THE JOURNAL OF SYMBOLIC LOGIC
Volume 31, Number 2, June 1966

THE UNDECIDABILITY OF THE TURING MACHINE IMMORTALITY PROBLEM[1]

PHILIP K. HOOPER

Part I. Introduction. A *Turing Machine* (TM) is an abstract, synchronous, deterministic computer with a finite number of internal states. It operates on the set of infinite words, or tapes, over some finite alphabet, scanning exactly one symbol of the tape at a time. (Only a 2-symbol alphabet, consisting of "0" and "|", will be considered here, and the scanned symbol of a tape will be distinguished by an underscore.) depending upon its internal state and the symbol under scan, it can perform one or more of the following operations: replace the scanned symbol with a new symbol, focus its attention on an adjacent square, and transfer control to a new state.

The instructions for the TM are quintuples of the form $[q_i, s_j, s_k, D, q_l]$ where q_i and s_j represent the present state and scanned symbol, s_k the (possibly) new symbol to be printed, D the direction of motion of the scanner (L, R, or N for left-shift, right-shift, and no-shift), and q_l the new internal state. For consistency, no two quintuples may begin with the same two symbols. If the TM enters a state-symbol configuration for which there is no corresponding quintuple, it is said to halt.

A tape with designated scanned symbol and an internal state of the TM together constitute an *instantaneous description* (ID) of the TM. If an ID has a corresponding quintuple, the result of applying it is another ID, a successor of the original; otherwise, it is a terminal ID. Extending the relation of successor to its transitive completion, each ID with a terminal successor can be termed "mortal", the others, those that do not lead to terminal ID's but rather run for ever, as "immortal".

The classical result of Turing Machine theory is the unsolvability of the problem of determining, for a given ID, whether it is mortal or immortal, usually called the "halting problem". The "immortality problem" is the problem of deciding, for a given TM, whether or not there exists an immortal ID. This is essentially the (T_2) problem of Büchi [2], and its undecidability, proved below, provides a basis for an alternative proof that the $\forall\exists\forall$ prenex class forms a reduction class for the predicate calculus.

The somewhat special nature of the immortality problem arises from

---

Received June 7, 1965.

[1] This paper was presented to the Division of Engineering and Applied Physics of Harvard University in partial fulfillment of the requirements for the Ph. D. degree in applied mathematics, and the work was supported, in part, by the Bell Telephone Laboratories, of Murray Hill. The author wishes to express his gratitude to Professor Hao Wang for inspiring and supervising this research.

the absence of initial constraints, making the problem trivially solvable for most useful classes of TM's. In particular, any machine that incorporates an unbounded search has an obviously immortal ID, a vain search. In the construction showing the immortality problem undecidable, most of the complication arises from the necessity of avoiding unbounded searches (Part IV). Once this difficulty is overcome, the proof follows quite readily.

Most of the detailed work has been banished to the Appendices, to give the casual reader an opportunity to sample the flavor of the construction without choking on its bones. Of these, the Basic Lemma of Appendix VII is the spine, and the reader who thoroughly understands it, its proof, and its significance, should then have a complete grasp of the entire construction. This lemma is misleadingly simple in both statement and proof; for the induction proceeds in a straightforward manner that obscures the complexity of the underlying situation, the fact that, given enough tape, the computation will become nested to an arbitrary depth. (Specifically, a scanned section of tape of length $(2n+7)2^k$ must, eventually and recurrently contain k levels of nesting. A routine and unimaginative analysis-of-cases proof would point this out more clearly; but it has remained unwritten since, as a rather tedius insult to the alert, qualified reader, it would surely remain unread.)

It is perhaps worth noting that, at each stage of the construction, not one but an entire class of machines is being designed, with the additional indices being suppressed to keep the notation as unencumbered as possible. To prevent confusion on this point, frequent reminders to the reader have been scattered throughout the text.

Part II. The modified Minsky machines. We begin with the 2-tape, non-writing machines of Minsky [7], each considered as a programmed computer T^* with two registers, T_1^* and T_2^*, capable of storing arbitrarily large integers. We will use $T_i^*[\alpha, \beta]$ to represent the ID of T^*, about to execute instruction I_i with its registers containing α and β. The program for T^* is a numbered list of instructions chosen from the following repertoire, I^*:

$I_i = R_1[n]$ takes $T_i^*[\alpha, \beta]$ to $T_n^*[\alpha+1, \beta]$

$I_i = R_2[n]$ takes $T_i^*[\alpha, \beta]$ to $T_n^*[\alpha, \beta+1]$

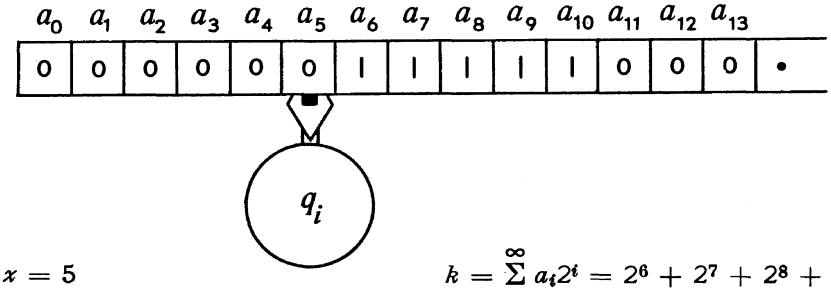
$I_i = L_1[n, m]$ takes $T_i^*[\alpha+1, \beta]$ to $T_n^*[\alpha, \beta]$ if $\alpha = 0$ and to $T_m^*[\alpha, \beta]$ otherwise

$I_i = L_2[n, m]$ takes $T_i^*[\alpha, \beta+1]$ to $T_n^*[\alpha, \beta]$ if $\beta = 0$ and to $T_m^*[\alpha, \beta]$ otherwise

Using this system one can simulate an arbitrary 2-symbol Turing machine, T_m, operating on semi-infinite tape, in the following manner. For each state, q_i, of T_m there is a subroutine $I^i = I_0^i, I_1^i, \ldots, I_{n(i)}^i$ for T^*, where

each I_j^i is chosen from I^*. For any (finite) tape of T_m, let a_i denote the symbol in the i^{th} square of the tape and define $k = \sum_0^\infty a_i 2^i$, as in Figure 1. An ID of T_m (including k, the tape contents, x, the index of the scanned square, and i, the state index) is then represented by the ID $T_i^*[2^k 3^{2^x}, 0]$ of T^*. (Here the "i" stands for instruction I_0^i).

Figure 1: Coding for the T^* Machines.



$x = 5$ $\qquad k = \sum_0^\infty a_i 2^i = 2^6 + 2^7 + 2^8 + 2^9 + 2^{10} = 1984$

Minsky shows that if an ID of T_m, with parameters (k, x, i) is followed by one with parameters (k', x', i'), then T^* started at $T_i^*[2^k 3^{2^x}, 0]$ goes eventually to $T_{i'}^*[2^{k'} 3^{2^{x'}}, 0]$. Most of the work of this simulation is done by a subroutine $C(R, S)$, coded over I^*. If R does not divide V and g.c.d. $(R, S) = 1$, then $C(R, S)$ takes $[R^n V, 0]$ into $[S^n V, 0]$; that is, every factor R of T_1^* is replaced by S. Appendix I contains examples of the use of this subroutine.

Perusing Minsky's paper one discovers the following. A transfer from subroutine I^i to subroutine I^j is made only if I^i has successfully completed its task, and I^j is entered *only* at I_0^j, with $T_2^* = 0$. Moreover, there is no infinite loop possible within a single subroutine I^i of T^* as long as both T_1^* and T_2^* are finite. Furthermore, the only primes used in the $C(R, S)$, Multiply, and Divide routines are 2, 3, 5, and 7, and any other prime factor will not effect the operation of T^* but will only ride along, appearing as a residual factor of T_1^* every time a transfer to some I_0^i is made. These observations permit the installation of some useful modifications.

For our first modification we notice that each T^* machine is designed to begin computing from its initial instruction, with T_2^* set at 0 and with some number, N, in T_1^*. It is under these conditions that the unsolvability of the halting problem arises. We can prefix each program with a sequence of $N R_1$ instructions, so that the halting problem now becomes unsolvable for a subclass of these T^* machines starting with two *empty* registers.

For the second modification we present the G (for growing) program, simply the instruction sequence $C(3, 5)$, $C(5, 33)$. This program transforms $[2^k 3^{2^x} M, 0]$ into $[2^k 3^{2^x} 11^{2^x} M, 0]$, provided that M is not divisible by 3 or 5.

Appendix I demonstrates that G can be introduced into T^* without altering the halting problem, while insuring the growth of T_1^* after the execution of every subroutine GI^i.

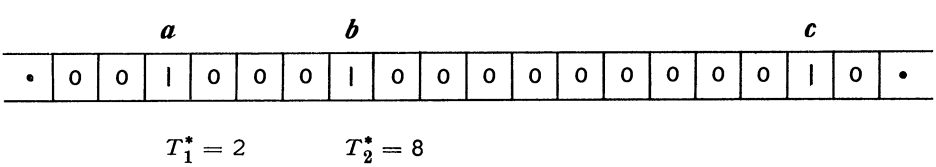
The third modification, the F (filter) program, is designed as a test for T_1^*, to determine whether or not it actually represents some ID of T_m. When the program F is applied to $[N, 0]$, it will halt either if N is divisible by 5 or 7, or if the exponent of 3 in N is not an integral power of 2. Otherwise we exit from F, leaving $[N, 0]$ unchanged. This program is presented in Appendix II, along with an explanation of its operation.

To incorporate these modifications into T^*, we will insert one copy of F and one of G before each subroutine I^i of T^*, redirecting each of the transfers that originally went to I_0^i to the first instruction of the F program preceding I^i. This produces a new machine, $\hat{T}$, with program $R_1^{(N)}F^0G^0I^0F^1G^1I^1 \ldots F^nG^nI^n$ and with a very useful property.

Let us consider any ID, $\hat{T}_j[\alpha, \beta]$, of $\hat{T}$. Unless it halts "soon", it must cause a transfer to the initial instruction of some F^i and must be of the form $\hat{T}_i[N, 0]$ at this time. The only way to exit from F^i without halting is for N to be of the form $2^k3^{2^x}M$, where none of 2, 3, 5, or 7 divide M. The G program then introduces a factor of 11^{2^x} into M and we find the machine in some ID, $\hat{T}_i[2^k3^{2^x}M', 0]$, where i is the index of some instruction I_0^i. Since M' can not interfere with the operation of $\hat{T}$, the above ID actually represents some ID of T_m, in state q_i scanning the x^{th} square of a tape containing the binary representation of k, and will continue to produce representations of successive ID's of T_m as long as T_m fails to halt. Hence, we can conclude this section with the remark that, providing both T_1^* and T_2^* are finite, any immortal ID of $\hat{T}$ must have successors in which T_1^* becomes arbitrarily large, due to repeated passes through G. Also worth mentioning is the fact that, for $\hat{T}$ to have an immortal ID, T_m must also possess one.

Part III. The 3-mark machines. Our next step is to simulate an arbitrary member of this class of $\hat{T}$ machines by a 2-symbol TM. This is easily done by considering a tape with only 3 marks on it and letting the two finite stretches delimited by these marks represent the contents of T_1^* and T_2^*, as in Figure 2. To keep the marks apart, we encode "x" by

Figure 2: Coding for the 3-Mark Machines.



$x+1$ spaces, so that "0" is represented by one space between marks. Although each mark will be a "|", we will call them ***a***, ***b***, and ***c*** for easier reference. Instead of fixing ***b*** in place and moving ***a*** and ***c*** to alter T_1^* and T_2^*, we fix ***a***, an approach that restricts action to a semi-infinite section of the tape, thus permitting information to be stored to the left of ***a***.

In designing the TM it is helpful to paraphrase each instruction of I^* as it actually applies to the simulation, and then design a set of TM quintuples for each paraphrased instruction. Appendix III presents sets of quintuples embodying the following paraphrased instructions. We assume that *each* instruction begins *and* ends with the machine scanning ***b***.

$R_2[n]$: Go right to ***c***. Move ***c*** one square right. Return left to ***b*** and transfer control to I_n. This merely increases T_2^* by 1.

$R_1[n]$: Move ***b*** one square right, then operate as in $R_2[n]$. This increases T_1^* by 1, by moving both ***b*** and ***c*** to the right.

$L_2[n, m]$: Go right to find ***c*** and move it one square left. Go 2 squares left and, if ***b*** is encountered there, transfer to I_n. Otherwise return left to ***b*** and transfer to I_m. This decreases T_2^* by 1, transferring to I_n if T_2^* becomes 0 and to I_m otherwise.

$L_1[n, m]$: Go right to ***c***, and move it one square left. Return left to ***b***, and move it one square left. Go 2 squares left. If ***a*** is encountered, return right two squares to ***b*** and enter I_n. If not, go left to find ***a*** (to assure T_1^* is finite), return to ***b***, and transfer to I_m. This leaves T_2^* inchanged, decreases T_1^* by 1, and branches on the final value of T_1^*.

Note that the mark ***a*** is never moved, or even crossed, during the execution of these instructions. Proceeding as above for each instruction of $\hat{T}$ produces a 2-symbol TM, T^R, which, when started in some state q_0^n scanning a tape configuration $\ldots 000|0^k|0^m|000\ldots$ will imitate the ID $\hat{T}_n[k-1, m-1]$ of $\hat{T}$. (In particular, T^R started in q_0 on $\ldots 0|0|0|0\ldots$ will simulate $\hat{T}_0[0, 0]$, so for this type of ID the class of T^R machines has an unsolvable halting problem.)

Let us distinguish three subsets of the states of T^R. S_L^R will be the set of "left-search" states, each with a quintuple of the form $q_i 00 L q_i$, and S_R^R will denote the similarly selected "right-search" states. (The presence of these unbounded searches makes the immortality problem for these T^R machines trivially solvable: each has an obviously immortal ID, a search state operating on blank tape.) The third subset, P_R, is the set of "right-print" states, those which print ***c*** one square to the right of its previous position, but halt on a "|". Only the states in these subsets will be modified in the remaining steps of the construction.

Each T^R machine has an interesting property. The only possible immortal ID's of a T^R machine must either get locked within one of the I^* subroutines (implying an unsuccessful, unbounded search) or must execute arbitrarily

many of them. The latter case implies that the machine is simulating *some* ID of $\hat{T}$; but, since *any* immortal ID of $\hat{T}$ grows without bound, this case also forces T^R to enter arbitrarily long searches.

Part IV. Bounding the searches. We shall extend the above construction by replacing each unbounded search by a bounded search which, if it fails, generates a new computation nested within the original one. Since both left and right searches can fail, we must be prepared to compute in either direction. (T^R computes, grows, to the right.) For this purpose we use the "mirror" machine, T^L, constructed from T^R by merely interchanging "R" and "L" in the defining quintuples of T^R and by adding n, the number of states of T^R, to the index of each state. This produces a machine which, if viewed in a mirror, is behaviorially identical with T^R, except for the naming of its states, and contains corresponding sets S_R^L and S_L^L of search states and a set P_L of "left-print" states. (In T^L, $\boldsymbol{c}$ progresses to the *left*.)

We also need a device to destroy a nested computation as soon as it has grown beyond the space available for it. To do this we use, in each direction, two simple "erase" machines and a "count" machine. To erase left, we first enter E_{L1}, a single search state which erases the first "|" to its left and transfers to E_{L2}. E_{L2}, another search state, transfers to C_L, without erasing, when it locates the first mark to its left. (Both search states will be modified later.) C_L moves left, counting while erasing a contiguous block of marks. If the length of this block corresponds to the index of a right-search state (the *only* case in which the machine is legitimately nested), we reenter that state, halting otherwise. The machines for erasing and counting to the right are simply the mirrors of these three, E_{R1}, E_{R2}, and C_R, and quintuples for all six machines appear in Appendix IV.

For the final step of the construction we define a (right) bounded search machine, $\bar{S}^x$, to replace (right)-search state q_x. This machine first searches right up to $x+4$ squares for a mark, and, if successful, returns to q_x. Otherwise, it prints the string $|^x0|0|$ in the $x+4$ blank squares and transfers to q_0, the initial state of T^R, ready to begin a (nested) computation simulating $\hat{T}_0[0, 0]$, a computation which, as noted before, either halts or grows (to the right) without bound. Since the rightmost of the string of x marks is treated as $\boldsymbol{a}$ in this nested computation, this block will not be disturbed during the simulation of $\hat{T}_0[0, 0]$ and, hence, preserves the index of the state from which we left the surrounding computation. Also, since q_0 is *not* a search state, x will be at least 1, and the three marks necessary for the nested computation will all be printed when a search fails.

A machine of this type is to be constructed for each right-search state, including those for E_{R1} and E_{R2}; and, similarly, a mirror of this prototype is to be constructed for each left-search state. A set of the quintuples and a state diagram for a typical $\bar{S}^x$ is offered in Appendix V. From these, it is

obvious that, once in an $\bar{S}^x$, the machine will either find a "|" and reenter q_x, begin simulating $\hat{T}_0[0, 0]$, or halt.

Machines T^R and T^L each use n states, the erase machines use 4, and the two count machines together use $4n+5$, for a total of $6n+9$ states. The maximum index of a search state is $2n+3$, and since $\bar{S}^x$ has $3x+9$ states, no $\bar{S}^x$ can have more than $6n+18$ states. Therefore we can assign indices to the states of each $\bar{S}^x$ machine by replacing q_i^x by q_{7nx+i}, providing a set of integers which includes a *distinct* index for each state of $\bar{T}$, the machine to be constructed from the submachines defined above.

As the final step we must interconnect these submachines to produce, for each T^R, a machine $\bar{T}$ with *all* of its searches bounded. This is done quite easily by replacing the search quintuple, $q_x 00 D q_x$, of each search state q_x, by a quintuple $q_x 00 D q_{7nx}$ which transfers to $\bar{S}^x$. The only additional changes are made from the sets P_R and P_L. Instead of halting on a "|" (which signals that a nested computation has grown to the boundary of the available space, the very "|" for which $\bar{T}$ was searching before failure caused the beginning of the nested computation) the states of these sets will now transfer to the erase machines (to unnest the computation) by adding the quintuple $q_x||Lq_{2n}$ for each state q_x of P_R, and $q_y||Rq_{2n+2}$ for each state q_y of P_L. This completes the construction of $\bar{T}$: a block diagram of $\bar{T}$ and its submachines appears as Appendix VI.

Part V. The immortality problem. The class of $\bar{T}$ machines just defined has an unsolvable immortality problem. To prove this, we will make use of a lemma, proved in Appendix VII, which states:

BASIC LEMMA. *If $\hat{T}_0[0, 0]$ is immortal, and $\bar{T}$ is in a right (left) search state q_x with a "|" somewhere to the right (left) of the scanned square, $\bar{T}$ will eventually be scanning that "|", in q_x, with no other change in the tape configuration. (That is, it will behave exactly as T^R, with unbounded search q_x, would have behaved in that situation.)*

We can quickly infer the following.

COROLLARY. *If $\hat{T}_0[0, 0]$ is immortal, $\bar{T}$ has an immortal* ID.

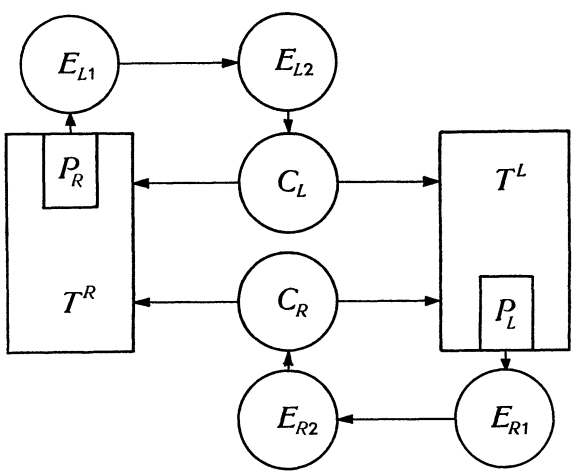
PROOF. $\bar{T}$, started in ID $\ldots 0|0|0|0\ldots$ in q_0 (or, for that matter, in q_n) will simulate $\hat{T}_0[0, 0]$, behaving like T^R (or T^L), and hence will have arbitrarily many successors. (Moreover, beginning on blank tape in *any* search state also provides an immortal ID, since it leads almost immediately to the situation above.)

The converse of the basic lemma (CBL), also in Appendix VII, states that the only possible influence that the mortality of $\hat{T}_0[0, 0]$ can have on an ID of $\bar{T}$ is to force a halt. We can now prove, as a corollary to this converse, the:

CONVERSE OF THE COROLLARY. *If $\bar{T}$ has an immortal* ID, *$\hat{T}_0[0, 0]$ is immortal.*

PROOF. Assume Δ is an immortal ID of $\bar{T}$. Since either the basic lemma or its converse must apply to every search (except, of course, on blank tape, where the result follows immediately from the fact that $\bar{T}$ simply begins simulating $\hat{T}_0[0, 0]$), the immortality of Δ implies the success of every search involved in the production of successors of Δ. (Otherwise $\bar{T}$ halts by CBL.) Therefore we can restrict our attention to an abbreviated block diagram, Figure 3, in which the bounded searches have been suppressed. Since Δ is immortal, $\bar{T}$ either remains eventually in T^R or T^L or goes through arbitrarily many erase-and-count loops, and each pass through such a loop erases, unnests, a level of computation, providing more blank type. So, in any case, $\bar{T}$ simulates larger and larger ID's of $\hat{T}$ (in one direction or the other, possibly alternating). It must, therefore, embark on increasingly long searches which provide the opportunity of simulating arbitrarily many steps of $\hat{T}_0[0, 0]$. Hence, since Δ does not lead to a halt, $\hat{T}_0[0, 0]$ must also be immortal.

Figure 3: Abbreviated Block Diagram for the $\bar{T}$ Machines.



Together, these yield the:

THEOREM. *The immortality problem for* 2*-symbol* TM*'s is recursively undecidable.*

PROOF. Each $\bar{T}$ machine is a 2-symbol TM, and each has an immortal ID if and only if its corresponding $\hat{T}_0[0, 0]$ is immortal. So the result is immediate from the undecidability of the halting problem for the class of $\hat{T}$ machines beginning with empty registers.

'Anyone who wishes, then, to argue on this
basis for the existence of the soul will
have to be prepared to hug the souls of
Turing Machines to his philosophic bosom!'

Hilary Putnam

Part VI. Supplementary results and remarks. Here we present some straightforward extensions of the above result on the *immortality problem* (IP) for TM's. Most are immediate, requiring only brief comment to convince the reader or lead him to a proof of his own. For the others, proofs-by-construction have been sketched, with details left to the reader.

1) Although the $\bar{T}$ machines operate on 2-way infinite tape, the result carries over immediately to the case of 1-way infinite tape.

2) The *initialized* IP, whether or not a TM has an immortal ID when started in a *specified initial state*, is also undecidable.

3) Since the undecidability proof is for 2-symbol TM's, it holds also for the classes of TM's with more than 2 symbols.

4) Considering some other formalisms for TM's, after Fischer [4], we note that, while the IP remains undecidable for the class of 2-symbol "Post machines" (quadruple), it becomes trivially solvable for the quadruple classes of "*D*-" and "*S*-machines", as well as for the triple formulation of "*U*-machines". (*Any* quadruple or triple which moves the scanner without changing state provides a trivial immortal ID.)

5) The IP is *solvable* for *2-state* TM's. Each instruction must produce a *change* of state, and neither state can produce shifts in *both* directions, or we get an immediate immortal ID of left- or right-shifting. Hence one needs only consider possible loops involving 2 tape squares.

6) Any system with *reversible* operations (Thue systems, Post systems, Davis [3]), has immortal ID's, hence a solvable IP.

7) The IP for T^* machines is recursively undecidable. This follows directly from the last remark of Part II.

8) The IP for Semi-Thue systems (Davis [3], p. 98) is recursively undecidable. This is not immediate, for the ordinary simulation does not exclude the possibility of words with more than one "state" symbol. However, if one extends the alphabet by replacing each "tape" symbol, S_i by two new symbols, S_i^R and S_i^L, redesigning the system to print S_i^R on every right-shift and S_i^L on every left-shift while prohibiting any right-shift *onto* an S_i^R (or left-shift onto an S_i^L) one ensures *independent* operation of the "state" symbols; and the result follows.

9) The IP for Normal Algorithms (Markov [6]) is immediately undecidable from the result for Semi-Thue systems.

10) The IP for Polygenic Post Normal Systems (PPNS) is recursively undecidable (Hooper [5]). This is shown by constructing, for any domino

set, $\{D_i\}$, a PPNS over the alphabet $\{d_i\} \cup \{*\}$ having productions of the following two types:

a) $d_x d_y P \to P d_z d_z$ if  is a compatible domino configuration.

b) $d_x * d_y P \to P d_u d_u * d_v d_v$ if  and  are both compatible.

It is not difficult to show that there exists an immortal word of this PPNS (without occurrences of "$*$") if and only if the set $\{D_i\}$ has a (periodic) solution. Hence, the domino problem (shown unsolvable by Berger [1]) reduces to the IP for PPNS, proving the latter undecidable.

11) Some of the above results can perhaps be rephrased in the metalanguage of formal grammars to produce statements about the productive ability of various types of generative grammars.

12) The following problems remain tantalizingly open:

a) The IP for non-erasing TM's. (Wang [9])

b) The IP for Lag Systems. (Wang [10])

c) The IP for *Monogenic* Post Normal Systems.

d) The IP for Tag Systems. (Wang [10])

e) The IP for S.S.Machines. (Shepherdson & Sturgis [8])

The undecidability of (c) would follow from the undecidability of (d) and would imply the undecidability of (e). While the author suspects these three to be undecidable, and (a) and (b) to be decidable, he has been unable to confirm or deny these suspicions.

Appendix I. The G program.

G: $C(3, 5)$

$C(5, 33)$

If $T \overset{*}{2} = 0$ and $T_1^* = 2^k 3^{2^x} M$ (where neither 5 nor 7 divides M), G transforms T_1^* into $2^k 5^{2^x} M$ and then into $2^k (33)^{2^x} M = 2^k 3^{2^x} 11^{2^x} M = 2^k 3^{2^x} M'$. The *maximum* decrease that a subroutine I^a can produce on T_1^* is to decrease the exponent of 3^2 by 1 (corresponding to a left-shift for T_m) and decrease the exponent of 2, k, by the exponent of 3, 2^x, corresponding to changing symbol a_x from "|" to "0". If G and then I^a are applied to T_1^* it is transformed first into $2^k 3^{2^x} 11^{2^x} M$ (by G), and then is changed by I^a into a number no smaller than $2^{k-2^x} 3^{2^{x-1}} 11^{2^x} M > 2^{k-2^x} 3^{2^{x-1}} 6^{2^x} M$ and we see that this equals $2^k 3^{2^x} 3^{2^{x-1}} M$, which is greater than the above value of T_1^*. So the sequence GI^i invariably increases T_1^*, altering only the residual factor M which, since it remains the product of primes greater than 7, can not effect the halting of T^*.

Appendix II. The *F* program.

F:	$Div(5)$	Halt if division is even.
	$Div(7)$	Halt if division is even.
	$Div(3)$	Halt if division is *not* even.
	$Div(3)$	If not even, $Mpy(3)$ and exit. (if exponent of 3 is 1)
	$Mpy(9)$	(Restores T_1^* to its original value)
	$Mpy(25)$	(To initialize exponent of 5)
a)	$C(3^2 5^2, 7)$	
	$Div(5)$	Halt if division is even.
	$Div(3)$	Go to (b) if division is *not* even.
	$Mpy(3)$	(replacing the factor just removed)
	$C(7, 5^4 3^2)$	Go to (a).
b)	$C(7, 3^2)$	Exit. (to the first instruction of some G^i)

Assume $T_1^* = 3^\alpha 5^\beta 7^\gamma M$, where none of 3, 5, or 7 divides M. Clearly F halts immediately if β or $\tilde{\gamma} \neq 0$, or if $\alpha = 0$. If $\alpha = 1$, we exit, since 9 does not divide T_1^*. If α exceeds 1, F compares α (the exponent of 3) with 2^x (the exponent of 5), as x increases through the positive integers. (Note that the loop through (a) is essentially the sequence of instructions $C(5^2, 7)$, $C(7, 5^4)$, which takes $5^{2^x} = (5^2)^{2^{x-1}}$ into $7^{2^{x-1}}$ and then into $(5^4)^{2^{x-1}} = (5^{2^2})^{2^{x-1}} = 5^{2^{x+1}}$.) If there is an x such that $\alpha = 2^x$, (a) will at some stage remove all factors of both 3 and 5 from T_1^*, so F will transfer to (b), restore T_1^*, and exit. Otherwise, 2^x will eventually exceed α and force a halt when the division by 5 following step (a) is successful. Hence, F halts unless $\alpha = 2^x$ for some x, in which case we exit from F with T_1^* unchanged.

Appendix III. Quintuples for the submachines of T^R.

If I_x is $R_2[n]$:	$[q_0^x \| \| R q_1^x$ $q_1^x 00 R q_1^x,\ q_1^x \| 0 R q_2^x$ $q_2^x 0 \| L q_3^x$ $q_3^x 00 L q_3^x,\ q_3^x \| \| N q_0^n]$	(Here q_1^x is S_R^R, q_2^x is in P_R, and q_3^x is in S_L^R.)
If I_x is $R_1[n]$:	$[q_0^x \| 0 R q_4^x$ $q_4^x 0 \| R q_1^x]$	(With q_1^x, q_2^x, and q_3^x as in $R_2[n]$)
If I_x is $L_2[n, m]$:	$[q_0^x \| \| R q_1^x$ $q_1^x 00 R q_1^x,\ q_1^x \| 0 L q_2^x$ $q_2^x 0 \| L q_3^x$ $q_3^x 00 L q_4^x$ $q_4^x 00 L q_5^x,\ q_4^x \| \| N q_0^n$ $q_5^x 00 L q_5^x,\ q_5^x \| \| N q_0^m]$	(The $q_0^x 0$ halt implies that **b** is *not* under scan when the routine is entered.) (The $q_2^x \|$ halt is for the case in which we are trying to relocate **c** in a square that is *already* a marked square.)

If I_x is $L_1[n, m]$: $[q_0^x||Rq_1^x$
$q_1^x00Rq_1^x$, $q_1^x|0Lq_2^x$ (The other halting configurations
$q_2^x0|Lq_3^x$ are for cases in which two marks
$q_3^x00Lq_3^x$, $q_3^x|0Lq_4^x$ are, improperly, adjacent.)
$q_4^x0|Lq_5^x$ (Note that the *only* loops possible
$q_5^x00Lq_6^x$ within one of these routines are
$q_6^x00Lq_9^x$, $q_6^x||Rq_7^x$ those remaining in a search state.)
$q_7^x00Rq_8^x$
$q_8^x||Nq_0^n$
$q_9^x00Lq_9^x$, $q_9^x||Rq_{10}^x$
$q_{10}^x00Rq_{10}^x$, $q_{10}^x||Nq_0^m]$

Appendix IV. Quintuples for E_{L1}, E_{L2}, E_{R1}, E_{R2}, C_L, and C_R.

E_{L1}: $[q_{2n}00Lq_{2n}, q_{2n}|0Lq_{2n+1}]$ (Since these four states are search
E_{L2}: $[q_{2n+1}00Lq_{2n+1}, q_{2n+1}||Nq_{2n+4}]$ states, they will be modified to pro-
E_{R1}: $[q_{2n+2}00Rq_{2n+2}, q_{2n+2}|0Rq_{2n+3}]$ vide transfers to bounded search
E_{R2}: $[q_{2n+3}00Rq_{2n+3}, q_{2n+3}||Nq_{4n+7}]$ routines.)

C_L: $q_{2n+4+i}|0Lq_{2n+4+(i+1)}$ $i = 0, 1, \ldots, 2n+2$
$q_{2n+4+i}00Rq_i$ $i = 0, 1, \ldots, 2n+3 \;\&\; i \in S_R^R \cup S_R^L \cup E_{R1} \cup E_{R2}$
(Halt in all other cases, since the length of the block of marks does not correspond to the index of a right search state.)

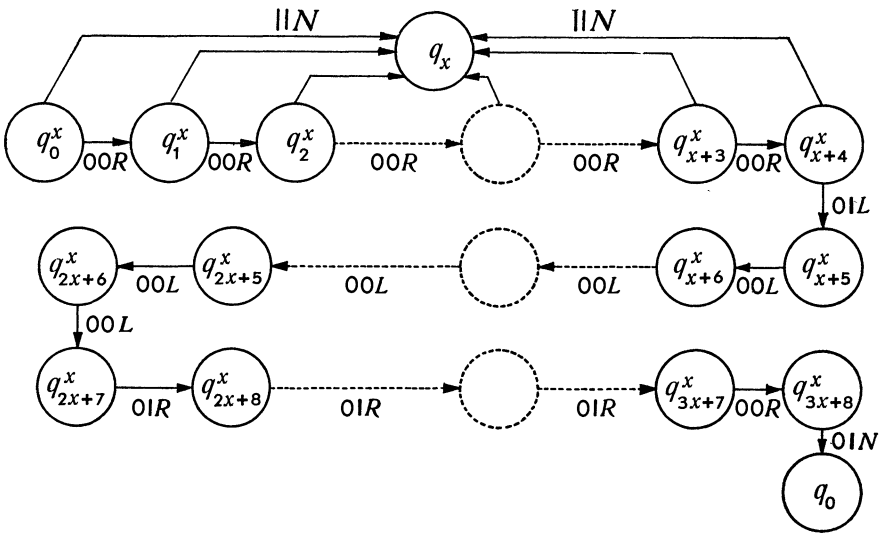
C_R: $q_{4n+7+i}|0Rq_{4n+7+(i+1)}$ $i = 0, 1, \ldots, 2n$
$q_{4n+7+i}00Lq_i$ $i = 0, 1, \ldots, 2n+1 \;\&\; i \in S_L^R \cup S_L^L \cup E_{L1} \cup E_{L2}$
(Halt otherwise, as above.)

The indexing of these states has been chosen to keep the various sub-machines of $\overline{T}$ disjoint, the first n integers being used for T^R, and the next set of n for T^L.

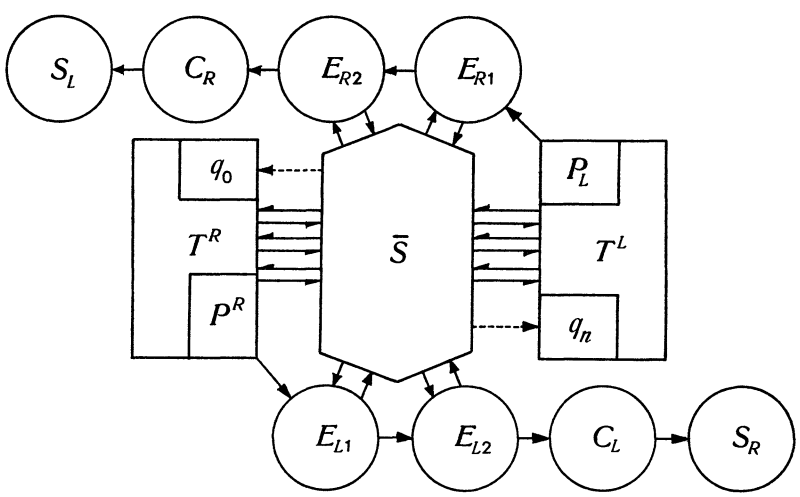
Appendix V. Quintuples and state diaqram for a typical (right) bounded search machine.

$\bar{S}^x$: $[q_i^x00Rq_{i+1}^x$ $i = 0, 1, \ldots, x+3$ (Search right up to $x+4$ squares)
$q_i^x||Nq_x$ $i = 0, 1, \ldots, x+4$ (reenter q_x if a "|" is located.)
$q_{x+4}^x0|Lq_{x+5}^x$ (print a "|" [c] at the end of an unsuccessful search.)
$q_i^x00Lq_{i+1}^x$ $i = x+5, x+6, \ldots, 2x+6$ (return left $x+2$ more squares.)
$q_i^x0|Rq_{i+1}^x$ $i = 2x+7, 2x+8, \ldots, 3x+6$ (print a block of "x" marks.)
$q_{3x+7}^x00Rq_{3x+8}^x$ (skip right over one blank)
$q_{3x+8}^x0|Nq_0**$ (print another "|" and enter q_0)

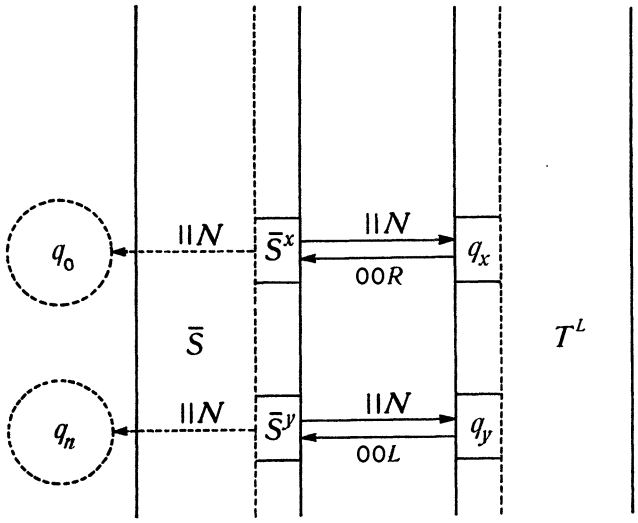
** If q_x had been a left-search state, $\bar{S}^x$ would be the mirror of the above, and this transfer would be to q_n instead of to q_0. $\bar{T}$ then would be in the initial state of T^L ready to begin simulating $\hat{T}_0[0, 0]$ to the *left*.



Appendix VI. Block diagram of $\bar{T}$.



All bounded-search machines are included within $\bar{S}$. S_R is the set of all right-search states; those from T^R, T^L, E_{R1}, and E_{R2}. Similarly for S_L. The broken arrows represent transfers after vain searches, into nested computation.



Detail of $\bar{S}$ — T^L interface, showing right-search state q_x and left-search state q_y, and their associated bounded-search machines.

Appendix VII. The basic lemma and its converse.

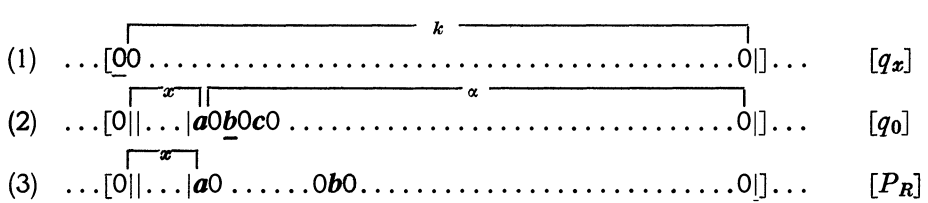
LEMMA. *If $\hat{T}_0[0, 0]$ is immortal (hence, grows) and $\bar{T}$ is in a right-(left-) search state q_x with a "|" somewhere to the right (left) of the scanned square, $\bar{T}$ will eventually be scanning that "|", in q_x, with no other change in the tape configuration.*

PROOF. We let d be the distance (number of blank squares) between the scanned square and the sought-for "|", and induct simultaneously on all search states, left and right, of $\bar{T}$. Since the argument is symmetric, no generality is lost by considering q_x to be a right-search state.

Basis: $d = 1$. Obviously the lemma holds by the design of the $\bar{S}^x$.

Inductive Hypothesis: (IH) We assume the lemma holds for all states when d is less than k.

Inductive Step: Choose any right-search state q_x and assume $d = k$. (If $k < x+4$, the proof is also immediate from $\bar{S}^x$, since the search is immediately successful. Therefore, we assume that $k \geqq x+4$.) The following ID's will be produced: (***a***, ***b***, and ***c*** are simply marks, as before.)



(4) $\ldots[0\overset{\overset{x}{\ulcorner\!\urcorner}}{||}\ldots|\boldsymbol{a}0\ldots\ldots0\underline{\boldsymbol{b}}0\ldots\ldots\ldots\ldots\ldots\ldots\ldots0|]\ldots$ $[E_{L1}]$

(5) $\ldots[0\overset{\overset{x}{\ulcorner\!\urcorner}}{||}\ldots|\underline{\boldsymbol{a}}0\ldots\ldots000\ldots\ldots\ldots\ldots\ldots\ldots0|]\ldots$ $[C_L]$

(6) $\ldots[0\overset{\overset{k-1}{\ulcorner\!\!\!\!\!\!\!\!\!\!\!\!\urcorner}}{\underline{0}0}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots0|]\ldots$ $[q_x]$

(1) leads to (2) simply because the search fails. $\overline{T}$ then simulates $\hat{T}_0[0, 0]$ in the region α. All searches involved in this simulation must succeed, by IH, since they are of length less than k, so the computation grows (to the right) until eventually it exceeds the available computing space, at (3), where a "|" is located in some state of P_R, as $\boldsymbol{c}$ is being moved to the right. This causes a transfer to E_{L1}, searching for $\boldsymbol{b}$; and, since this search is of length less than k, it succeeds, producing (4). The mark, $\boldsymbol{b}$, is then erased, and the search of E_{L2} locates $\boldsymbol{a}$, and enters C_L, as in (5). Since "x" *is* the index of a right-search state, $\overline{T}$ then reenters q_x at (6); but now, with a search of length $k-1$, IH applies and we can conclude that the lemma holds for all d and all search states.

Q.E.D.

CONVERSE OF THE BASIC LEMMA. *If $\hat{T}_0[0, 0]$ is mortal, and $\overline{T}$ is in some right- (left-) search state q_x with a "|" somewhere to the right (left) of the scanned square, $\overline{T}$ will either halt, or it will eventually scan that "|", in q_x, with no other change in the tape configuration.*

PROOF. The immortality of $\hat{T}_0[0, 0]$ was used in the lemma *only* to insure the growth of the nested computation, so that $\overline{T}$ would eventually be in ID (3) above and would begin to "unnest" the computation. The mortality of $\hat{T}_0[0, 0]$ can effect the situation only if region α is large enough to permit $\overline{T}$ to simulate $\hat{T}_0[0, 0]$ through its terminal ID, forcing a *halt* of $\overline{T}$. Otherwise, the lemma remains in force.

Q.E.D.

REFERENCES

[1] ROBERT BERGER, *The undecidability of the domino problem*, Doctoral Dissertation, Harvard University, Cambridge Massachusetts, July, 1964.

[2] J. R. BÜCHI, *Turing-machines and the entscheidungsproblem*, ***Mathematische Annalen***, vol. 148 (1962), pp. 201–213.

[3] MARTIN DAVIS, ***Computability and Unsolvability***, MacGraw-Hill, 1958.

[4] PATRICK C. FISCHER, *On formalisms for Turing machines*, Report No. BL–35, by the Staff of the Harvard Computation Laboratory to Bell Telephone Laboratories, July, 1964.

[5] PHILIP K. HOOPER, *Post normal systems: the unrestricted halting problem*, (abstract) ***Notices of the American Mathematical Society***, vol. 12, no. 3 (1965) p. 371.

[6] A. A. MARKOV, ***Theory of Algorithms,*** OTS 60–51805 (1961).

[7] MARVIN L. MINSKY, *Recursive unsolvability of Post's problem of tag,* ***Annals of Mathematics,*** vol. 74 (1961) pp. 437–455.

[8] J. C. SHEPHERDSON and H. E. STURGIS, *The computability of partial recursive functions by forms of Turing machines,* ***Journal of the Association of Computing Machinery,*** vol. 10 (1963) pp. 217–255.

[9] HAO WANG, *Tag systems and lag systems,* ***Mathematische Annalen,*** vol. 152 (1963) pp. 65–74.

[10] HAO WANG, *A variant to Turing's theory of computing machines,* ***Journal of the Association of Computing Machinery,*** vol. 4 (1957) pp. 63–92.

HARVARD UNIVERSITY