

A Universal Turing Machine

1 Conventions and Preliminaries

A **Turing machine** consists of an infinitely long tape divided into individual cells, a movable “head” to read and write characters on the cells, and a program that dictates how the head should react to the machine’s potential tape configurations. By convention, the leftmost cell on the tape always contains the special symbol \triangleright , which marks the edge of the tape. The remaining tape extends infinitely to the right, with each successive cell containing exactly one symbol from the finite **alphabet** $A = \{\triangleright, a_1, a_2, \dots, a_n, \sqcup\}$, where \sqcup is called a **blank** square. The Turing machine program invokes a finite collection of **Turing states** $Q = \{q_0, q_1, \dots, q_k, q_h\}$, where q_h denotes the distinguished **halting state**. In each step of a computation, the Turing machine reads the symbol underneath its head, writes a symbol in the current cell, changes to a new Turing state, and finally moves the head one cell to the left, right, or remains neutral. The details of this computation process are determined by the program.

The **input** for the Turing machine is represented by the prepared initial pattern on the tape, and the **output** of the machine is given by the existing symbols on the tape when the machine reaches the halting state q_h . We will assume that the input for any Turing machine is always finite in the sense that to the right of some cell in the initial pattern, there will be an infinite string of blank cells. The output of the Turing machine may be interpreted in any way we like; we will not use the convention that the output consists only of those symbols residing to the left of the first blank. Ultimately, we will find this flexibility convenient when we both input and finally read off the output from our universal Turing machine.

In addition to terminating upon reaching the halting state q_h , a Turing machine will also halt if either the procedure for the current state and symbol pair (q_i, a_j) is not defined in the program, or if the head falls off the left end of the tape. We designate either of the last two cases to be an error, and we may consider the resulting output to be rubbish. It is also possible that the Turing program may not halt at all.

By convention, we will assume that the Turing machine begins in the **initial state** q_0 , with the machine head covering the leftmost square, \triangleright .

Formally, a **program** consists of a collection of **transition functions**

$$(Q \setminus \{q_h\}) \times A \mapsto (Q \setminus \{q_0\}) \times A \times \{L, N, R\},$$

where the codomain tells the Turing machine which state to change to, which symbol to write, and whether to move left, right, or neither (L , R , or N) when the machine head encounters a symbol while in a given state. We may thus think of a program as a collection of quintuples of the above form. By convention, the first line of the program describes what to do with initial state q_0 and symbol \triangleright . A Turing program is best illustrated with an example. Consider the following program, which inputs a nonempty string without blanks from the alphabet $A = \{\sqcup, 1, 2, 3, \triangleright\}$. This program outputs that same string with all of the 1's replaced with 2's, and changes the rightmost cell to a "3":

$$\begin{aligned} \langle q_0, \triangleright \rangle &\mapsto \langle q_0, \triangleright, R \rangle \\ \langle q_0, 1 \rangle &\mapsto \langle q_0, 2, R \rangle \\ \langle q_0, 2 \rangle &\mapsto \langle q_0, 2, R \rangle \\ \langle q_0, 3 \rangle &\mapsto \langle q_0, 3, R \rangle \\ \langle q_0, \sqcup \rangle &\mapsto \langle q_1, \sqcup, L \rangle \\ \langle q_1, 1 \rangle &\mapsto \langle q_h, 3, N \rangle \\ \langle q_1, 2 \rangle &\mapsto \langle q_h, 3, N \rangle \\ \langle q_1, 3 \rangle &\mapsto \langle q_h, 3, N \rangle \end{aligned}$$

Proposition 1.1 *Each Turing machine can be encoded by a distinct, finite string of 1's and 0's.*

Proof: Let M be a Turing machine with alphabet $A = \{a_1, \dots, a_n\}$, Turing states $Q = \{q_0, q_1, \dots, q_k, q_h\}$, and individual program line quintuples $\Delta = \{\delta_1, \dots, \delta_r\}$. Define the **unary representation** of a natural number n to be the length- n string of 1's, $\underbrace{11 \dots 1}_n$. We assign a distinct unary representation

to every symbol from the alphabet, including the auxiliary symbols \sqcup and \triangleright , as follows:

- \sqcup is 1,
- a_1 is 11,
- a_2 is 111,
- \vdots

$$a_n \text{ is } \underbrace{1 \dots 1}_{n+1}, \text{ and}$$

$$\triangleright \text{ is } \underbrace{1 \dots 1}_{n+2}.$$

Similarly, we can find a unary representation for each element in Q :

$$q_0 \text{ is } 1,$$

$$q_1 \text{ is } 11,$$

$$\vdots$$

$$q_n \text{ is } \underbrace{1 \dots 1}_{n+1}, \text{ and}$$

$$q_h \text{ is } \underbrace{1 \dots 1}_{n+2}.$$

We will code the direction symbols L , N , and R by 1, 11, and 111, respectively.

We can thus represent any program line quintuple as a string of 1's and 0's by attaching the unary representation of each element in the ordered tuple end-to-end, with a 0 to mark the separation between each element in the series. Finally, we can represent the entire series of program lines Δ as a string of 1's and 0's by writing 00 as a separator between each quintuplet of the Turing program.

The Turing machine from the example above thus corresponds to the following sequence of 1's and 0's:

```
1011111010111110111001011010111011100101110101110111001011110101111011100
1010110101001101101110111011001101110111011101100110111101110111101100.
```

The sequence of 0's and 1's corresponding to a particular Turing machine is called the **Turing number** of that machine. ■

Corollary 1.2 *There are countably many Turing machines.*

Proof: There are only countably many finite strings of 0's and 1's. ■

2 Tools

We will need to code a few subroutines for use in the construction of our universal Turing machine.

Lemma 2.1 (copying machine) *Let X and Y be symbols on a tape, with X lying to the left of Y and only blanks and 1's written on the cells in between. Assume that the head of the Turing machine is initially covering the cell containing X . Then there is a Turing program that counts the number of contiguous 1's sitting to the immediate right of Y and pastes that same number of 1's to the right of X . We will assume that we start with sufficient blank cells to the right of X to contain the intended string of 1's. This copying process may also be implemented in the reverse direction, from X to Y .*

Proof: Let X and Y be as above, and let B be a symbol in the alphabet that is distinct from X , Y , \sqcup , and 1. The symbol B will be used to keep track of how many cells have been copied. The symbols X and Y will remain stationary throughout this construction. That is, X and Y will be neither written nor erased. The algorithm is as follows:

Step 1: Move to the right and stop at the first 1. Mark this cell with a B . (If no "1" is found, that is, we reach either Y or a blank cell, go to Step 4.)

Step 2: Move to the right and stop at the first blank cell beyond Y . Write a 1 in this cell.

Step 3: Move back to the left until the first B is encountered, then go to Step 1.

Step 4: Move to the left, changing B 's into 1's until the symbol X is reached, at which point halt.

The following time diagram illustrates a sample copy process. Lines (a)–(d) in the sequence shows the tape status at the beginning of Step 1, and the underlined character indicates the position of the the tape head at that moment:

(a)	1	<u>X</u>	1	1	1	□	1	1	Y	□	□	□	□	1
(b)	1	X	<u>B</u>	1	1	□	1	1	Y	1	□	□	□	1
(c)	1	X	B	<u>B</u>	1	□	1	1	Y	1	1	□	□	1
(d)	1	X	B	B	<u>B</u>	□	1	1	Y	1	1	1	□	1
(e)	1	<u>X</u>	1	1	1	□	1	1	Y	1	1	1	□	1

The steps in this process suggest formal program lines for our Turing machine:

$$\begin{aligned}
 \langle q_0, X \rangle &\mapsto \langle q_0, X, R \rangle \\
 \langle q_0, 1 \rangle &\mapsto \langle q_1, B, R \rangle \\
 \langle q_0, \sqcup \rangle &\mapsto \langle q_4, \sqcup, L \rangle \\
 \langle q_0, Y \rangle &\mapsto \langle q_4, Y, L \rangle \\
 \langle q_0, B \rangle &\mapsto \langle q_0, B, R \rangle \\
 \langle q_1, 1 \rangle &\mapsto \langle q_1, 1, R \rangle \\
 \langle q_1, \sqcup \rangle &\mapsto \langle q_1, \sqcup, R \rangle \\
 \langle q_1, Y \rangle &\mapsto \langle q_2, Y, R \rangle \\
 \langle q_2, 1 \rangle &\mapsto \langle q_2, 1, R \rangle \\
 \langle q_2, \sqcup \rangle &\mapsto \langle q_3, 1, L \rangle \\
 \langle q_3, 1 \rangle &\mapsto \langle q_3, 1, L \rangle \\
 \langle q_3, Y \rangle &\mapsto \langle q_3, Y, L \rangle \\
 \langle q_3, B \rangle &\mapsto \langle q_0, B, N \rangle \\
 \langle q_4, X \rangle &\mapsto \langle q_h, X, N \rangle \\
 \langle q_4, B \rangle &\mapsto \langle q_4, 1, L \rangle
 \end{aligned}$$

■

Lemma 2.2 (matching machine) *Let X and Y be symbols on a tape, with X lying to the left of Y and only blanks and 1's written on the cells in between. Assume that the head of the Turing machine is initially covering the cell containing X . Then there is a Turing program that compares the number of contiguous 1's sitting to the right of X to the number of contiguous 1's sitting to the right of Y . If these two quantities are the same, then the program terminates in one state. If they are different, then it terminates in another.*

Proof: Let X and Y be as above, and, as in Lemma 2.1, let B be a symbol in the alphabet that is distinct from X , Y , \sqcup , and 1. We'll show that if the string of 1's to the right of X is the same length as the string of 1's to right of Y , the matching machine terminates in state q_m ; otherwise it terminates in state q_n . The algorithm is as follows:

Step 1a: Move to the right and stop at the first 1. Mark this cell with a B . (If no "1" is found, that is, we reach either Y or a blank cell, then go to Step 2.)

Step 1b: Move to the right until the head reaches Y .

Step 1c: Move to the right and stop at the first 1. Mark this cell with a B . (If no “1” is found, that is, we reach a blank cell, then go to Step 3.)

Step 1d: Move to the left and stop at X . Go to Step 1a.

Step 2: Move to the right until the head reaches Y . Continue to the right and find the first 1. Return to X , and terminate in state q_n . (If no “1” is found, that is, we reach a blank cell, then return to X and terminate in state q_m .)

Step 3: Move to the left until the head reaches X , and then terminate in state q_n .

If we were to run the matching program on the following initial tape pattern, our tape would develop as follows:

- (a) 1 X 1 1 1 \sqcup 1 1 Y 1 1 1 1 \sqcup 1
- (b) 1 X B 1 1 \sqcup 1 1 Y B 1 1 1 \sqcup 1
- (c) 1 X B B 1 \sqcup 1 1 Y B B 1 1 \sqcup 1
- (d) 1 X B B B \sqcup 1 1 Y B B B 1 \sqcup 1

This process is similar to the copy machine (Lemma 2.1) in the sense that we use the symbol B to mark off the 1’s that we have already counted. ■

Lemma 2.3 (term substitution machine) *Let X and Y be symbols on a tape, with X lying to the left of Y , and only blanks and 1’s in between. Let S be a sequence of unary representations, with each term separated by 0, and suppose S lies immediately to the right of Y . Let k denote the single unary representation lying to the immediate right of X . Assume that the head of the Turing machine is initially covering the cell containing X . Then there exists a Turing program which replaces the first term in S with k .*

Proof: The term substitution machine will substitute for a unary term in a sequence using a two-stage process. In the first “collapsing” stage, the substitution machine completely removes the term next to the right of Y , shifting all of the remaining terms to the left to collapse the void. In the second stage, the substitution machine “inserts” the new term, pushing the remaining terms back just far enough to make space for the new term.

The collapsing stage works as follows. The far right end of the sequence S must be marked by some symbol, say 00:

Step 1: First, the Turing machine checks whether the cell right-adjacent to Y contains a 0 or a 1. If it's a 0, then the sequence is already collapsed, and so the machine proceeds directly to the insertion stage. Otherwise, it continues to Step 2.

Step 2: The head shifts to the far right end of the sequence and proceeds to shift each cell in the sequence one unit to the left. During this shifting process, the Turing machine “remembers” the content of each cell as it is erased by converting to an appropriate, designated state. The cell marked with Y , however, is not overwritten; when the head reaches this cell, this shifting step is complete. Go to Step 1.

The insertion stage is similar to the collapsing stage, but the cells are shifted in the opposite direction. Each time that the machine passes through the insertion cycle, another 1 in the unary term next to X is marked off with a B . After sufficient repetition, that is, when all of the 1's have been marked off, the insertion stage is complete. For example purposes, consider the following time diagram, which uses the initial tape pattern from the example in Lemma 2.2:

(a) initial pattern	1	<u>X</u>	1	1	1	0	1	1	Y	1	1	1	0	1	...	
(b) after collapsing	1	<u>X</u>	1	1	1	0	1	1	<u>Y</u>	0	1	...				
(c) after insertion	1	<u>X</u>	1	1	1	0	1	1	Y	1	1	1	0	1	...	■

3 Main Construction

Theorem 3.1 There exists a universal Turing machine.

Proof: We are now ready to construct a **universal Turing machine**. Our universal Turing machine U will input the Turing number of a Turing machine M plus some initial tape pattern S , and it will return the output of machine M upon initial pattern S .

We will prepare the initial tape for our universal machine U as follows. We divide the tape into three sections. The first section of the tape will be U 's memory **buffer**, the second segment will permanently contain the **machine description** for the machine M , namely M 's Turing number, and the final section will contain M 's **tape description**, where M 's computation is performed:

▷	buffer	machine description	tape description
---	--------	---------------------	------------------

Our universal Turing machine will consist of the alphabet

$$A = \{0, 1, X, Y, Z, B\},$$

where we use “0” to represent the blank symbol for U , and X to represent the symbol ▷. X , Y , and Z will serve as tape markers, and the symbol B will be an auxiliary symbol, as applied in the lemmas of Section 2. We will make no further explicit remarks regarding B .

If we assume that machine M has r distinct Turing states and s symbols in its alphabet, then we require that the buffer region be at least $(r + s + 2)$ cells long. We’ll use the buffer region to store the current state and current symbol from Turing machine M . The length indicated above will ensure sufficient space so that we can store any symbol-state pair in the buffer, with a 0 separating each element in the pair, and another 0 separating the pair from the machine description. Initially, the buffer section contains all 0’s (i.e., all blanks).

The machine description segment will contain M ’s Turing number, exactly as calculated in Proposition 1.1. In addition, M ’s Turing number will be preceded by the symbol Y and followed by the sequence 000.

The tape description will contain a description of M ’s tape pattern, with each cell written in unary representation (see Proposition 1.1), and each term separated by a 0. The only exception here is that we will not use unary code for the infinite string of blanks at the end the tape pattern; we will simply use an infinite string of blanks, 00000 . . . , to represent the blank tail. Finally, the unary representation of S will be preceded by the symbol Z .

Now we describe the program for U . Before beginning Step 1 below, U ’s head is prepared so that it covers the cell marked Y :

Step 1—Copy current state into buffer: U makes a copy of the unary term adjacent-right to Y and writes this term in the cells next to X (by Lemma 2.1). U then marks the end of the term stored in the buffer with a second auxiliary X , erases Y , and moves the head onto Z .

Step 2—Copy scanned symbol into buffer: U finds the unary term adjacent-right to Z and copies this term adjacent-right to the auxiliary X (by Lemma 2.1).

Step 3—Set X and Y markers: U erases the auxiliary X , leaving a blank cell in that position. U then writes a Y in the leftmost cell of the machine

description section. At this point, the buffer contains the current state and scanned symbol of M , in that order.

Step 4—Search for quintuple: U searches through the quintuples in the machine description section to find a match for the state-symbol pair stored in the buffer (by Lemma 2.2). In detail, for each quintuple, U compares the first term in the quintuple to the first term in the buffer. If they match, then U checks the second term as well. When a match is finally found, the symbol Y will be posted between the matching pair. (If no match is found, then machine M would halt with the current tape description output, so U halts at this point. If M ended in a halting state, then we record this by having U terminate in a halting state. Otherwise there was some error, and so U terminates in a designated error state).

Step 5—Erase buffer: U erases the contents of the buffer and shifts the marker Y two terms to the right so that it stands to the left of the new symbol to be written to the tape description.

Step 6—Write new symbol in tape description: U updates the tape description by substituting the term right-adjacent to Y at the position right-adjacent to Z (by Lemma 2.3). U then reads the directional component of the marked quintuplet in the machine description and, in order to “remember” the direction for the next step, switches to a new, indicative state.

Step 7—Shift tape marker: U shifts the symbol Z one term to the left, right, or neither depending on which direction symbol has been read in the previous step. (If Z is forced to move left out of the tape description section in this step, then M 's head would have fallen off the tape, so U terminates in a designated error state).

Step 8—Locate next state: U moves the marker Y one term (within the quintuple) back to the right so that it rests to the left of the next machine state for M . With U 's machine head covering the symbol Y , U now returns to Step 1.

We now provide an example to illustrate the program described above. We'll look at the Turing program from Section 1, using initial tape pattern

▷ 1 2 □ □ □ □ ...

Each of the following lines shows the tape machine after the designated step has been processed. The first line in each step represents the buffer,

