COSC 385.001 – Automata

Spring 2017

Project #3

Due: Wednesday, April 11, 2018, 03:00 PM

Problem-11

Student: Jose Dixon

Instructor: Vojislav Stojkovic

Points: 15

**Part #2**

x>y; x and y are unsigned integer numbers

**Part 3**

# What is a Turing Machine?

## By Jack Copeland

#### © Copyright B.J. Copeland, July 2000

* [The head and the tape](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#head)
* [States](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#states)
* [Atomic operations](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#atom)
* [The instruction table](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#table)
* [An example](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#example)
* [Universal Turing machines](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#univ)
* [Computable numbers](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#comp)
* [Uncomputable numbers](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#uncomp)
* [Computable and uncomputable functions](http://www.alanturing.net/turing_archive/pages/reference%20articles/what%20is%20a%20turing%20machine.html#func)

Turing first described the Turing machine in an article published in 1936, 'On Computable Numbers, with an Application to the Entscheidungsproblem', which appeared in *Proceedings of the London Mathematical Society* (Series 2, volume 42 (1936-37), pp. 230-265).

### The head and the tape

A Turing machine is an idealised computing device consisting of a read/write head (or 'scanner') with a paper tape passing through it. The tape is divided into squares, each square bearing a single symbol--'0' or '1', for example. This tape is the machine's general purpose storage medium, serving both as the vehicle for input and output and as a working memory for storing the results of intermediate steps of the computation.

The input that is inscribed on the tape before the computation starts must consist of a finite number of symbols. However, the tape is of unbounded length--for Turing's aim was to show that there are tasks that these machines are unable to perform, even given unlimited working memory and unlimited time.

|  |
| --- |
|  |
| **A Turing machine** |

The read/write head is programmable. It is be helpful to think of the operation of programming as consisting of altering the head's internal wiring by means of a plugboard arrangement. To compute with the device, you program it, inscribe the input on the tape (in binary or decimal code, say), place the head over the square containing the leftmost input symbol, and set the machine in motion. Once the computation is completed, the machine will come to a halt with the head positioned over the square containing the leftmost symbol of the output (or elsewhere if so programmed).

### States

The head contains a subdevice that I call the *indicator*. This is a second form of working memory. The indicator can be set to any one of a number of 'positions'. In Turing machine jargon, the position of the indicator at any time is called the *state* of the machine at that time. To give a simple example of the indicator's function, it may be used to keep track of whether the symbol last encountered was '0' or '1'. If '0', the indicator is set to its first position, and if '1', to its second position.

### Atomic operations

There are just six types of fundamental operation that a Turing machine performs in the course of a computation. It can:

* read (i.e. identify) the symbol currently under the head
* write a symbol on the square currently under the head (after first deleting the symbol already written there, if any)
* move the tape left one square
* move the tape right one square
* change state
* halt.

These are called the *primitive* or *atomic* operations of the machine. A complicated computation may consist of hundreds of thousands, or even millions, of occurences of these atoms

Commercially available computers are hard-wired to perform primitive operations considerably more sophisticated than those of a Turing machine--add, multiply, decrement, store-at-address, branch, and so forth. The precise constitution of the list of primitives varies from manufacturer to manufacturer. It is a remarkable fact that none of these computers can outdo a Turing machine. Despite the Turing machine's austere simplicity, it is capable of computing anything that any computer on the market can compute.

Indeed, since it is an abstract or notional machine, a Turing machine can compute *more* than any physical computer. This is because (1) the physical computer has access to only a bounded amount of memory, and (2) the physical computer's speed of operation is limited by various real-world constraints.

It is sometimes said, incorrectly, that a Turing machine is necessarily slow, since the head is continually shuffling backwards and forwards, one square at a time, along a tape of unbounded length. But since a Turing machine is an idealised device, it has no real-world constraints on its speed of operation.

### The instruction table

A program or 'instruction table' for a Turing machine is a finite collection of instructions, each calling for certain atomic operations to be performed if certain conditions are met. Every instruction is of the form:

If the current state is n and the symbol currently under the head is x, then write y on the square currently under the head [y may be identical to x], go into state m [m may be n], and - - - .

In place of - - - may be written either 'move left one square' or 'move right one square' or 'halt'.

### An example

The machine in this example starts work with a blank tape. The tape is endless. The problem is to set up the machine so that if the scanner is positioned over any square and the machine is set in motion, it will print alternating binary digits on its tape, 0 1 0 1 0 1..., working to the right from its starting place, and leaving a blank square in between each digit.

In order to do its work the machine makes use of four states, labelled '**a**', '**b**', '**c**' and '**d**'. The machine is in state **a**when it starts work.

The table of instructions for this machine is as follows.The top line of the table reads: if you are in state **a** and the square you are scanning is blank then print 0 on the scanned square, move right one square, and go into state **b**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **state** | **scanned symbol** | **print** | **move** | **next state** |
| **a** | blank | **0** | **R** | **b** |
| **b** | blank |  | **R** | **c** |
| **c** | blank | **1** | **R** | **d** |
| **d** | blank |  | **R** | **a** |

A machine acting in accordance with this table of instructions toils endlessly on, printing the desired sequence of digits and leaving alternate squares blank.

### Universal Turing machines

There are in fact two ways of arranging for a Turing machine to act in accordance with a machine table or program. One, as already mentioned, is to appropriately modify the 'wiring' in the head of a Turing machine. The other is to translate the machine table into, say, binary code and inscribe the result on the tape of a special type of Turing machine known as a *universal* Turing machine.

Turing was able to demonstrate that there is a table U which is such that if the head of a Turing machine is programmed in accordance with U, and if any table whatsoever, P, is translated and written out on the machine's tape, then the machine will behave *as if* its head had been programmed in accordance with P. A universal Turing machine is any Turing machine whose head has been programmed in accordance with U.

There are many different tables that will do the work of U and thus many distinct universal Turing machines, all equivalent in computational power. The most economical such table is due to Marvin Minsky. Using an alphabet of four symbols and making use of only seven states, this table consists of just 28 instructions!

Turing's greatest contribution to the development of the digital computer were

1. this idea of controlling the function of a computing machine by storing a program of symbolically, or numerically, encoded instructions in the machine's memory, and
2. his proof that, by this means, a *single* machine--a universal machine--is able to carry out every computation that can be carried out by any other Turing machine whatsoever.

### Computable numbers

A Turing machine program is said to *terminate* just in case any Turing machine running the program will halt no matter what the input. An easy way to write a *non-terminating* program is simply to omit an instruction to halt. In the real world, computer programs that never terminate by design are commonplace. Air traffic control systems, automated teller machine networks, and nuclear reactor control systems are all examples of such.

An example of a non-terminating Turing machine program is a program that calculates sequentially each digit of the decimal representation of pi (say by using one of the standard power series expressions for pi). A Turing machine running this program will spend all eternity writing out the decimal representation of pi digit by digit, 3.14159 . . .

Turing called the numbers that can be written out by a Turing machine the *computable* numbers. That is, a number is computable, in Turing's sense, if and only if there is a Turing machine that calculates in sequence each digit of the number's decimal representation.

### Uncomputable numbers

Straight off, one might expect it to be the case that *every* number that *has* a decimal representation, either finite or infinite--that is to say, every *real* number--is computable. (The real numbers comprise the integers, the rational numbers--which is to say, numbers that can be expressed as a ratio of integers, for example 1/2 and 3/4--and the irrational numbers, e.g. the square root of 2 and pi, which cannot be expressed as a ratio of integers.) For what could prevent there being, for any particular real number, a Turing machine that 'churns out' that number's decimal representation digit by digit?

However, Turing was able to prove that not every real number is computable. The decimal representations of some real numbers are so completely lacking in pattern that there simply is no finite table of instructions of the sort that can be followed by a Turing machine for calculating the nth digit of the representation, for arbitrary n.

In fact, computable numbers are relatively scarce among the real numbers. There are only countably many computable numbers, whereas--as the mathematician Georg Cantor showed in 1874--there are uncountably many real numbers. (A set is countable if and only if either it is finite or its members can be put into a one-to-one correspondence with the integers.)

### Computable and uncomputable functions

Following Turing, to say that a mathematical *function*, for example addition over the integers, is computable is to say that there is a Turing machine which is such that if, for any pair of integers x and y, the machine is given x and y as input, it will print out the value of x+y and halt.

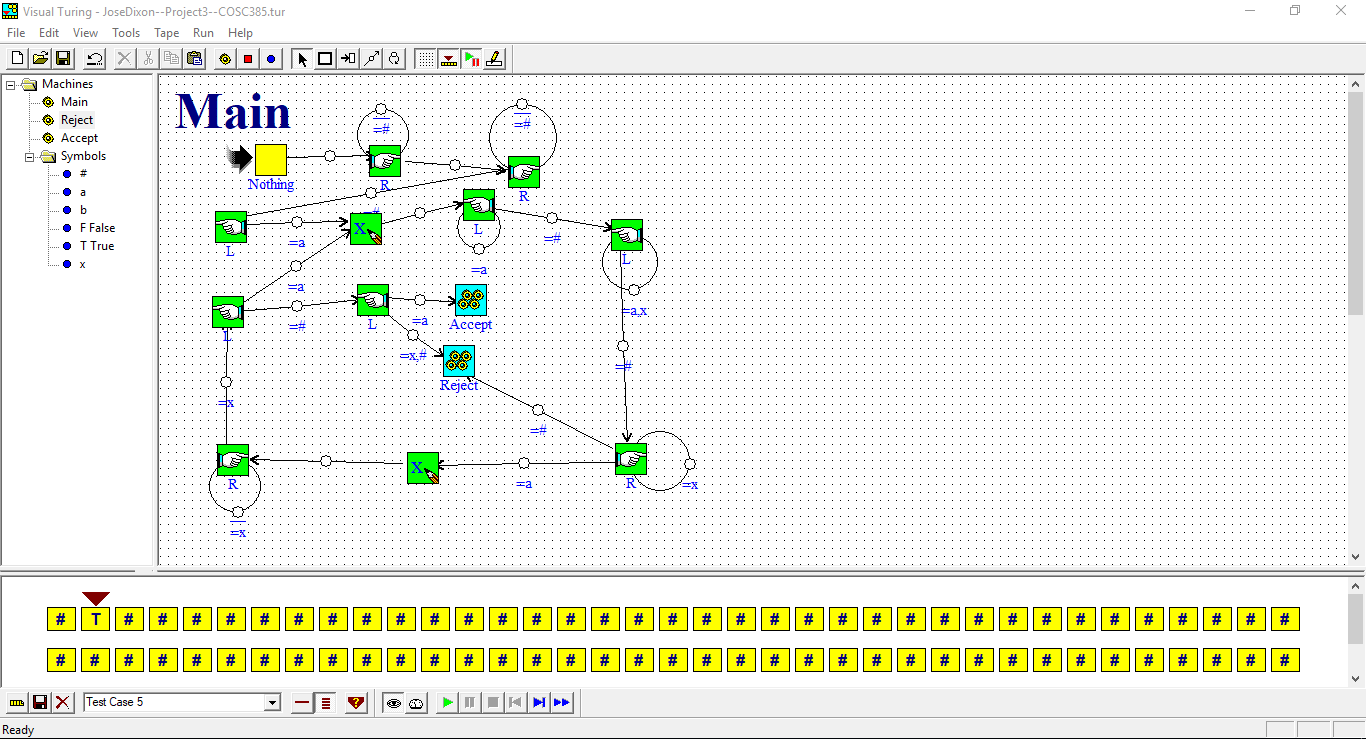
Addition over the integers is a computable function, whereas addition over the real numbers is not a computable function. This is because there is no way even of inscribing the inputs x and y on a Turing machine's tape if x and y are uncomputable numbers. (Remember that the input must consist of a finite number of symbols.)

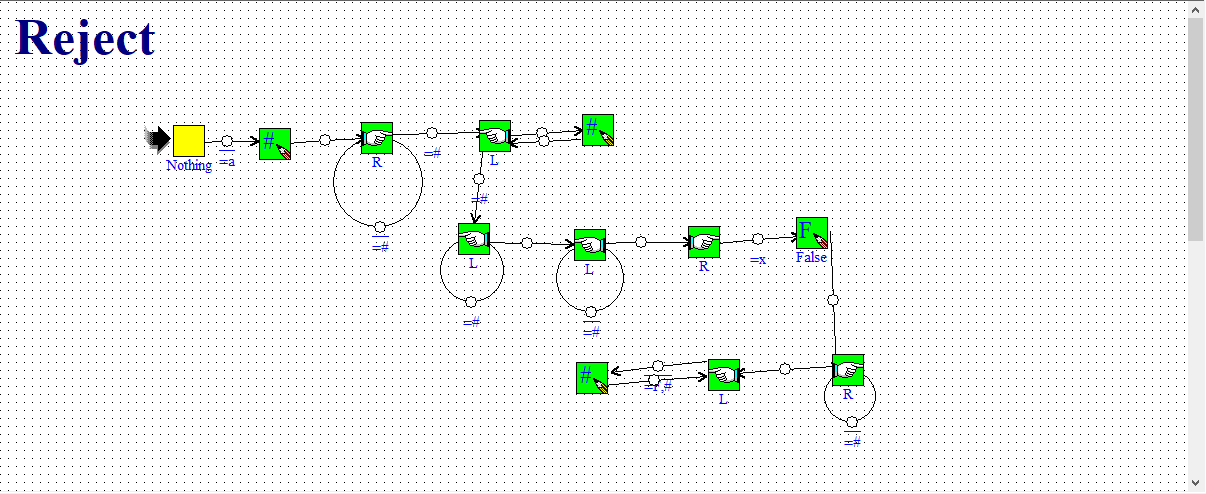
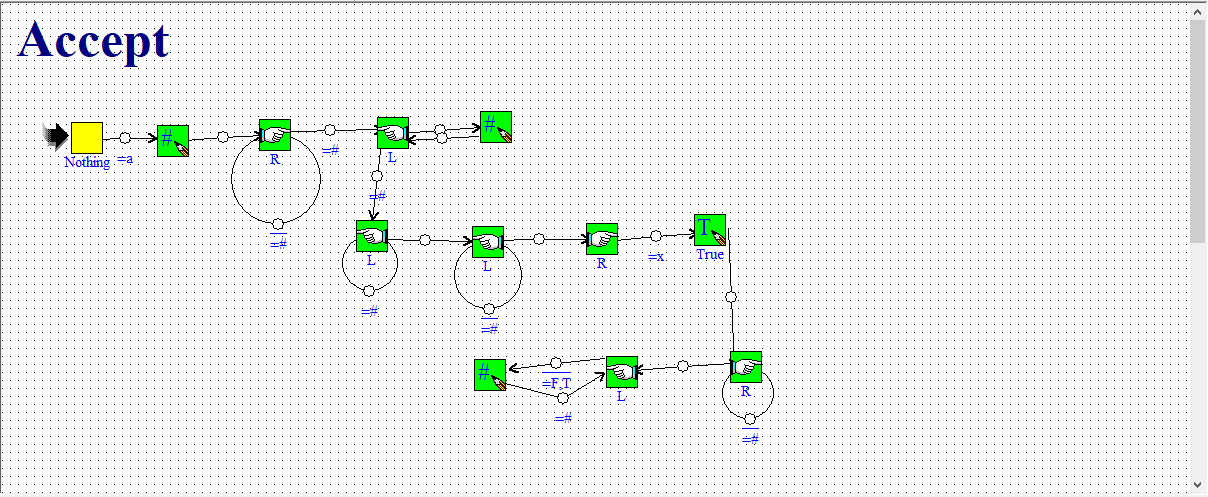
Is addition over the *computable numbers* a computable function? That is, is x+y computable whenever x and y are both computable numbers?

The answer is *yes*, but off the top of one's head one might think otherwise, for if x (or y) is a computable number having an infinite decimal representation, for example ¹, how can x be input, given the restriction that the input inscribed on the tape must consist of a finite number of symbols? The solution is to input x in the form of a *program*. The computable number x may be represented by means of a program which, if inscribed on the (otherwise blank) tape of some paricular universal Turing machine, would cause the universal machine to calculate the decimal representation of x digit by digit.

Thus Turing has given us a new method for representing numbers. In this system, there are *finite* representations of some numbers which, in the decimal system, can be represented only by means of an infinite sequence of symbols.

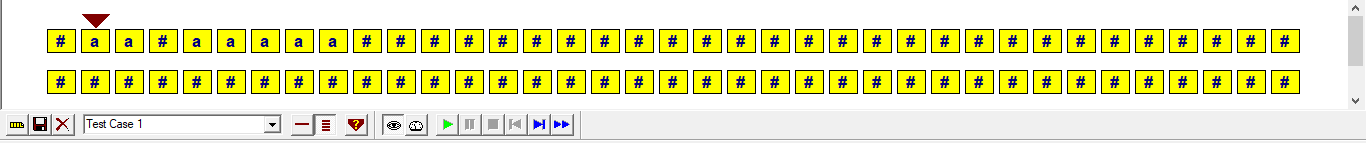
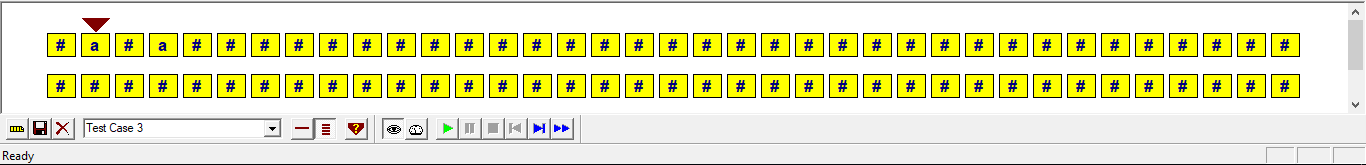
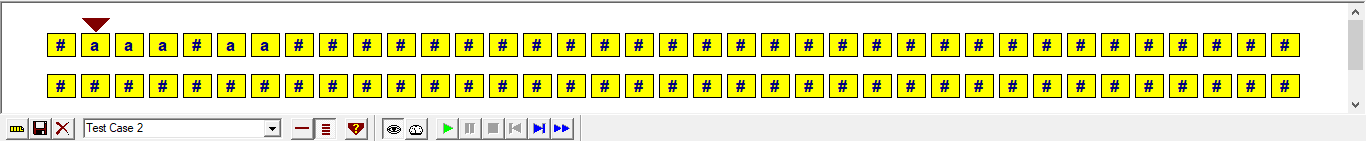
**Part #4**





**Part #5**

**Input**



**Output**

