization of a single number. John Cocke suggested that one ought to be able to manipulate the two quantities of theorem Ia of [1] directly, and by using the "phase-shifting" device buried in the lower left of the present production diagram he and I were able to reduce the deletion number to 2, as shown above. Putting these ideas together eliminated the two major complications of [1], namely the factorization in Sec. 1.2 and the division algorithm in Sec. 2.1 of [1]. The demonstration in [1], that there are universal two-tape non-writing machines, can be reconstructed easily from the above.

(1) Observe that each $(q_i s_j) \rightarrow$ 12 productions, and by some merging, each $q_i \rightarrow$ 13 prods.
(2) Personal communication.

§2. Coding for the 4 x 7 Universal Turing Machine.

It follows from §1 that we can make a Universal machine if we can simulate the application of an arbitrary Tag system with deletion number 2 to an arbitrary sequence of symbols. The different symbols will be represented by different blocks made up of the four symbols available to our machine. The tape of this machine will be divided into three regions:



We will use four symbols 0, 1, X, and B. 0 will represent the blank squares of the tape. The productions will be represented by an arrangement of 0's and 1's in the "rules region" and the working sequence will be represented by an arrangement of X's and B's in the "sequence region".

Let the letters of the alphabet of the Tag system of §1 be $A_1, \ldots, A_n$ and let the production rules have the form $[A_j \to A_{j1}A_{j2}\cdots A_{jr(j)}]$. (From the construction in §1 we know that $1 \le r(j) \le 4$ but we do not use this fact.) We will represent these in the "rules region" as strings of 0's and 1's as follows.

Define

$$\overline{A_j} = j + \sum_{i=1}^{j-1} r(i) = 1 + \sum_{i=1}^{j-1} [r(i) + 1]$$

Now for each letter A_j we will have a rule string R_j of the form

$$R_j = 110^{\overline{A_{jr(j)}}}1\ldots10^{\overline{A_{j2}}}10^{\overline{A_{j1}}}1$$

where 0^n means n zeros. We concatenate these in descending order to form the representation of the rules:

$$R = R_n R_{n-1} \cdots R_1 .$$

When a symbol is encountered in the sequence region, it will be necessary for the Turing machine to locate, in the rules region, the corresponding rule. The use of the representation $\overline{A_j}$ for a symbol makes this location easy, for the value of $\overline{A_j}$ gives (almost) the number of 1's that the machine must traverse to reach the appropriate rule (moving from the right). The double 1 1 at the left marks the end of the rule. The counting process used in locating the rule does not count the left one of this pair so that it sees just r(i) + 1 ones in the counting. The symbol has one extra 0 because the copying process which places the new word at the end of the sequence region skips the first 0 of each symbol copied.

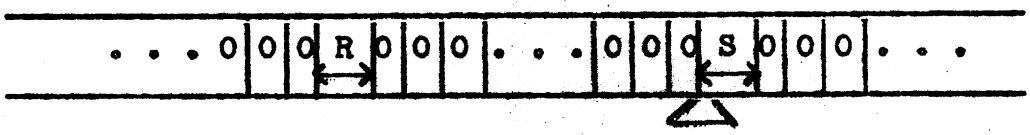
Finally we must represent the working string in the sequence region. This is marked as a string of X's and B's. If the sequence is $A_{k_1} A_{k_2} \cdots A_{k_s}$ then it is represented in the sequence region, from left to right now, as

$$S = X^{\overline{A_{k_1}}-1} B X^{\overline{A_{k_2}}-1} B \cdots B X^{\overline{A_{k_s}}-1} .$$

The symbol A_1 will have $\overline{A_1} = 1$ and $\overline{A_1} - 1$ will be 0. This symbol must not be used in regular operation (because it will cause 2 consecutive B's in region s) but, as seen below, can be used as a halting symbol with the production $a_1 \rightarrow a_1 a_1 a_1$.

Finally the "erased region", where initial parts of the working sequence have been deleted (we do not shift the sequence to the left between applications of the rules), is just an interval of 0's (blanks).

So the entire tape has the form



and the machine's operation begins with the reading head centered on the first mark (an X) at the left of the sequence (S).

Example of encoding:

Suppose the productions are

$$a_1 \to a_1\ a_1\ a_1$$

$$a_2 \to a_2\ a_2$$

$$a_3 \to a_3\ a_3\ a_2\ a_3$$

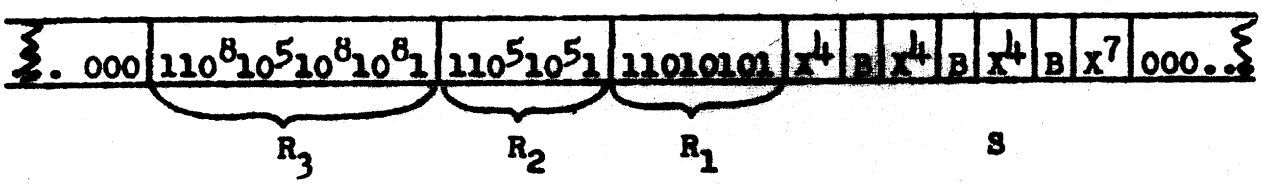
Then $r(1) = 3$, $r(2) = 2$, $r(3) = 4$,

$\overline{a_1} = 1$, $\overline{a_2} = 5$, $\overline{a_3} = 8$,

and

$$R_1 = 11010101$$
$$R_2 = 110^5 10^5 1$$
$$R_3 = 110^8 10^5 10^8 10^8 1 .$$

If the starting sequence is $a_2 a_2 a_2 a_3$, the machine's initial tape will be



Operation of the Universal Turing Machine.

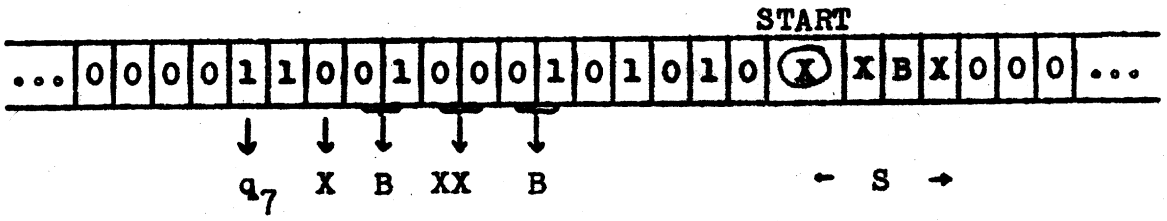
When started, the machine moves back and forth counting X's to the right and 1's to the left. When it encounters the first B to the right it will have erased the first symbol A_{k_1} of S, and it will have <u>located</u> to the left the rule R_{k_1} associated with that symbol. It then goes into another back-and-forth mode which <u>copies</u> the rule word at the extreme right of S. It works from the inside out, which is why the sequences are written left to right in S but right to left in R. Each 0 in an R yields an X in S, and each 1 in R yields a B in S. (When the latter occurs, an extra 0 is erased in R, accounting for the difference of one zero in the representations on the two sides.)

When the '11' is encountered in R, copying stops and the machine <u>restores</u> the tape to its initial configuration, removing the place markers set up in the locate and copy phases. This restitution process, almost by magic, erases the next word in the sequence region S, thus executing the process with deletion number 2. In the operation of the machine, one will note some manipulations exchanging 1's and B's; these are tricks by which we mark locations without using new symbols.

Below is the state transition table for the machine. The triples associated with each symbol-state pair are: [<u>symbol written</u>, <u>direction of motion</u>, and <u>new state</u> (if different)]. The machine starts in state 1 on the first X in region (S).

	q_1	q_2	q_3	q_4	q_5	q_6	q_7
X	OL	OL_{q1}	XL	XL	XR	XR	OR
0	OL	XR	HALT	XR_{q5}	XL_{q3}	BL_{q3}	XR_{q6}
I	IL_{q2}	BR	BL	IL_{q7}	BR	BR	IR
B	IL	XL_{q3}	IL_{q4}	IL	IR	IR	OR_{q2}

The following sequence demonstrates most features of the machine's operation.



This sequence contains one genuine rule string - 110010001 - which when copied produces the sequence BXXBX . The two 1's in the center of the example are skipped over in reading, and are there to illustrate how the rule is located--precisely by skipping over two 1's. When the string is copied, the original is erased, so this example keeps cycling and never halts.

The Halting Condition

The machine will halt if it is in state q_3 and encounters a 0 . This will occur if there are two consecutive B's in S encountered in the locating phase. We can define a letter A_H in the alphabet of the formal system to be a Halting symbol which stops the generation of sequences (e.g., by providing no rule for operating on it). In the Turing machine representation this is easily represented by assigning to the symbol A_H the R-sequence

1 1 0 1 0 1

which will cause two B's to be written consecutively at the end of S . When the machine encounters this it will stop. It is a little hard to insure that this will be hit in the correct state, so that it may be simpler to guarantee this by writing 3 B's, e.g., by using for A_H the sequence 1 1 0 1 0 1 0 1. For this we should take $r(H) = 3$. It is convenient to let A_H be a_1 .

For a general discussion of small Universal Machines, see [2].

§3. The Reduction of Machine Computations to Programs Involving Two Arithmetic Instructions Operators on Two Registers.

Theorem I_a of (1) states that

> we can represent any partial recursive function $T(n)$ by a program which operates on two integer variables S_1 and S_2; this program is composed entirely of instructions of the four types
>
> (A1) Add 1 to S_1. Go to I_j.
>
> (A2) Add 1 to S_2. Go to I_j.
>
> (B1) If $S_1 > 0$, subtract 1 from S_1 and go to I_{j1}; otherwise go to I_{j2}.
>
> (B2) If $S_2 > 0$, subtract 1 from S_2 and go to I_{j1}; otherwise go to I_{j2}.

By a <u>partial recursive function</u> we mean, of course, a mapping from some integers to other integers defined by a Turing machine together with an interpretation of some of its tape configurations as representations of integers. If $T(n)$ is such a partial function the cited theorem states that there is such a program which, if started at an initial instruction I_1 with S_1 set at 2^n and S_2 set at 0, then (if $T(n)$ is defined) the program will halt at a terminal instruction I_z with $S_1 = 2^{T(n)}$ and $S_2 = 0$. (The question is open, and presumably negative, as to whether one can do better with programs of such instructions so as to eliminate the exponent in the representation.)

The theorem cited above concerns programs made of four instruction

types. We can reduce this to three, in effect by replacing types A2 and B2 by an exchange operation which switches S_1 and S_2. We can reduce the basis to two by merging the exchange operation with A1 and B1 as follows.

We define new add and conditional subtract instructions:

(A) Add 1 to S_1, exchange S_1 and S_2, and go to I_j.

(B) If $S_1 > 0$, subtract 1 from S_1, exchange S_1 and S_2, and go to I_{j1}; otherwise exchange S_1 and S_2 and go to I_{j2}.

Now consider the effect of a linear sequence (subroutine) of the form

$$A \rightarrow A \rightarrow A \rightarrow B \rightrightarrows I_j .$$

This will add 1 to S_1, then add 1 to S_2, then add another 1 to S_1 and then subtract 1 from S_2. The latter subtraction must occur since the second add assures us that S_2 will not be zero at the time of the fourth operation. Thus the net result amounts to an operation of the form of the original A1, except that we have added 2 to S_1 instead of just 1.

Similarly, it can be seen that the subroutine AABA

$$A \rightarrow A \rightarrow B \rightarrow A \rightarrow I_j$$

has the effect of A2, again with an increment of 2 instead of 1.

Now consider the sequence

$$A \rightarrow B \rightrightarrows B \rightrightarrows B \begin{matrix} \nearrow I_{j1} \\ \searrow I_{j2} \end{matrix}$$

This leaves S_1 invariant in any case. We make both branches from the second instruction go to the third. Then if S_2 is zero the

program exits from the second branch without changing anything. If $S_2 \geq 2$ then S_2 is decremented by 2 and the prgram leaves by the first branch. So unless S_2 happens to equal 1, this routine is like B2 except for decrementing by 2. Similarly

$$B \rightrightarrows A \rightarrow B \begin{cases} \nearrow B \rightrightarrows I_{j1} \\ \searrow B \rightrightarrows I_{j2} \end{cases}$$

is like B1.

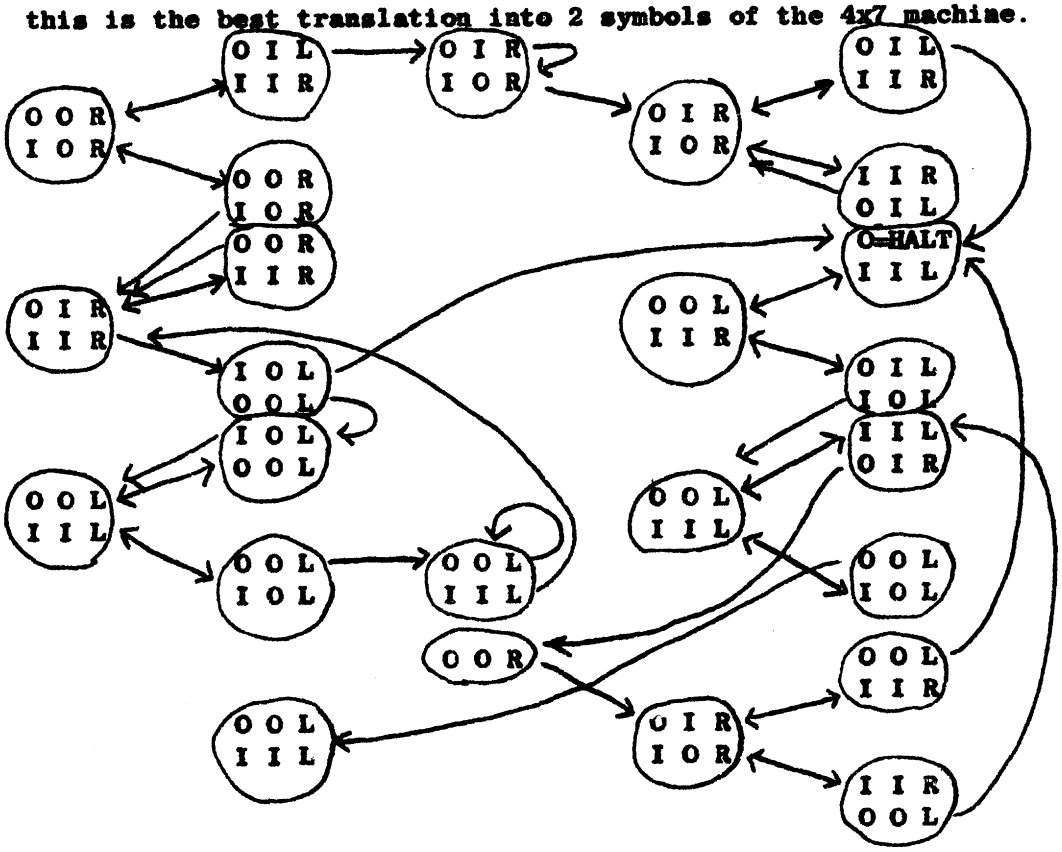
If we note that the numbers are changed by two in every case, we see that if we restrict S_1 and S_2 to be even numbers then the system (A A A B, A A B A, A B B B, B A B B) is isomorphic to the old system (A1, A2, B1, B2).

I wish to acknowledge the value of discussions with A. R. Tritter in connection with the work reported here.

IV. A 2-symbol 25-state Universal machine.

The diagram below gives the structure of a 2x25 machine obtained from the 4x7 machine by encoding the symbols of the latter in two-digit blocks:
0 --- 00 ; 1 --- 01 ; X --- 10 ; B --- 11.

The translation from the 4x7 machine is straightforward, with the same tape encoding. Each of the seven states is replaced by a three-state decoding tree. In some of these trees it was possible to encode the full quintuples, but in others additional states were needed. There is no reason to suppose this is the best translation into 2 symbols of the 4x7 machine.



REFERENCES

(1) M. Minsky, "Recursive Unsolvability of Post's Problem of Tag," Ann. Math. Vol. 74, no. 3, Nov. 1961.

(2) M. Minsky, "Size and Structure of Universal Turing Machines Using Tag Systems," AMS Symposia on Pure Mathematics, Vol. 5, 1961 (in press).

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-33

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 16(20-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☒ Single-sided or
☐ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-16) UNN'ED TITLE PAGE, 2-14, UNN'ED FIG, 15 (17-20) SCAN CONTROL, TRGT'S (?)	

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 12/12/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**