Turing Machines

Table of Contents

What are they for?

They are a way to answer fundamental questions about computers, like 'What can a computer do?'and 'what can no computer ever do?' The idea is to be able to answer such questions not as a matter of opinion, but with a mathematical proof.

This depends on what a 'computer' is. For example, does it matter if it has a red or black case? If the case is plastic or metal? If it has an infinite amount of storage? A Turing machine is an abstract model of a computer.

What is it?

There are slightly different versions, by people such as Turing, Post, Minsky and Wolfram. The versions are formally equivalent.

A Turing machine has a *tape*, which is indefinitely long. The tape is a series of cells, on which *symbols* can be written. A cell can also be blank, with no symbol on it.

There is a tape *head* which can read and write symbols on tape cells, and the tape head can move left or right (or not move at all).

What the machine does is controlled by a set of *quintuples.* The machine executes a series of steps, in each of which it

- Reads the symbol under the tape head
- Then depending on its state, and the symbol it has read, it writes a new symbol on that cell (or maybe not)
- moves the head left or right, or not, and
- changes to a new state

A quintuple controls what it does for any state and symbol. One state is a 'halt' state, which stops the machine.

A quintuple has 5 items, which mean

[current state, symbol read, new state, symbol written, move left or right].

For example [5 A 3 P R] means that in state 5, if we read an A, we switch to state 3, write a P, and move the head right. The 5th element is L for move left and R for right.

A quintuple is in effect a function, of current state and symbol, and outputs the new state, the symbol to write and the direction to move.

The set of quintuples is the 'program' of the machine. But unlike normal programs, the order of quints does not matter, because only one will match the current state and symbol read.

A quintuple like

[5 1 5 1 R]

means in state 5, if we read 1, we stay in state 5, write 1 (so leave it unchanged in fact) and go right. In other words in state 5, go right over any 1s until we read something differernt.

The web page

The user can input the initial tape. The tape head starts at the left-hand cell. The tape is an infinite number of blank cells, the input string (if any), and another infinite string of blank cells. So if the user sets the initial tape as

ABC

then in fact it is infinite blanks, 3 cells containing A B and C, and infinite blanks. So that a blank cell can be seen, it is written as a full stop character '.'

Check -

0 . 1 h R

means reading a blank in state 0, but

0 0 1 h R

means reading a symbol 0 in state 0

The tape head starts at the leftmost input cell (the A in this example).

States are labelled as an integer 0 to n-1, where the machine has n states, plus the halt state H. The number of states can be input. This is to help validate quint input.

The machine starts in state 0. The halt state is labelled H. The quintuples can be input and edited as text, one quintuple per line, elements separated with spaces, as in the default example. A line can include a comment, starting with a #. These are to make quints easier to understand, and are ignored.

The machine can be 'set', meaning configured with the initial tape and quintuples.

It is possible to execute single steps, or run to a halted state. This is preceded by an attempt to set the initial machien configuration, if needed. This might fail if an invalid quint is found.

Example machines

Hello world

This is the default. The initial tape is blank, and the quints go:

```
0 . 1 h R # print h and move right
1 . 2 e R # print e and move right
2 . 3 l R # and so on..
3 . 4 l R
4 . 5 o R
5 . 6 W R
6 . 7 o R
7 . 8 r R
8 . 9 l R
9 . H d R # print d and halt
```

It starts in state 0, with a blank cell, so it first executes

```
0 . 1 h R
```

This prints an h on the cell, moves right, and switches to state 1. So the next quint to execute is

```
1 . 2 e R
```

which outputs e, moves right and changes to state 2.

And so on. In state 9 it executes

```
9 . H d R
```

This outputs the final d and enters the halt state

Parity checker

This machines inputs a sequence of 1s and 0s, and outputs a 0 if the parity is even (even number of 1s) and outputs a 0 if the parity is odd.

The initial tape is the sequence of 1s and 0s.

The quints are

```
0 1 1 . R # go right, switch to state 1 for a 1 read
0 0 0 . R # go right. stay in state 0 for a 0
0 . H 0 R # halt when reach a blank cell, print 0
1 1 0 . R # state 1 switches back to 0 when reads a 1
1 0 1 . R # but stays in state 1 when reads a 1
1 . H 1 R # halts, prints parity 1 n blank
```

(these can be copy pasted into the web page).

The machine has 2 states - the initial 0, and the other is 1 – plus the halt state H.

In state 0, reading a 1 switches to state 1 (and writes a blank and moves right). If it reads a 0, it stays in state 0. If it reads a blank (end of input) it writes a 0.

State 1 does the reverse – reading a 1 switches to 0, and reading a 0 switches to 1. A blank halts and writes 1.

So it is in state 0 if the machine has read 0, or 2, or 4, or 6 1s – any even number.

And it is in state 1 for 1, 3, 5 1s – any odd number

Brackets matcher

This machine inputs a string of brackets and outputs whether they pair up correctly. Valid strings are for example (()()) and (())(()), while invalid ones are ((() and))((.

Initial tape:

The string of brackets

Quints:

```
0 ) 1 X L
0 ( 0 ( R
0 . 2 . L
0 X 0 X R
1 ) 1 ) L
1 ( 0 X R
1 . H 0 -
1 X 1 X L
2 ) H 0 -
2 ( H 0 -
2 . H 1 -
2 X 2 X L
```

How does it work? There are 3 states. State 0 moves right from a (until it finds a), which it erases with an X and switches to state 1

State 1 moves left to a (, erases it with X and switches back to state 1.

This repeats until:

State 1 cannot find a (before reaches a blank. It prints 0 = invalid, and halts

Or state 0 cannot find a) before reaching a blank. It switches to state 2, moves left

State 2 moves left over any Xs. If it reaches a blank, a 1 = valid is printed and halts. If it finds a (it outputs a 0 and halts.

It can never find a) - this is impossible

Adder

This is a machine to do arithmetic – to add two natural numbers.

The Turing machine is type free. It knows nothing about numbers, or any other data type.

But we can program it to do arithmetic with numbers.

We need a way to represent numbers. In real computers we can use ones complement, or twos complement, or BCD. A choice in a Turing machine is to represent the number n by n tape cells containing a 1. We have a 0 between them. So

11101111

is 3 followed by 4.

To add these is very simple. We have
<n 1s>0<m 1s>

and we want to change that to

0<n 1s><m 1s>

so we just need to change the initial 1 to a 0, and the 0 in the middle to a 1

Quints

0 1 1 . R # start. Write blank over first 1, go right, into state 1
1 1 1 1 R # go right over 1s
1 0 H 1 R # replace 0 by 1

Busy beaver

Suppose we have a machine with n states writing two symbols 0 and 1, with an initial blank tape, coded so it halts. Different machines like this, with different quints, will output different numbers of 0s and 1s. So there must be a machine (maybe more than 1) which outputs the *maximum* number of 0s and 1s.

For example, 2 state maximum 1s

0 1 1 1 R
0 . 1 1 L
1 1 H 1 -
1 . 0 1 R

This writes 4 1s before halting.

There are two issues:

1. The number of 1s output. This is written Σ(n), Rado's sigma function

2. The number of steps it takes before halting, written S(n)

For some n, different machines produce maximum 1s and maximum steps. Both are called busy beavers, by different people

The 3 state maximum output is

0 . 1 1 R
0 1 2 1 L
1 . 0 1 L
1 1 1 1 R
2 . 1 1 L
2 1 H 1 R
which outputs 6 1s before halting, after 13 steps

But the 3 state machine going through the most steps is different – it is

0 . 1 1 R
0 1 H 1 R
1 . 1 1 L
1 1 2 . R

2 . 2 1 L
2 1 0 1 L
halts after 21 steps – but only outputs 5 1s.

A 6 state machine outputs more than 10^{865} 1s (Marxen, H. "Busy Beaver." http://www.drb.insel.de/~heiner/BB/.)