WANG, H.
Math. Annalen 152, 65—74 (1963)

Tag Systems and Lag Systems*

By

HAO WANG in Cambridge (Mass.)

A combinatorial system in the most general sense would be any finite set of rules each of which effectively produces a finite set of conclusions from a finite set of premises. The most intensively studied case is the one in which each rule has a single premise and a single conclusion. Such a system is called monogenic if the rules are such that for any string at most one rule is applicable.

From this broad class of monogenic systems, POST [3] chooses to consider the tag systems. A tag system is determined by a finite set of rules:

$$T_i\colon s_i \longrightarrow E_i\,, \qquad i = 1, \ldots, \varrho\,,$$

such that if the first symbol of a string is s_i, then the first β symbols are removed and the string E_i is appended at the end. Since the system is monogenic, $s_i \neq s_j$ when $i \neq j$. If the alphabet contains σ symbols, then $\varrho \leqq \sigma$.

Another natural class is, for want of a better name, the lag systems. A lag system is a set of $\leqq \sigma^\beta$ rules:

$$L_i\colon s_{i_1} \ldots s_{i_\beta} \longrightarrow E_i\,,$$

such that if the first β symbols of a string are $s_{i_1} \ldots s_{i_\beta}$, the first symbol, viz., s_{i_1}, is deleted and E_i is appended at the end of the string. In either kind of system, E_i is permitted to be the null string. Let ε_i be the length of E_i, ε be the maximum and ε^- be the minimum of ε_i. Thus, each system is associated with three constants, β, ε, σ, which, in general, appear to be of decreasing importance in that order. Clearly, when $\beta = 1$, tag systems and lag systems coincide. In general, lag systems are less wasteful since no symbol in a string is overlooked.

We shall establish, in a sense to be specified, that certain tag systems and lag systems are "undecidable", and all "simpler" ones are decidable. The undecidable tag system is a slight improvement over the one constructed by JOHN COCKE and MINSKY [2] with $\beta = 2$, $\varepsilon = 4$, to one with $\beta = 2$, $\varepsilon = 3$.

1. All Monotone Systems Are Decidable

With regard to each (tag or lag) system, there is a halting problem and a derivability problem. A system halts on a given string S if S ever leads to a string Q whose length $|Q| < \beta$, or to which no rule in the given system is applicable. The halting problem is to give a general method to decide, for each given string from the alphabet of the system, whether the system halts on the

* The work for this paper was supported by Bell Telephone Laboratories, Murray Hill, New Jersey.

string. The derivability problem is to decide, for any two given strings, whether the rules will lead us from the first to the second.

If we follow POST [3] in requiring $\varrho = \sigma$ for tag systems, and analogously $\varrho = \sigma^\beta$ for lag systems, then, since there are only finitely many strings of length less than β, a positive solution of the derivability problem yields one for the halting problem, and a negative solution of the latter yields one for the former. In permitting $\varrho < \sigma$ or σ^β, we do not have such a simple connection between the two problems. We shall confine ourselves to the more restricted tag and lag systems.

It is quite evident that the decision problems become complex only when some rules expand a string while others contract it. This remark can be stated and justified more exactly in two theorems.

Theorem 1. *For any given tag system T, if $\beta \geqq \varepsilon$ or $\beta \leqq \varepsilon^-$, then the derivability (and hence also the halting) problem for T is decidable.*

Theorem 2. *For any given lag system L, if $\varepsilon \leqq 1$ or $\varepsilon^- \geqq 1$, then the derivability (and hence also the halting) problem for L is decidable. This includes all lag systems in which no E_i is the null string.*

The proofs are similar. We give only a proof of Theorem 1.

Let $\beta \leqq \varepsilon^-$ and S be a given string. If $\beta < \varepsilon^-$, then, for each $n \geqq |S|$, there is at most a single consequence of S by T that is of length n, and there is no consequence shorter than S. Suppose, for some i, $\beta = \varepsilon_i$. Each time such a "stable" rule is applied, the length of the sequence does not change. Let $|S| = a$, $|Q| = b$, $b - a = c$. If c is negative, then of course Q is not a consequence of S. Otherwise, we write out the successive consequences of S one by one, $S = S_0, S_1, S_2$, etc., until we obtain either a repetition or a string longer than Q. Clearly $|S_i| \leqq |S_{i+1}|$, for all i. Moreover, for each fixed length k, there can be at most σ^k strings of length k. Hence, the process must always terminate. If $S_p = S_q$, $q > p$, and Q is not among $S_0, \ldots, S_{q-1}$, then Q is not a consequence of S because $S_p, \ldots, S_{q-1}$ will repeat and all consequences are contained in $S_0, \ldots, S_{q-1}$. If now $|S_t| > Q$ and Q is not among $S_0, \ldots, S_t$, then again Q is not a consequence of S.

Suppose $\beta \geqq \varepsilon$ and S is a given string. The argument is similar except that in this case we can list, once and for all, all consequences of S. Thus, either we get $S_q = S_p$, $p < q$, then $S_0, \ldots, S_{q-1}$ are all the consequences; or else, we get S_t which is the null sequence, and then $S_0, \ldots, S_{t-1}$ are all the consequences.

2. Every System with $\beta = 1$ is Decidable

Since there is no distinction between tag systems and lag systems for $\beta = 1$, we shall speak only in terms of tag systems. We assume therefore a tag system T with rules:

$$R_i\colon\ s_i \longrightarrow E_i\,, \qquad\qquad i = 1, \ldots, \sigma\,.$$

Since $\beta = 1$, the only contraction rules are those which produce the null string. By Theorem 1, we only have to consider the case when there are contraction rules.

2.1. *Definition of ranks.* If E_i is null, the rule R_i and the symbol s_i are of rank 1. If E_i is not null, but every symbol in E_i is of finite rank, then the rank of s_i and R_i is $n + 1$, n being the maximum of the ranks of the symbols in E_i.

If a rule R_i does not get a rank in the above manner, then R_i and s_i are said to have an infinite rank. Clearly:

2.2. If every rule of T has a finite rank, then every string S has only finitely many consequences. If some rules have the infinite rank but S contains only symbols with finite ranks, S again has only finitely many consequences.

It may be noted that the halting problem is easy to decide. Thus, given a tag system T and a string S, let T^* be obtained from T by deleting first all rules of finite ranks, and then all symbols of finite ranks from the remaining rules. Similarly, let S^* be obtained from S by deleting all symbols of finite ranks. T halts on S if and only if S^* is null. Thus, if S^* is not null, S can keep on producing consequences by T^* and therefore by T. On the other hand, if S^* is null, then there can be only finitely many consequences.

To decide the derivability problem, we introduce more definitions.

2.3. Let R_i be of infinite rank. Consider the σ consequences $A_1, \ldots, A_\sigma$ of s_i, with all symbols of finite ranks deleted. If some A_j contains s_i as a proper part, then R_i and s_i are of finite degree. If at least one A_j contains at least one symbol of finite degree, then R_i and s_i are of finite degree. Otherwise, R_i is a circular rule and s_i is a circular symbol.

2.4. Given a string S and a tag system T, we have made a round if we have' operated on every symbol in the string. The next round takes the result of the previous round as the given string. If T contains η circular rules, a circular symbol s_i is periodic if beginning with s_i as the initial string, we arrive, after η or less rounds, at a string in which s_i is again the only symbol of infinite rank.

To clarify these definitions, we observe the following. If T contains no circular rules, then, beginning with a string S containing some symbols of infinite rank (other strings being trivial by 2.2), we must get a string with more symbols with infinite ranks, after at most σ rounds. Hence, if $|Q| = m$, either Q occurs among the first σm rounds of consequences of S, or Q is not a consequence of S. Moreover, even when T contains circular rules, if S contains symbols of finite degree, the considerations still hold. Hence, we have to consider only strings containing no symbols of finite degree in systems containing circular rules.

Given one such system T, there must be at least one periodic symbol s_i. Thus, beginning with any circular s_j, we can never encounter a symbol of finite degree because otherwise s_j itself would be of finite degree. Hence, at each stage there is exactly one circular symbol. Since there are only finitely many (certainly $\leq \sigma$) circular symbols, at least one of them must be a periodic symbol.

2.5. Beginning with a periodic symbol a_i as the initial string, we can always find in less than $2\sigma^2$ rounds two consequences S_p, S_q, such that $p \neq q$ but $S_p = S_q$.

The only complication is with symbols of finite ranks since at each stage there is exactly one symbol of infinite rank. If now, beginning from a_i, we come, after enough (say $t \leq \sigma$) rounds, for the first time to a string A_i containing a_i again, say $x_1 \ldots x_u a_i y_1 \ldots y_v$, then $x_1, \ldots, x_u, y_1, \ldots, y_v$ must all be of finite ranks. Call t the period of a_i. From A_i on, after each t rounds, we get another sequence which contains A_i as a (proper or improper) part, since a_i always produces A_i after t rounds. The maximum finite rank of the rules of T is $\leq \sigma t$, and if B_i is obtained from A_i after σt ($\leq \sigma^2$) rounds, then, B_i is again obtained after σt more rounds. This is so because, although B_i may be $C_i A_i D_i$, C_i and D_i can have no more effect after σt rounds and the result is entirely determined by A_i.

We are now ready to settle the principal case. Thus, T contains circular rules, and each symbol in S is either of finite rank or circular. We wish to decide whether an arbitrary string Q is a consequence of S. Note first that after at most σ rounds, we eliminate all the circular symbols which are not also periodic. Hence, if t is the least common multiple of the periods $t_1, \ldots, t_j$ of all the periodic elements, then, by 2.5, after $(2\sigma + 1)t$ rounds, we get a repetition and therefore the set of all consequences of S.

Hence we have proved:

Theorem 3. *The derivability problem for each tag system or lag system with $\beta = 1$ is decidable.*

3. Undecidable Lag Systems

We shall give a lag system with $\beta = \varepsilon = 2$ whose halting problem is undecidable, by using SS machines introduced in SHEPHERDSON-STURGIS [4]. They have shown that every Turing machine (in particular, a universal one) can be simulated by an SS machine on the alphabet $\{0, 1\}$. We shall give a procedure of simulating these SS machines.

An SS machine is a finite sequence of instructions each of which is of the following two types.

P_0, P_1: print 0 (or 1) at the right end of the string S and go to the next instruction.

$SD(k)$: scan and delete the leftmost symbol of S; if it is 0, go to the next instruction, otherwise go to instruction k; if S is null, halt.

Let $q_1, \ldots, q_n$ be the instructions of an SS machine working on strings from $\{0, 1\}$. If the initial string is $x_1 \ldots x_p$, we shall represent it by $b_1 x_1 \ldots x_p e_1$. At each stage, if the state or instruction is q_i, the working string is of the form $b_i x_1 \ldots x_p e_i$. Consider now any instruction q_i. For brevity, let $j = i + 1$.

For the n instructions, we have altogether $2n + 4$ symbols $b_1, \ldots, b_{n+1}$, $e_1, \ldots, e_{n+1}$, 0, 1.

Case 1. q_i is P_0. We wish to get from $b_i x_1 \ldots x_p e_i$ to $b_j x_1 \ldots x_p 0 e_j$.

The rules are:

(1) $$x_t x_{t+1} \longrightarrow x_{t+1} \qquad (x_t, x_{t+1} = 0 \text{ or } 1) ,$$

(2) $$b_i x \longrightarrow b_j x \qquad (x = 0 \text{ or } 1) ,$$

(3) $$e_i b_j \longrightarrow e_j$$

(4) $$x e_i \longrightarrow 0 \qquad (x = 0 \text{ or } 1) .$$

Hence, by (2), $b_i x_1 \ldots x_p e_i \longrightarrow x_1 \ldots x_p e_i b_j x_1$,

by (1), $\longrightarrow x_p e_i b_j x_1 \ldots x_p$,

by (4), $\longrightarrow e_i b_j x_1 \ldots x_p 0$,

by (3), $\longrightarrow b_j x_1 \ldots x_p 0 e_j$.

In order to cover the case $p = 0$, we add the rule:

(5) $b_i e_i \longrightarrow b_j 0$.

Hence, by (5), $b_i e_i \longrightarrow e_i b_j 0$,

by (3), $\longrightarrow b_j 0 e_j$.

Case 2. q_i is P_1. The rule (1) above plus:

(6) $b_i x \longrightarrow b_j x$,

(7) $e_i b_j \longrightarrow e_j$,

(8) $x e_i \longrightarrow 1$,

(9) $b_i e_i. \longrightarrow b_j 1$.

Case 3. q_i is $SD(k)$. We wish to get from $b_i x_1 \ldots x_p e_i$ to $b_j x_2 \ldots x_p e_j$ if $x_1 = 0$, to $b_k x_2 \ldots x_p e_k$ if $x_1 = 1$. The rules are, besides (1) above:

(10) $b_i 0 \longrightarrow b_j$,

(11) $b_i 1 \longrightarrow b_k$,

(12) $x e_i \longrightarrow$,

(13) $e_i b_j \longrightarrow e_j$,

(14) $e_i b_k \longrightarrow e_k$,

(14*) $b_i e_i \longrightarrow$ (redundant) .

The last rule covers the trivial case when the string is null. The SS machines halt then. For the nontrivial case, we have:

Case 3.1. $x_1 = 0$.

By (10), $b_i 0 x_2 \ldots x_p e_i \longrightarrow 0 x_2 \ldots x_p e_i b_j$,

by (1), $\longrightarrow x_p e_i b_j x_2 \ldots x_p$,

by (12), $\longrightarrow e_i b_j x_2 \ldots x_p$,

by (13), $\longrightarrow b_j x_2 \ldots x_p e_j$.

Case 3.2. $x_1 = 1$.

By (11), $b_i 1 x_2 \ldots x_p e_i \longrightarrow 1 x_2 \ldots x_p e_i b_k$,

by (1), $\longrightarrow x_p e_i b_k x_2 \ldots x_p$,

by (12), $\longrightarrow e_i b_k x_2 \ldots x_p$,

by (14), $\longrightarrow b_k x_2 \ldots x_p e_k$.

There are two remaining loose ends to tidy up. If $i = n$, then $j = i + 1 = n + 1$, and in $SD(k)$, we may have $k > n$. In these cases, the SS machine is to halt. Clearly when $k > n$ in $SD(k)$, we can always put $k = n + 1$. Hence, the necessary rule needed is to halt when we encounter $b_{n+1} x_1 \ldots x_p e_{n+1}$. This is taken care of by the following rules:

(15) $$b_{n+1} x \longrightarrow b_{n+1} ,$$

(16) $$x e_{n+1} \longrightarrow e_{n+1} ,$$

(17) $$e_{n+1} b_{n+1} \longrightarrow ,$$

(18) $$b_{n+1} e_{n+1} \longrightarrow .$$

Hence, by (15), $b_{n+1} x_1 \ldots x_p e_{n+1} \longrightarrow x_1 \ldots x_p e_{n+1} b_{n+1}$,

by (1), $\longrightarrow x_p e_{n+1} b_{n+1} x_2 \ldots x_p$,

by (16) and (17), $\longrightarrow b_{n+1} x_2 \ldots x_p e_{n+1}$,

repeating, $\longrightarrow b_{n+1} e_{n+1}$,

by (18), $\longrightarrow e_{n+1}$.

Hence, the lag system halts by definition.

The other loose end is that we have not used all the $(2n + 4)^2$ pairs of symbols in the rules, e.g., $e_i e_k$. This is a simple matter since such combinations of letters do not occur. We add simply:

(19) $cd \longrightarrow$, if cd does not begin any of the rules (1) to (18) .

Therefore, we have established:

Theorem 4. *There is a lag system with $\beta = \varepsilon = 2$, whose halting problem is unsolvable.*

By Theorem 3, lag systems with $\beta = 1$ are always decidable. By Theorem 2, when $\beta = 2$, a lag system is decidable if either no $\varepsilon_i \geqq 2$ or no $\varepsilon_i = 0$. Hence, if we disregard the less decisive factor, the number σ of symbols in the alphabet, the above result is the best possible.

4. Undecidable Tag Systems

We modify somewhat the proof in [2] to get the following theorem.

Theorem 5. *There is a tag system with $\beta = 2$, $\varepsilon = 3$, $\varepsilon^- = 1$, whose halting problem is unsolvable.*

By Theorem 3 and Theorem 1, we can say this is the best possible result. It is perhaps of interest to compare this with the undecidable lag system: while we need here $\varepsilon = 3$ rather than 2, we do not need rules with the null string as consequences.

In order to prove the theorem, we use a program formulation of Turing machines developed in [5]. Each machine is represented by a finite sequence of instructions (or states) of the following five types: M (mark), E (erase), L (shift left), R (shift right), Ck (transfer to instruction k if the square under scan is marked, otherwise go to next instruction). Compared with the proof in [2], the present version separates the different actions (shift, write, transfer) so that the argument is perhaps easier to follow.

Assume now we are given a universal Turing machine and its program, with the two symbols 0, 1 only. The halting problem is unsolvable with initial inputs confined to include only finitely many 1's. At instruction or state i, the whole configuration is represented by $(x_i x_i)^m s_i s_i (y_i y_i)^n$, where the minimum portion including all 1's and the symbol s_i under scan is $\ldots a_2 a_1 a_0 s b_0 b_1 b_2 \ldots$, and $m = \Sigma a_t 2^t$, $n = \Sigma b_t 2^t$. The exponention on $x_i x_i$ and $y_i y_i$ means repetition so that, e.g., $(x_i x_i)^0$ is the empty string. (Compare, e.g., [5], p. 66.)

For each state i, a corresponding alphabet is introduced with tag rules which produce $(x_j x_j)^{m'} s_j s_j (y_j y_j)^{n'}$ from $(x_i x_i)^m s_i s_i (y_i y_i)^n$, j being the next state. The simulation for all instructions of a kind is uniform. Except for Ck, j is always $i + 1$, although we could as well take any fixed j in each case, and Ck is the only kind of instruction that introduces branching.

(a) Mark: i.M. This leads from $(x_i x_i)^m s_i s_i (y_i y_i)^n$ to

$$(x_{i+1} x_{i+1})^m 1_{i+1} 1_{i+1} (y_{i+1} y_{i+1})^n .$$

$$x_i \longrightarrow x_{i+1} x_{i+1} ,$$

$$s_i \longrightarrow 1_{i+1} 1_{i+1} ,$$

$$y_i \longrightarrow y_{i+1} y_{i+1} .$$

(b) Erase: i.E. Similar, replace second rule by: $s_i \longrightarrow 0_{i+1} 0_{i+1}$.

(c) Conditional transfer: i.Ck. Given $(x_i x_i)^m s_i s_i (y_i y_i)^n$, this goes to $(x_{i+1} x_{i+1})^m s_{i+1} s_{i+1} (y_{i+1} y_{i+1})^n$ if s_i is 0_i, and to $(x_k x_k)^m s_k s_k (y_k y_k)^n$ otherwise.

This case gives a simple illustration of the "phase-shifting" device of [2]. The rules are:

(1) $x_i \longrightarrow t_i t_i$, $\quad 0_i \longrightarrow 0'_i$, $\quad 1_i \longrightarrow 1'_i 1'_i$, $\quad y_i \longrightarrow u_i u'_i$,

(2) $t_i \longrightarrow x'_i x''_i$,

(3) $0'_i \longrightarrow x'_i 0''_i 0''_i$, $\quad 1'_i \longrightarrow 1''_i 1''_i$, $\quad u_i \longrightarrow y'_i y''_i$, $\quad u'_i \longrightarrow y''_i y'_i$,

(4) $x'_i \longrightarrow x_k x_k$, $\quad x''_i \longrightarrow x_{i+1} x_{i+1}$, $\quad 0''_i \longrightarrow 0_{i+1} 0_{i+1}$, $\quad 1''_i \longrightarrow 1_k 1_k$,

$y'_i \longrightarrow y_k y_k$, $\quad y''_i \longrightarrow y_{i+1} y_{i+1}$.

Deductions.

When $s = 0$, $(x_i x_i)^m 0_i 0_i (y_i y_i)^n$,

by (1), $\longrightarrow (t_i t_i)^m 0'_i (u_i u'_i)^n$,

by (2), $\longrightarrow 0'_i u_i (u'_i u_i)^{n-1} u'_i x'_i (x''_i x'_i)^{m-1} x''_i$,

by (3), $\longrightarrow (x''_i x'_i)^m 0''_i 0''_i (y''_i y'_i)^n$,

by (4), $\longrightarrow (x_{i+1} x_{i+1})^m 0_{i+1} 0_{i+1} (y_{i+1} y_{i+1})^n$.

When $s = 1$,

$$\longrightarrow (x_i x_i)^m 1_i 1_i (y_i y_i)^n$$

$$\longrightarrow (t_i t_i)^m 1'_i 1'_i (u_i u'_i)^n$$

$$\longrightarrow (x'_i x''_i)^m 1''_i 1''_i (y'_i y''_i)^n$$

$$\longrightarrow (x_k x_k)^m 1_k 1_k (y_k y_k)^n .$$

(d) Shift right: i.R. Additional complication is necessary in this case when we reduce ε from 4 to 3. In particular, the doubling requires a bit of elementary arithmetic. Thus, if we wish to double $(aa)^m$ to get $(bb)^{2m}$, we may use the rules $a \longrightarrow ccd$, $c \longrightarrow bbb$, $d \longrightarrow bb$. When m is even, we do succeed because $(aa)^2$ gives $(uuv)^2$ and $(bb)^4$; when m is odd, the last (aa) gives uuv and $(bbbbb)$ by borrowing a symbol from the sequence following v. However, we can manage to cut off the very first b and then get $(bb)^{2m}$. In what follows we shall assume these simple calculations. Although the whole set of symbols and rules are introduced all for a given state i, we shall, to avoid excessive subscripts, omit references to i. We wish to get from $(x_i x_i)^m s_i s_i (y_i y_i)^n$ to $(x_{i+1}x_{i+1})^{2m+s} s'_{i+1} s'_{i+1} (y'y')^{[n/2]}$, where $s'_{i+1} = (n - [n/2])_{i+1}$. Observe that s and s' may each be 0 or 1. The alphabet: x_i, s_i (0_i and 1_i), y_i; $x_a x'_a$, s_a (0_a and 1_a), y_a, x_b, t, y_b, x_c, t_a, t'_a, y_c.

The rules:

(1) $x_i \longrightarrow x_a x_a x'_a$, $\quad s_i \longrightarrow s_a s_a$, $\quad y_i \longrightarrow y_a y_a$.

(2) $x_a \longrightarrow x_b x_b x_b$, $\quad x'_a \longrightarrow x_b x_b$.

(3) $0_a \longrightarrow t$, $\quad 1_a \longrightarrow x_b x_b t$, $\quad y_a \longrightarrow y_b$.

(4) $x_b \longrightarrow x_c x_c$.

(5) $t \longrightarrow t_a t'_a$, $\quad y_b \longrightarrow y_c y_c$.

(6) $x_c \longrightarrow x_{i+1} x_{i+1}$.

(7) $t_a \longrightarrow 1_{i+1} 1_{i+1}$, $\quad t'_a \longrightarrow x_{i+1} 0_{i+1} 0_{i+1}$, $\quad y_c \longrightarrow y_{i+1} y_{i+1}$.

By (1), $(x_i x_i)^m s_i s_i (y_i y_i)^n \longrightarrow (x_a x_a x'_a)^m s_a s_a (y_a y_a)^n$.

By (2), if m is even $\longrightarrow s_a s_a (y_a y_a)^n (x_b x_b)^{2m}$,

if m is odd $\longrightarrow s_a (y_a y_a)^n (x_b x_b)^{2m} x_b$.

By (3), in both cases $\longrightarrow (x_b x_b)^{2m+s} t y_b^n$.

By (4), $\longrightarrow t y_b^n (x_c x_c)^{2m+s}$.

At this stage, we have two cases according as n is even or odd.

n is even:
$$t y_b^n (x_c x_c)^{2m+s} \xrightarrow{(5)} (x_c x_c)^{2m+s-1} x_c t_a t'_a (y_c y_c)^{n/2},$$
$$\xrightarrow{(6)} t'_a (y_c y_c)^{n/2} (x_{i+1} x_{i+1})^{2m+s},$$
$$\xrightarrow{(7)} (x_{i+1} x_{i+1})^{2m+s} 0_{i+1} 0_{i+1} (y_{i+1} y_{i+1})^{n/2}.$$

n is odd:
$$t y_b^n (x_c x_c)^{2m+s} \xrightarrow{(5)} (x_c x_c)^{2m+s} t_a t'_a (y_c y_c)^{(n-1)/2},$$
$$\xrightarrow{(6)} t_a t'_a (y_c y_c)^{(n-1)/2} (x_{i+1} x_{i+1})^{2m+s},$$
$$\xrightarrow{(7)} (x_{i+1} x_{i+1})^{2m+s} 1_{i+1} 1_{i+1} (y_{i+1} y_{i+1})^{(n-1)/2}.$$

(e) Shift left: i.L. We desire:
$$(x_i x_i)^m s_i s_i (y_i y_i)^n \longrightarrow (x_{i+1} x_{i+1})^{[m/2]} s'_{i+1} s'_{i+1} (y_{i+1} y_{i+1})^{2n+s},$$
where $s = m - 2[m/2]$.

Rules:

(1) $x_i \longrightarrow x_a x_a$, $\quad s_i \longrightarrow s_a s_a$, $\quad y_i \longrightarrow y_a y_a y'_a$, $x_a \longrightarrow x_b x_b$, $s_a \longrightarrow s_b s_b$

(2) $y_a \longrightarrow y_b y_b y_b$, $\quad y'_a \longrightarrow y_b y_b$, $\quad x_b \longrightarrow x_c x_c$.

(3) $0_b \longrightarrow t$, $\quad 1_b \longrightarrow t y_c y_c$, $\quad y_b \longrightarrow y_c y_c$, $\quad x_c \longrightarrow x_d$.

(4) $t \longrightarrow t_1 t_2$, $\quad y_c \longrightarrow y_d y_d$.

(5) $x_d \longrightarrow x_{i+1} x_{i+1}$.

(6) $t_2 \longrightarrow x_{i+1} 0_{i+1} 0_{i+1}$, $t_1 \longrightarrow 1_{i+1} 1_{i+1}$, $y_d \longrightarrow y_{i+1} y_{i+1}$.

Deductions:

$$(x_i x_i)^m s_i s_i (y_i y_i)^n \longrightarrow (y_a y_a y'_a)^n (x_b x_b)^m s_b s_b ,$$
$$\longrightarrow s_b s_b (y_b y_b)^{2n} (x_c x_c)^m (n \text{ even}) ,$$

or
$$\longrightarrow s_b y_b (y_b y_b)^{2n} (x_c x_c)^m (n \text{ odd}) .$$

In both cases,
$$\longrightarrow t (y_c y_c)^{2n+s} x_d^m ,$$
$$\longrightarrow x_d^{m-1} t_1 t_2 (y_d y_d)^{2n+s} ,$$
$$\longrightarrow t_2 (y_d y_d)^{2n+s} (x_{i+1} x_{i+1})^{m/2} (m \text{ even}) ,$$
$$\longrightarrow (x_{i+1} x_{i+1})^{[m/2]} 0_{i+1} 0_{i+1} (y_{i+1} y_{i+1})^{2n+s} ,$$

or
$$\longrightarrow t_1 t_2 (y_d y_d)^{2n+s} (x_{i+1} x_{i+1})^{(m-1)/2} (m \text{ odd}) ,$$
$$\longrightarrow (x_{i+1} x_{i+1})^{[m/2]} 1_i 1_i (y_{i+1} y_{i+1})^{2n+s} .$$

To complete the argument, we assume that the machine simulated has n instructions $q_1, \ldots, q_n$ and rewrite Ck as $C(n+1)$, whenever $k > n$. Whenever we get to $n+1$ from q_n or by $C(n+1)$, the machine is to stop. We add simply a new symbol h and the rules:

$$x_{n+1} \longrightarrow h, \quad s_{n+1} \longrightarrow h, \quad y_{n+1} \longrightarrow h, \quad h \longrightarrow h .$$

This completes the proof of Theorem 5.

5. Monogenic Normal System ([3])

A normal system is any set of rules

$$N_i \colon B_i \longrightarrow E_i$$

such that, for each given string, if it is $B_i P$, it becomes $P E_i$ by the rule. It is monogenic if, for $i \neq j$, B_i is never B_j or B_j followed by some string. A different definition of monogenic system would be that any string can be broken up in at most one way into the B_i's. The two definitions are not quite the same, since the latter includes more. For example, if the alphabet is $\{0, 1\}$, and B_1 is 00, B_2 is 001, B_3 is 11. Given any string, when we encounter 001, we have a question of taking it to be $B_1 1$ or B_2, but this can be settled by determining whether there are an even or an odd number of consecutive 1's immediately following 00. For our purpose, it is convenient to use the narrower definition.

It is very easy to use the SS machines to give a new proof of the known fact that there are monogenic normal systems with an unsolvable halting problem. Thus, as before, take a universal SS machine with n instructions. The rules for the corresponding monogenic normal system are simply:

(1) $$0 \longrightarrow 0\,.$$

(2) $$1 \longrightarrow 1\,.$$

For each q_i which is P_0:

(3) $$b_i \longrightarrow b_{i+1}\,.$$

(4) $$e_i \longrightarrow 0e_{i+1}\,.$$

For each q_i which is P_1:

(5) $$b_i \longrightarrow b_{i+1}\,.$$

(6) $$e_i \longrightarrow 1e_{i+1}\,.$$

For each q_i which is $SD(k)$:

(7) $$b_i0 \longrightarrow e_{i+1}b_{i+1}\,.$$

(8) $$b_i1 \longrightarrow e_kb_k\,.$$

(9) $$e_ie_{i+1} \longrightarrow e_{i+1}\,.$$

(10) $$e_ie_k \longrightarrow e_k\,.$$

Observe that if we reverse the arrows in the above rules, we do not get a monogenic system. We may modify (7)—(10) to read: $b_i0 \longrightarrow e'_ib_{i+1}$, $b_i1 \longrightarrow e''_ib_k$, $e_ie'_i \longrightarrow e_{i+1}$, $e_ie''_i \longrightarrow e_k$. Nevertheless, we would still have 0 at the right hand side of (1), $0e_{i+1}$ at that of (4), 1 in (2), $1e_{i+1}$ in (6); in fact, e_{i+1} in (4) or (6) might be the same as e_k in (10).

References

[1] MINSKY, M. L.: Recursive unsolvability of Post's problem of Tag. Ann. Math. **74**, 437—455 (1961).

[2] — Universality of ($p = 2$) Tag systems. A. I. Memo No. 33, Cambridge, Mass. (1962). A modified version of this (Memo No. 52, April 1963) has been prepared by COCKE and MINSKY for publication. This new proof is such that a simple modification would yield Theorem 5 above.

[3] POST, E. L.: Formal reduction of the general combinatorial decision problem. Am. J. Math. **65**, 197—215 (1943).

[4] SHEPHERDSON, J. C., and H. E. STURGIS: The computability of partial recursive functions by forms of turing machines. Abstracts, International Congress for Logic, Methodology and Philosophy of Science, Stanford, California (1960), p. 17. The full paper has just appeared as "Computability of recursive functions", Journal of Association for Computing Machinery **10**, 217—255 (1963).

[5] WANG, HAO: A variant to Turing's theory of computing machines. Journal of Association for Computing Machinery **4**, 63—92 (1957).

[6] — Tag systems and lag systems. (Abstract) Notices of the Am. Math. Soc. **9**, 407 (1962). ALAN TRITTER has recently made use of results in [2] and [6] to obtain a universal Turing machine with 4 symbols and 6 states, the smallest at present.

(Received October 22, 1962)