A UNIVERSAL TURING MACHINE WITH TWO INTERNAL STATES

Claude E. Shannon

INTRODUCTION

In a well-known paper[1], A. M. Turing defined a class of computing machines now known as Turing machines. We may think of a Turing machine as composed of three parts — a control element, a reading and writing head, and an infinite tape. The tape is divided into a sequence of squares, each of which can carry any symbol from a finite alphabet. The reading head will at a given time scan one square of the tape. It can read the symbol written there and, under directions from the control element, can write a new symbol and also move one square to the right or left. The control element is a device with a finite number of internal "states." At a given time, the next operation of the machine is determined by the current state of the control element and the symbol that is being read by the reading head. This operation will consist of three parts; first the printing of a new symbol in the present square (which may, of course, be the same as the symbol just read); second, the passage of the control element to a new state (which may also be the same as the previous state); and third, movement of the reading head one square to the right or left.

In operation, some finite portion of the tape is prepared with a starting sequence of symbols, the remainder of the tape being left blank (i.e., registering a particular "blank" symbol). The reading head is placed at a particular starting square and the machine proceeds to compute in accordance with its rules of operation. In Turing's original formulation

---

[1] Turing, A. M., "On Computable Numbers, with an Application to the Entscheidungsproblem," Proc. of the London Math. Soc. 2 - 42 (1936), pp. 230 - 265.

alternate squares were reserved for the final answer, the others being used for intermediate calculations. This and other details of the original definition have been varied in later formulations of the theory.

Turing showed that it is possible to design a universal machine which will be able to act like any particular Turing machine when supplied with a description of that machine. The description is placed on the tape of the universal machine in accordance with a certain code, as is also the starting sequence of the particular machine. The universal machine then imitates the operation of the particular machine.

Our main result is to show that a universal Turing machine can be constructed using one tape and having only two internal states. It will also be shown that it is impossible to do this with one internal state. Finally a construction is given for a universal Turing machine with only two tape symbols.

THE TWO-STATE UNIVERSAL TURING MACHINE

The method of construction is roughly as follows. Given an arbitrary Turing machine A with an alphabet of m letters (symbols used on the tape, including the blank) and n internal states, we design a machine B with two internal states and an alphabet of at most $4mn + m$ symbols. Machine B will act essentially like machine A. At all points of the tape, except in the position opposite the reading head and one adjacent position, the tape of B will read the same as the tape of A at corresponding times in the calculation of the two machines. If A is chosen to be a universal Turing machine, then B will be a universal Turing machine.

Machine B models the behavior of machine A, but carries the information of the internal state of A via the symbols printed on the tape under the reading head and in the cell of the tape that the reading head of A will next visit. The main problem is that of keeping this state information up to date and under the reading head. When the reading head moves, the state information must be transferred to the next cell of the tape to be visited using only two internal states in machine B. If the next state in machine A is to be (say) state 17 (according to some arbitrary numbering system) this is transferred in machine B by "bouncing" the reading head back and forth between the old cell and the new one 17 times (actually 18 trips to the new cell and 17 back to the old one). During this process the symbol printed in the new cell works through a kind of counting sequence ending on a symbol corresponding to state 17, but also retaining information as to the symbol that was printed previously in this cell. The bouncing process also returns the old cell back to one of the elementary symbols (which correspond one-to-one with the symbols used by machine A), and in fact returns it to the particular elementary symbol that should be printed in that cell when the operation is complete.

The formal construction of machine B is as follows: Let the symbol alphabet of machine A be $A_1, A_2, \ldots, A_m$, and let the states be $S_1, S_2, \ldots, S_n$. In machine B we have m elementary symbols corresponding to the alphabet of the A machine, $B_1, B_2, \ldots, B_m$. We further define 4mn new symbols corresponding to state symbol pairs of machine A together with two new two-valued indices. These symbols we denote by $B_{i,j,x,y}$ where $i = 1, 2, \ldots, m$ (corresponding to the symbols), $j = 1, 2, \ldots, n$ (corresponding to the states), $x = +$ or $-$ (relating to whether the cell of the tape is transmitting or receiving information in the bouncing operation) and $y = R$ or L (relating to whether the cell bounces the control to the right or left).

The two states of machine B will be called α and β. These two states are used for two purposes: First, on the initial step of the bouncing operation they carry information to the next cell being visited as to whether the old cell is to the right (α) or left (β) of the new one. This is necessary for the new cell to bounce the control back in the proper direction. After the initial step this information is retained in the new cell by the symbol printed there (the last index y). Second, the states α and β are used to signal from the old cell to the new one as to when the bouncing operation is complete. Except for the initial step of bouncing, state β will be carried to the new cell until the end of the bouncing operation when an α is carried over. This signifies the end of this operation and the new cell then starts acting as a transmitter and controlling the next step of the calculation.

Machine B is described by telling what it does when it reads an arbitrary symbol and is in an arbitrary state. What it does consists of three parts: printing a new symbol, changing to a new state, and moving the reading head to right or left. This operation table for machine B is as follows.

symbol;	state ⟶	symbol;	state;	direction		
B_i;	α ⟶	$B_{i,1,-,R}$;	α;	R	$(i = 1, 2, \ldots, m)$	(1)
B_i;	β ⟶	$B_{i,1,-,L}$;	α;	L	$(i = 1, 2, \ldots, m)$	(2)
$B_{i,j,-,x}$;	β ⟶	$B_{i,(j+1),-,x}$;	α;	x	$(i = 1, 2, \ldots, m)$ $(j = 1, 2, \ldots, n-1)$ $(x = R, L)$	(3)
$B_{i,j,+,x}$;	α or β ⟶	$B_{i,(j-1),+,x}$;	β;	x	$(i = 1, 2, \ldots, m)$ $(j = 2, \ldots, n)$ $(x = R, L)$	(4)
$B_{i,1,+,x}$;	α or β ⟶	B_i;	α;	x	$(i = 1, 2, \ldots, m)$ $(x = R, L)$	(5)

So far, these operations do not depend (except for the number of symbols involved) on the operation table for machine A. The next and last type of operation is formulated in terms of the operation table of the machine being modeled. Suppose that machine A has the operation formula

$$(6) \qquad A_i;\ S_j \longrightarrow A_k;\ S_\ell;\ \begin{matrix} R \\ L \end{matrix}.$$

Then machine B is defined to have

$$(7) \qquad B_{i,j,-,x};\ \alpha \longrightarrow B_{k,\ell,+,\begin{matrix} R \\ L \end{matrix}};\ \begin{matrix} \beta \\ \alpha \end{matrix};\ \begin{matrix} R \\ L \end{matrix}$$

where if the upper letter (R) occurs in (6) the upper letters are used in (7) and conversely.

To see how this system works, let us go through a cycle consisting of one operation of machine A and the corresponding series of operations of machine B.

Suppose that machine A is reading symbol A_3 and is in state S_5 and suppose its operation table requires that it print A_8, go into state S_4 and move to the right. Machine B will be reading (by inductive assumption) symbol $B_{3,7,-,x}$ (whether x is R or L depends on preceding operations and is irrelevant to those which follow). Machine B will be in state α. By relation (7), machine B will print $B_{8,4,+,R}$, go into state β, and move to the right. Suppose the cell on the right contains A_{13} in machine A; in machine B the corresponding cell will contain B_{13}. On entering this cell in state β, by relation (2) it prints $B_{13,1,-,L}$, goes into state α, and moves back to the left. This is the beginning of the transfer of state information by the bouncing process. On entering the left cell, it reads $B_{8,4,+,R}$ and by relation (4) prints $B_{8,3,+,R}$, goes to state β and moves back to the right. There, by relation (3), it prints $B_{13,2,-,L}$, goes into state α and returns to the left. Continuing in this manner, the process is summarized in Table I.

The operations indicated complete the transfer of state information to the right cell and execution of the order started in the left cell. The left cell has symbol B_8 registered (corresponding to A_8 in machine A) and the right cell has symbol $B_{13,4,-,L}$ registered, with the reading head coming into that cell with internal state α. This brings us back to a situation similar to that assumed at the start, and arguing by induction we see that machine B models the behavior of machine A.

To get machine B started in a manner corresponding to machine A, its initial tape is set up corresponding to the initial tape of A (with A_i replaced by B_i) except for the cell initially occupied by the reading head. If the initial state of machine A is S_j and the initial symbol in this cell is A_i, the corresponding cell of the B tape has $B_{i,j,-,R(\text{or } L)}$ registered and its internal state is set at α.

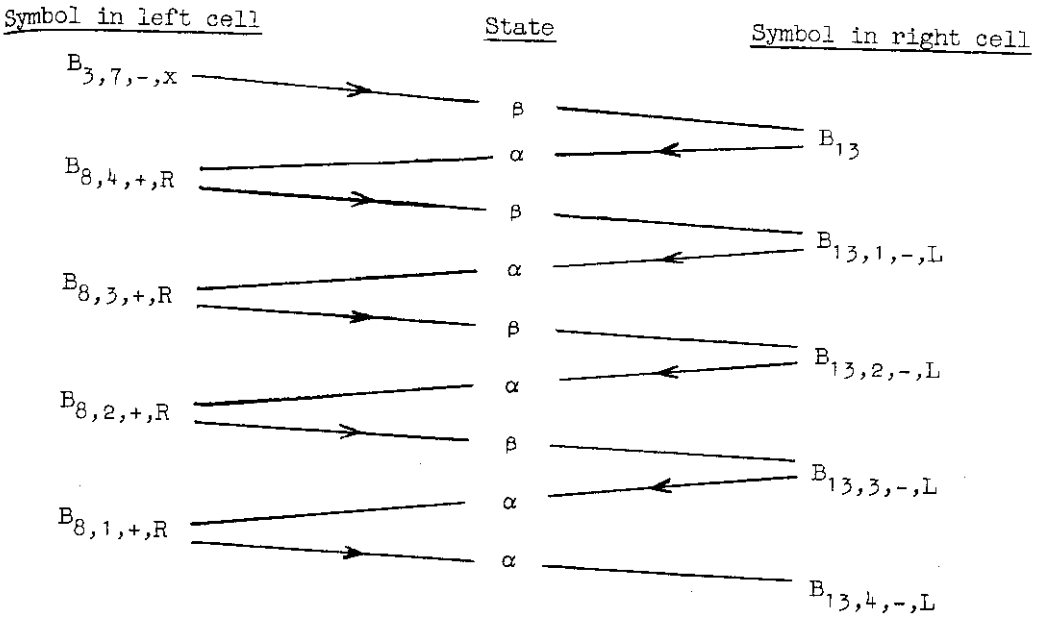

Table I

IMPOSSIBILITY OF A ONE-STATE UNIVERSAL TURING MACHINE

It will now be shown that it is impossible to construct a universal Turing machine using one tape and only one internal state.

Suppose we have a machine satisfying these conditions. By registering a suitable "description number" of finite length on part of the tape (leaving the rest of the tape blank), and starting the reading head at a suitable point, the machine should compute any computable number, in particular the computable irrational numbers, e.g., $\sqrt{2}$. We will show that this is impossible.

According to Turing's original conception, $\sqrt{2}$ would be computed in a machine by the machine printing the successive digits of $\sqrt{2}$ (say, in binary notation) on a specified sequence of cells of the tape (say, on alternate cells, leaving the others for intermediate calculations). The following proof assumes $\sqrt{2}$ to be calculated in such a form as this, although it will be evident that modifications would take care of other reasonable interpretations of "calculating $\sqrt{2}$."

Since $\sqrt{2}$ is irrational, its binary digits do not, after any finite point, become periodic. Hence if we can show that with a one-state machine either (1) all but a finite number of the cells eventually have the same symbol registered, or (2) all but a finite number of the cells change indefinitely, we will have proved the desired result.

Assume first a doubly infinite tape — an infinite number of blanks each side of the description number for $\sqrt{2}$. When the reading head enters a blank cell it must either stay there indefinitely or eventually move out either to right or left. Since there is only one state, this behavior does not depend on previous history of the computation. In the first case, the reading head will never get more than one removed from the description number and all the tape except for a finite segment will be constant at the blank symbol. If it moves out of a blank symbol to the left, either the left singly infinite section of blank tape is not entered in the calculation and therefore need not be considered, or if it is entered, the reading head from that time onward continues moving to the left leaving all these previously blank cells registering the same symbol. Thus the tape becomes constant to the left of a finite segment and blank to the right of this segment and could not carry $\sqrt{2}$. A similar situation arises if it emerges to the right from an originally blank cell. Hence the doubly infinite tape is no better than the singly infinite tape and we may assume from symmetry a singly infinite tape to the right of the description number.

Now consider the following operation. Place the reading head on the first cell of this infinite blank strip. The machine will then compute for a time and perhaps the reading head will be transferred back out of this strip toward the description number. If so, replace it on the first cell of the now somewhat processed blank tape. If it returns again off the tape, again replace it on the first cell, etc. The number of times it can be placed on the first cell in this fashion will be called the reflection number of the machine and denoted by R. This will be either an integer 1, 2, 3, ..., or ∞.

Now consider placing the reading head at its appropriate start for the description number to compute $\sqrt{2}$. After a certain amount of computation the reading head will perhaps emerge from the description number part of the tape. Replace it on the last cell of the description number. Again after a time it will possibly emerge. Continue this process as long as possible. The number of times it emerges will either be an integer 0, 1, 2, 3, ..., or ∞. This number, S, we call the reflection number for the $\sqrt{2}$ description.

If S is finite and R (possibly ∞) $>$ S, the reading head after a finite time will be trapped in the part of the tape that originally contained the description number. Only a finite amount of the blank tape will have been changed and the machine will not have calculated $\sqrt{2}$.

If both R and S are infinite, the reading head will return indefinitely to the description number part of the tape. The excursions into the originally blank parts will either be bounded or not. If they are bounded only a finite amount of the blank tape will have been changed as in the preceding case. If the excursions are unbounded, all but a finite segment of tape will be operated on by the reading head an unlimited number of times.

since there is only one state and a finite alphabet of tape symbols, the symbol registered in a cell visited an unlimited number of times must either come to a constant (the same for all these cells) or else change cyclically an infinite number of times. In the first case, all the originally blank tape becomes constant and cannot represent $\sqrt{2}$. In the second case all the blank tape is continually changing and cannot be the computation of anything.

If $R \leq S$, the reading head eventually moves into the original blank part of the tape and stays there. In this case it can be shown that the symbols in the originally blank part become constant. For either it moves to the right out of the first blank cell into the second blank cell at least R times, or not. If not the reading head is trapped in what was the first blank cell after a finite time, and all but a finite amount of tape remains constant at the blank symbol. If it does move out R times it will not return to the first originally blank cell since R is the reflection number for blank tape. This first cell will then have registered the result of operating on a blank 2R times (R coming in from the left and R from the right). The second originally blank cell will eventually register the same constant symbol, since the same argument applies to it as to the first. In each case the machine works into the same tape (an infinite series of blanks) and enters the same number of times (R). This exhausts the cases and completes the proof.

MODELING A TURING MACHINE WITH ONLY TWO TAPE SYMBOLS

It is also possible, as we will now show, to construct a machine, C, which will act like any given Turing machine A and use only two symbols 1 and 0 on its tape, one of which, 0 say, is the symbol for a blank square. Suppose, as before, a given machine A has m tape symbols and n internal states. Let ℓ be the smallest integer such that m is less than or equal to 2^{ℓ}. Then we may set up an arbitrary association of the m symbols used by machine A with binary sequences of length ℓ, letting however the blank symbol of machine A correspond to the sequence of ℓ zeroes. Basically, the machine C will operate with binary sequences; an elementary operation in machine A will correspond in machine C to stepping the reading head to the right $\ell - 1$ squares (storing the read information in its internal state) then stepping back to the left $\ell - 1$ squares, writing the proper new symbol as it goes, and finally moving either to the right or to the left ℓ squares to correspond to the motion of the reading head of machine A. During this process, the state of machine A is also, of course, carried in machine C. The change from the old state to the new state occurs at the end of the reading operation.

The formal construction of machine C is as follows. Corresponding to states $S_1, S_2, \ldots, S_n$ of machine A we define states $T_1, T_2, \ldots, T_n$ in

machine C (these will occur when machine C is at the beginning of an operation, reading the first symbol in a binary sequence of length ℓ). For each of these T_i we define two states T_{i0} and T_{i1}. If machine C is in state T_i and reads symbol 0, it moves to the right and goes into state T_{i0}. If it reads a 1, it moves to the right and goes into state T_{i1}. Thus, after reading the first symbol of a binary sequence, these two states remember what that symbol was. For each of these there are again two states T_{i00}, T_{i01} and T_{i10} and T_{i11}. If the machine is in the state T_{i0} for example and reads the symbol 0 it goes to the state T_{i00} and similarly for the other cases. Thus these states remember the initial state and the first two symbols read in the reading process. This process of constructing states is continued for $\ell - 1$ stages, giving a total of $(2^\ell - 1)n$ states. These states may be symbolized by

$$T_{i,x_1,x_2,\ldots,x_s} \quad i = 1, 2, \ldots, n; \quad x_j = 0, 1; \quad s = 0, 1, \ldots, \ell - 1.$$

If the machine is in one of these states ($s < \ell - 1$) and reads 0 or 1, the machine moves to the right and the 0 or 1 appears as a further index on the state. When $s = \ell - 1$, however, it is reading the last binary symbol in the group of ℓ. The rules of operation now depend on the specific rules of machine A. Two new sets of states somewhat similar to the T states above are defined, which correspond to writing rather than reading:

$$R_{i,x_1,x_2,\ldots,x_s} \quad \text{and} \quad L_{i,x_1,x_2,\ldots,x_s}.$$

A sequence $x_1, x_2, \ldots, x_{\ell-1}, x_\ell$ corresponds to a symbol of machine A. Suppose that when machine A is reading this corresponding symbol and is in state i it prints the symbol corresponding to the binary sequence $y_1, y_2, \ldots, y_{\ell-1}, y_\ell$, goes to state j and moves (say) right. Then we define machine C such that when in state $T_{i,x_1,x_2,\ldots,x_{\ell-1}}$, and reading symbol x_ℓ, it goes into state $R_{j,y_1,y_2,\ldots,y_{\ell-1}}$, prints y_ℓ and moves to the left. In any of the states $R_{i,y_1,y_2,\ldots,y_s}$ (or $L_{i,y_1,y_2,\ldots,y_s}$), machine C writes y_s, moving to the left and changes to state $R_{i,y_1,y_2,\ldots,y_{s-1}}$ (or $L_{i,y_1,y_2,\ldots,y_{s-1}}$). By this process the binary sequence corresponding to the new symbol is written in place of the old binary sequence. For the case $s = 1$, the writing of y_1 completes the writing operation of the binary sequence. The remaining steps are concerned with moving the reading head ℓ steps to the right or left according as the machine is in an R state or an L state. This is carried out by means of a set of U_{is} and V_{is} ($i = 1, 2, \ldots, n$; $s = 1, 2, \ldots, \ell - 1$). In state R_{ix_1} the machine writes x_1, moves to the right, and goes into state U_{i1}. In each of the U states it continues to the right, printing nothing and going into the next higher indexed U state until the last one is reached.

Thus U_{is} produces motion to the right and state U_{is+1} $(s < \ell - 1)$. Finally $U_{i\ell-1}$ leads, after motion to the right, to T_i, completing the cycle. In a similar fashion, L_{ix_i} leads to motion to the left and state V_{i1}; V_{is} gives motion to the left and V_{is+1} $(s < \ell - 1)$; finally, $V_{i\ell-1}$ gives motion to the left and T_i.

The initial tape for machine C is, of course, that for machine A with each symbol replaced by its corresponding binary sequence. If machine A is started on a particular symbol, machine C will be started on the left-most binary symbol of the corresponding group; if machine A is started in state S_i, C will be started in state T_i.

Machine C has at most $n(1 + 2 + 4 \ldots + 2^{\ell-1}) = n(2^\ell - 1)$ T states, similarly at most $n(2^\ell - 2)$ R states and $n(2^\ell - 2)$ L states, and finally $2n(\ell - 1)$ U and V states. Thus altogether not more than $3n2^\ell + n(2\ell - 7)$ states are required. Since $2^\ell < 2m$, this upper bound on the number of states is less than $6mn + n(2\ell - 7)$, which in turn is certainly less than $8mn$.

The results we have obtained, together with other intuitive considerations, suggest that it is possible to exchange symbols for states and vice versa (within certain limits) without much change in the product. In going to two states, the product in the model given was increased by a factor of about 8. In going to two symbols, the product was increased by a factor of about 6, not more than 8. These "loss" factors of 6 and 8 are probably in part due to our method of microscopic modeling — i.e., each elementary operation of machine A is modeled into machine B. If machine B were designed merely to have the same calculating ability as A in the large, its state-symbol product might be much more nearly the same. At any rate the number of logical elements such as relays required for physical realization will be a small constant (about 2 for relays) times the base two logarithm of the state-symbol product, and the factor of 6 or 8 therefore implies only a few more relays in such a realization.

An interesting unsolved problem is to find the minimum possible state-symbol product for a universal Turing machine.