EC-15, p 799-809, october 1966.

Short Notes

The Self-Diagnosability of a Computer

NARSINGH DEO

Abstract—Maximum capability for self-diagnosis with minimum additional hardware is the goal of every designer of a general purpose computer today. A yardstick with which the self-diagnosability of a system can be measured is proposed.

A self-diagnosable computer can be described as a system consisting of two interconnected but independent machines: the main processor M_1 and a much smaller machine M_2 (about 5 to 10 percent of the size of M_1), which is capable of (programmatically) detecting and locating a fault in M_1. This fault location should be pinpointed within a small number of replaceable modules (integrated circuit chips, parallel-plate packages, or printed circuit cards) [1], [2].

The most commonly employed technique for diagnosis is to prepare a list of a complete set of tests $T=\{T_1, T_2, \cdots, T_n\}$ such that every failure in the system will cause one or more of these tests to fail [2]–[4]. Let the set $F=\{F_1, F_2, \cdots, F_m\}$ represent all possible single failure cases in the system. By taking the intersection of the sets of suspects for the failing test cases $T_{i1}, T_{i2}, \cdots, T_{ir}$ one arrives at a fault F_i. Let $k_1, k_2, \cdots, k_m$ be the number of suspected modules under the faults $F_1, F_2, \cdots, F_m$, respectively. In other words, during the Maintenance Routine [3] run, if tests $T_{i1}, T_{i2}, \cdots, T_{ir}$ fail, and the rest of the tests pass, then from a look-up table we arrive at the conclusion that fault F_i has occurred, and in order to correct this fault F_i we have to either replace k_i number of modules or examine each of these k_i modules by some other means and replace the bad one.

Clearly then, if $N=$ total number of modules used in the machine,

$$\sum_{i=1}^{m} k_i \geq N. \tag{1}$$

Let p_i be the probability of occurrence of failure F_i, for $i=1, 2, \cdots, m$. Then assuming that at a given instance exactly one fault has occurred,

$$\sum_{i=1}^{m} p_i = 1. \tag{2}$$

Number $R_i=1/k_i$ is an indicator of the efficiency with which fault F_i can be repaired. The diagnostic efficiency of the entire system can be represented by

$$R = \frac{1}{\sum_{i=1}^{m} k_i p_i}. \tag{3}$$

This number R can be called the "resolution" of the entire system. The comparative figure of merit of a diagnostic subsystem is then

$$\frac{R}{\text{cost}} \tag{4}$$

where the cost includes the cost of hardware in M_2, of software, of development, and of running time of the maintenance routine.

If the maintenance routine only detects, and does not locate a fault, then R assumes its minimum possible value

$$R_{\min} = \frac{1}{N}. \tag{5}$$

This implies that one has to examine all N modules of the machine to locate the faulty module.

The resolution R is maximum when every failure can be traced down exactly to one module, i.e. $k_i=1$, for $1\leq i\leq n$. Then from (3), resolution becomes

$$R_{\max} = \frac{1}{\sum_{i=1}^{m} p_i} = 1. \tag{6}$$

If all modules are assumed to have equal probability of failure, then the probability of occurrence of failure F_i is given by

$$p_i = \frac{k_i}{\sum_{i=1}^{m} k_i} \tag{7}$$

and the resolution of the machine by substituting (7) in (3) turns out to be

$$R_e = \frac{\sum_{i=1}^{m} k_i}{\sum_{i=1}^{m} k_i^2}. \tag{8}$$

In absence of any statistical data available on the probability of various failures, (8) would be a good index of the diagnosability of a system.

In the author's opinion the resolution in (8) is a very important figure in the specification of any machine with diagnostic capability. The manufacturer should specify it, and the customer should ask for it. As discussed above, in general, R_e will have a value between 1 and $1/N$.

References

[1] R. E. Forbes, D. H. Rutherford, C. B. Stieglitz, and L. H. Tung, "A self-diagnosable computer," *Proc. F.J.C.C.*, vol. 27, pp. 1073–1086, November 1965.
[2] K. Malig and E. L. Allen, "A computer organization and programming system for automated maintenance," *IEEE Trans. on Electronic Computers*, vol. EC-12, pp. 887–895, December 1963.
[3] R. S. Ledley, *Digital Computer and Control Engineering*. New York: McGraw-Hill, 1960, pp. 135–139.
[4] J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the diagnosis of switching circuit failure," *IEEE Trans. on Communication and Electronics*, vol. 83, pp. 509–514, September 1964.

Manuscript received June 2, 1966.
The author is with Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Calif. He was formerly with Burroughs Corporation, Pasadena, Calif.

High Speed Binary Parallel Adder

HUEI LING

Introduction

The propagation of the carry is always a dominant problem in modern computers. Recently, Salter [1], Admodei [2], and Edwards [3], [4] increased the speed of the carry propagation by the use of tunnel diodes, or so-called "carry selecting switches." Instead of transmitting the carry signal as shown by Edwards and Salter, this short note presents a method of high speed addition by the use of a complementing signal.

Manuscript received September 24, 1965; revised December 16, 1965, and April 25, 1966.
The author is with the IBM Watson Research Center, Yorktown Heights, N. Y.

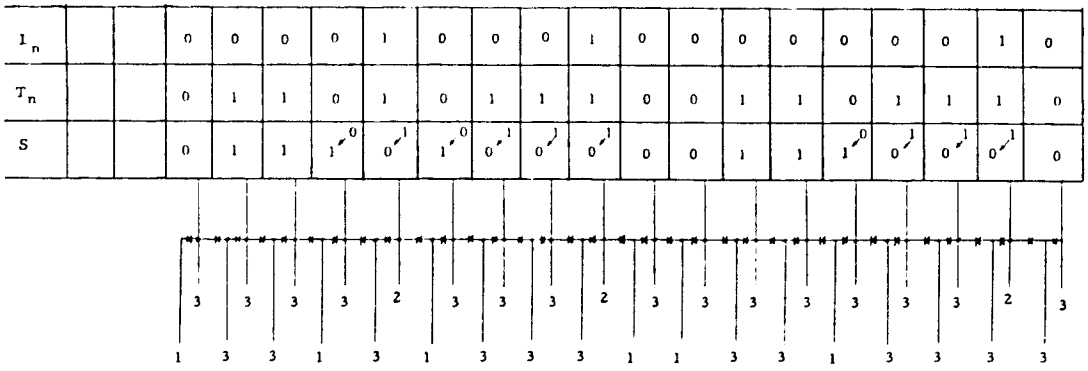

Fig. 1. The block diagram of Step IV. High speed binary parallel adder. 1) L_n and T_n both are zero, the inhibit circuit is on to prevent the complementing pulse from propagating down the line. 2) I_n and T_n both are 1, turn on the complementing pulse to invert the 0 to 1, and 1 to be 0. 3) Not in operation in this example. The diodes are used to prevent the complementing pulse from transmitting backward.

Basic Theory

Let A and B be the two numbers which are to be added together, where A and B can be represented as

$$A = a_n2^n + a_{n-1}2^{n-1} + \cdots + a_i2^i + \cdots + a_02^0$$
$$B = b_n2^n + b_{n-1}2^{n-1} + \cdots + b_i2^i + \cdots + b_02^0$$

where a_i's and b_i's are in binary and can be either 1 or 0. Its sum can be written as

$$S = 2^n(a_n + b_n) + 2^{n-1}(a_{n-1} + b_{n-1}) + \cdots + 2^0(a_0 + b_0) \quad (1)$$

or simply

$$S = \sum_0^n 2^n(a_n + b_n)$$

$$S = \sum_0^n \left[2^{n+1}a_nb_n + 2^n(a_nb_n' + a_n'b_n)\right] \quad (2)$$

where a_n, b_n, a_n' and b_n' are in binary and a_n' is the negation of a_n similarly for the b_n's.

If modulo 2 addition [5] is used, then the sum can be written as

$$S = \sum_0^n 2^{n+1}a_nb_n + \sum_0^n 2^n(a_n \oplus b_n). \quad (3)$$

Letting

$$T_n = \sum_0^n 2^{n+1}a_nb_n, \qquad P_n = \sum_0^n 2^n(a_n \oplus b_n)$$

(3) can be rewritten as

$$S = T_n + P_n. \quad (4)$$

Equation (4) indicates that the sum can be represented by the addition of (T_n+P_n) or, more simply, that the sum is performed by extraction of a_n and b_n with 1 bit left shifted, and adding the differences between a_n and b_n. This is another form of binary addition. The inherent property of (4) is that the addition of T_n and P_n will not contain the following cases:

$$\cdots 1\ 1\ 1 \cdots 1\ 0\ 1 \cdots 1\ 1\ 1\ 1 \cdots 1\ 1\ T_n$$

$$\cdots 1\ 0\ 1 \cdots 1\ 1\ 1 \cdots 1\ 1\ 1\ 1 \cdots 1\ 1\ P_n.$$

When a 1 occurs in P_n there will not be 1's in both T_{n+1} and T_{n-1} (the diagonal direction of P_n). It may possess a 1 in T_{n+1} or T_{n-1} but not in both, since the corresponding 1's have extracted each other followed with 1 bit left shifted, and have already been included in T_n.

In general, T_n and P_n will contain the following type of additions:

$$\cdots 1\ 1\ 1\ 1\ 1 \cdots 0\ 1 \cdots 0\ 0\ 1 \cdots 1\ 0\ 1\ 1\ 0\ T_n$$

$$\cdots 0\ 0\ 0\ 0\ 1 \cdots 0\ 1 \cdots 0\ 0\ 1 \cdots 1\ 0\ 0\ 1\ 0\ P_n.$$

This type of addition can be taken care of simply by putting all 1's of P_n into T_n and by then initiating the complementing pulse to invert the 1's or 0's in T_n which are needed to be inverted. The carry propagation in T_n and P_n addition has been realized by the simple inversion. The detailed explanation is shown in Fig. 1 and described below. The general circuit block diagram is shown in Fig. 2.

General Procedure of Circuit Design

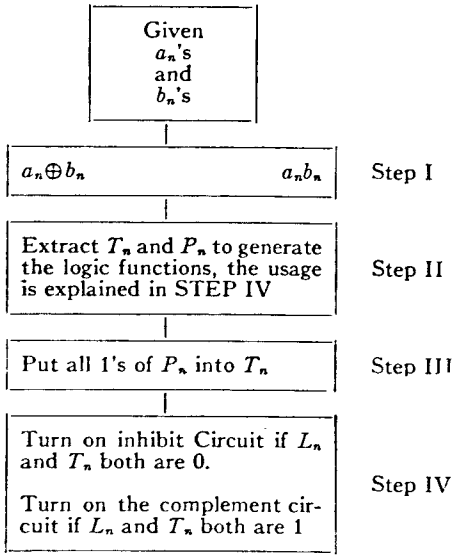

Fig. 2. Block diagram of circuit design.

Description

In order to explain the operating procedure step by step, an example is given. Let $A = 64281$, $B = 54615$; in binary, these numbers show as

$$A = 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1$$

$$B = 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1.$$

Step I

In order to generate $\sum 2^{n+1}a_nb_n$ A and B are extracted and followed by 1 bit left shifted.

$$T_n = 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0.$$

P_n is generated by finding the difference between A and B

$$P_n = \ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0$$

Step II

Generate the logic function L_n, the extraction between T_n and P_n. In this example, L_n is shown as

$$L_n = 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0.$$

The usage of this function is explained in Step IV

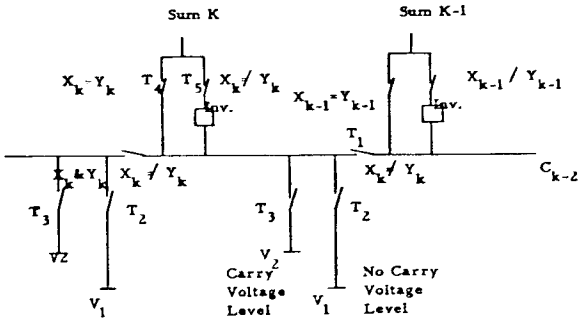

Fig. 3. Edwards' adder (logic design) (see [3], p. 465).

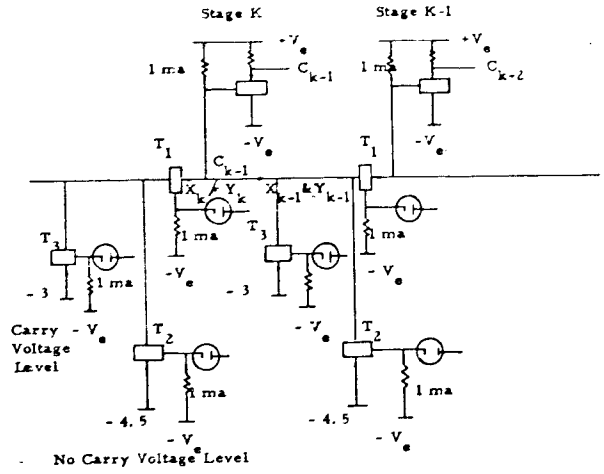

Fig. 4. Edwards' adder (carry control circuit) (see [3], p. 466).

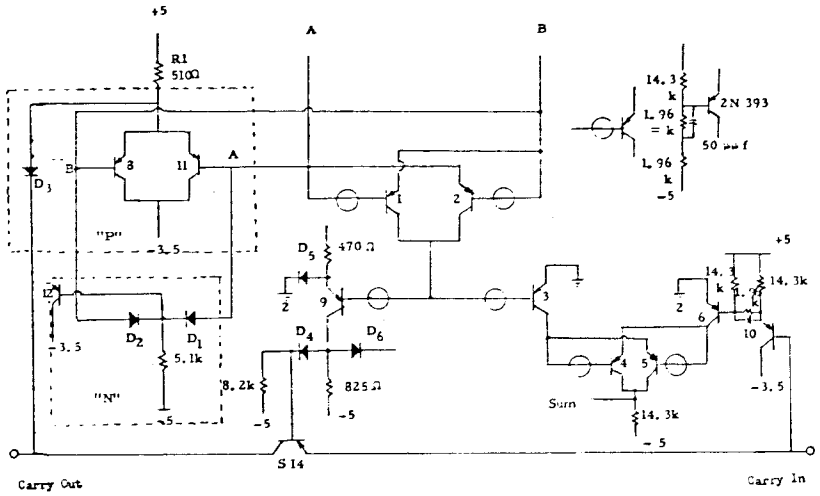

Fig. 5. Salter's adder (see [1], p. 462).

Step III

Put all 1's of P_n into T_n in the corresponding location. T_n and L_n then read

$$T_n = 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0$$
$$L_n = \ \ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0.$$

Step IV

Turn the inhibit circuits on if L_n and T_n equal 0. (After Step III, T_n contains original T_n and all 1's of P_n.) Turn the complementing pulse on if L_n and T_n both are 1. The former will hold the stages with no need to invert; the later will invert the stages which need to be inverted.

After completing Step IV, the sum shows

$$S = 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0$$

which is 118896 in decimal.

These four steps actually are formed simultaneously as soon as the a's and the b's are available. Equation (3) gives

$$S = \sum_{0}^{n} 2^{n+1}(a_n b_n) + \sum_{0}^{n} 2^n(a_n \oplus b_n).$$

If we write the sum S itself into the binary form, then

$$S = 2^{n+1}s_{n+1} + 2^n s_n + \cdots + 2^k s + \cdots + 2^0 s_0 \cdots. \quad (5)$$

By comparing (4) and (5), the value of the kth digit of the sum S can be expressed as

$$s_k = (a_{k-1}\ b_{k-1}) + (a_k \oplus b_k)$$

since we do not perform the actual addition here. We just put all 1's of P_n into T_n, therefore, s_k can be read as

$$s_k = (a_{k-1}\ b_{k-1}) \quad \text{or} \quad (a_k \oplus b_k). \quad (6)$$

By doing so, the complementing logic control circuit C_k and the inhibiting logic control circuit I_k should be added and are shown in (7) and (8).

$$C_k = (a_{k-1}\ b_{k-1}) \cdot (a_k \oplus b_k) \quad (7)$$
$$I_k = (a_{k-1}\ b_{k-1})' \cdot (a_k \oplus b_k)'. \quad (8)$$

Equations (6)–(8) show that s_k is formed simultaneously as soon as the a's and the b's are available; when the add instruction is executed, the logic control circuit C_k or I_k (not both) will be activated, and the sum is formed as shown in Fig. 1.

Comparison

In Edwards' adder the logical diagram of two adder stages is shown in Figs. 3 and 4. When X_k and Y_k both are 1, the switch T_2 is closed and a carry is generated. When X_k and Y_k both are 0, the switch T_3 is closed and the carry signal is inhibited. When X_k and Y_k are different, the switch T_1 is closed to admit the carry to propagate.

In Salter's adder as shown in Fig. 5, the carry signal is generated by the section "P" when A_k and B_k both are 1. The carry signal is inhibited by the section "N" when A_k and B_k both are 0. The carry selecting switch $S14$ is closed when A_k and B_k are different. Basically speaking these two adders are similar. The adder proposed here differs from these two as follows.

1) There is no carry selecting switch. Therefore, instead of transmitting the carry signal, the complementing signal is propagated.

2) In Salter's adder the value of the kth digit of the sum S is generated as $s_k = a_k \oplus b_k \oplus c_{k-1}$, which is controlled by transistors 1, 2, 3, 4, 5 and 6, as shown in Fig. 5. The s_k of the proposed adder is dependent upon a_{k-1}, b_{k-1}, a_k and b_k as expressed in (6) and controlled by the logic circuit as shown in (7) and (8).

References

[1] F. Salter, "High-speed transistorized adder for a digital computer," *IRE Trans. on Electronic Computers*, vol. EC-9, pp. 461–464, December 1960.
[2] J. J. Amodei, "High-speed adders and comparators using transistors and tunnel diodes," *IEEE Trans. on Electronic Computers*, vol. EC-13, pp. 563–575, October 1964.
[3] T. Kilburn, D. B. G. Edwards, and D. Aspinall, "Parallel addition in digital computers a new fast-carry circuit," *Proc. IEE (London)*, vol. 106, pt. B, pp. 464–466, September 1959.
[4] T. Kilburn, D. B. G. Edwards, and D. Aspinall, "A parallel arithmetic unit using a saturated-transistor fast-carry circuit," *Proc. IEE (London)*, vol. 107, pt. B, pp. 573–584, November 1960.
[5] G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*. New York: Macmillan, 1963.

The Conjectured Highest Scoring Machines for Rado's $\Sigma(k)$ for the Value $k=4$

ALLEN H. BRADY

Abstract—**A study of the output of a heuristic computer program reveals two four-state binary Turing machines which yield the highest known score for four states in Rado's co-called "Busy Beaver" logical game. There is evidence which supports the conjecture that this score of 13 is the particular value of $\Sigma(4)$, where Σ is a noncomputable integer function associated with this game. It is also conjectured that $S(4)=106$, where S is another noncomputable function, the maximum shift number, of interest in Rado's study. Complete solution of the problem for four states has been reduced to a relatively small set of machines.**

In a 1962 paper, Rado [1] defines a certain noncomputable integer function $\Sigma(k)$, where k is the number of states of a binary Turing machine and $\Sigma(k)$ is the least upper bound of the number of marks left on a blank input tape by any member of the (finite) class of such k-state machines which stop. Rado proposed a logical game (the "Busy Beaver Game") to find, for a given integer k, a k-state machine which will write a large number of marks on an (initially) all blank tape and then stop. The best obtained values for $k=2$ and 3, for instance, are 4 and 6, respectively. For these cases, however, it is known [2] that $\Sigma(2)=4$ and $\Sigma(3)=6$.

The determination of the value of $\Sigma(k)$ for a *particular* value of k essentially reduces to the solution of the halting problem for k-state machines with a blank input tape. An algorithm for making this decision does not, of course, exist for arbitrary k, since it is evidently equivalent to solving the halting problem for a universal Turing machine. In addition to the difficulty in determining whether or not a particular k-state machine will halt, there is also a hurdle posed by the sheer magnitude of the number of binary machines with k states. This number is of the order of $(6k)^{2k}$ and while not too large for $k=3$ is approximately 10^{11} for $k=4$. One can gain some feeling for the magnitude of this number by thinking of it in terms of cycles for a synchronous digital computer with, say, an eight microsecond cycle time: it represents eight continuous days of operation!

There are, of course, several immediate and obvious reductions which one can make in the size of the set of k-state machines to be considered, such as elimination of right-left symmetry and requiring the presence of at least one "halt" command. The primary economy, however, is effected in a tree generation "algorithm"[1] which eliminates the need to consider machines which are identical except for redundant commands or isomorphisms arising from permutations on the states. The tree generation process consists of starting in state 1 with a seed machine consisting of only the triple 1 R 2 (print a mark, move right, go into state 2) in the [1, S] (state one, scanning a blank space) entry of the description table.[2] The described machine is operated until either 1) it requires a new entry to be defined in the table or 2) it is determined that no undefined entry can be reached, i.e., the machine will never stop. Obviously, the seed machine will immediately require a table entry under [2, S] and we begin by inserting the triple S L1. The machine is operated as before. In the event it is determined that the machine will never stop, the triple in the most recently used entry is changed (increased) in a simple counting sequence. The maximum state to be used in the newly inserted triple cannot exceed by more than one the maximum state appearing in any previously entered triple. If by this rule alteration is not possible, then a retrace is made to the previous entry. We shall refer to any machine generated by this process as being in *tree-normal* form.

Strictly speaking, it is not required by the rules of the game that a machine start in state one, but it is obvious that any machine starting in a state m is isomorphic to some machine starting in state one, namely, the machine obtained by exchanging 1 for m throughout the machine description. Further, it is apparent that with a blank input tape any machine, except for redundant entries, is isomorphic to a machine in tree-normal form.

The tree generation process was included in a digital computer program which encompassed a heuristic solution to the halting problem [3] and all necessary bookkeeping connected with the game. The output of the program consisted of the highest scoring entries found for Rado's game and, in addition, a list of machines generated for which the halting problem was not solved. Nearly 550 000 machines were generated by the computer to exhaust the four-state case, with about 6000 machines remaining for which the halting problem was left undecided. (It was assumed by the program that they would never halt.) The remaining machines were each run on a 100 square blank tape for not more than 500 moves. Only two of these machines stopped. Their descriptions are given in Fig. 1.[3] Machine A stopped after 95 moves and machine B after 106 moves. Both machines leave 12 marks (1's) on their output tape. Using Rado's convention of a *halt state*, which is not counted as one of the k states, one can insert a command to print an additional mark and then stop. The final tape configurations are shown in Fig. 2. The result obtained is a "score" of 13[4] giving the relation $\Sigma(4)\geq 13$. We have further the relation $S(4)\geq 106$, where S is the noncomputable function determined by the maximum number of moves or *shift number* [1].

These machines yield the highest scores which are known to date for $k=4$ insofar as the author has been able to determine. All four-state machines known to have stopped do so within a tape region of 22 squares and in 106 or fewer moves. In addition, examination of samples of the remaining 6000 machines has revealed quite complex, but nevertheless not unfamiliar, patterns of degenerate behavior paralleling that seen in other machines known not to halt. From this evidence, it is conjectured that $\Sigma(4)=13$ and $S(4)=106$.

Manuscript received July 6, 1965; revised November 1, 1965. This work was supported in part by the Graduate Research Council of Oregon State University, Corvallis, Ore., and by the National Institute of Health under Grant GM 11178.

The author is with the Department of Computing Science, University of Notre Dame, Notre Dame, Ind.

[1] The process to be described is not an algorithm in the strict sense because of the dependency upon a solution to the halting problem; hence the quotes.

[2] One can show by means of a simple argument that no other entry in [1, S] need be considered. See Lin and Rado [2].

[3] The author is indebted to the referee for pointing out a most astounding coincidence: while the two machines of Fig. 1 are distinct in the tree-normal sense machine B is isomorphic to machine A if machine A is allowed to start in state 2. This fact is readily apparent from inspection of the state diagrams of the two machines. The referee further notes that, under this isomorphism, identical configurations are reached by the machines after 14 and 25 moves, respectively.

[4] To get the scores of 13 replace each "HALT" in Fig. 1 by the triple 1R0 (print "1," move right, go into state "0") where state 0 is the added *halt state*.