# Turing Machines

keystroke

## Definition of a Turing Machine

On what makes a Turing machine, Alan Turing makes it clear in his original 1936 paper on the subject[1], wherein he lays out a theoretical framework for a machine that can calculate numbers in any way.

This machine is described as an infinite reel of tape with squares that contain a symbol in them, a head that can move one square to either side at a time, read the symbol in the square it is on and write over it. This machine takes instructions based on its current state and the symbol underneath it, and on that information can write a symbol onto the tape, move left or right, and change state.

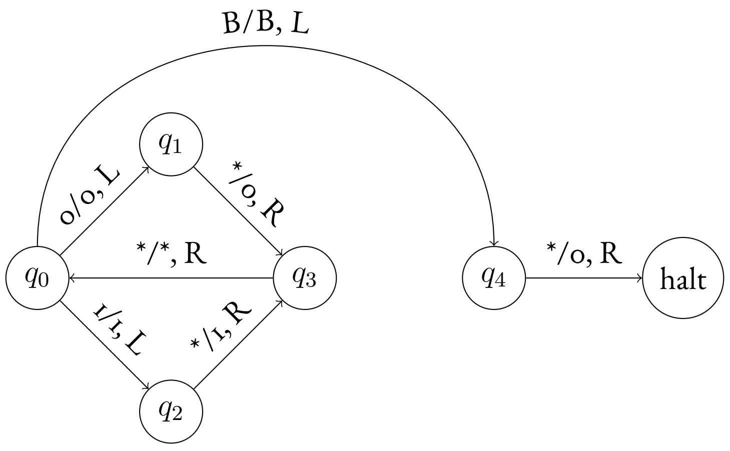
With this rather simple machine one might think, you can write an algorithm to calculate anything calculable. In fact, the Turing machine is so all-encompassing that whether or not a program or algorithm is deemed calculable is if it can be executed on a Turing machine[2].

This is what identifies a Turing machine in Turing's original theory. Since an infinite tape reel is impractical, implementations of this have had to settle for merely very large tape reels, or a lot of memory in the case of digital Turing machines. A language or computer system is considered Turing-complete when you can program a Turing machine into it, and can therefore theoretically solve any calculable sequence provided enough time and memory. This is what defines a Turing machine in the more practical sense.

## Designing a Turing Machine

This Turing machine has the tape language of $\Gamma = \{0, 1, B\}$. Here I have used $B$ to represent a blank space on the tape and used the * as a wildcard to represent all symbols as shorthand.

The TM is a simple algorithm for shifting a binary pattern to the left, which will multiply an inputted number by 2.



I have also tested this algorithm computationally using an online Turing machine visualiser[3] that allows you to see the structure and step through your algorithm one change at a time. I have included the source code for my algorithm below that can be used on this website to verify how the algorithm works.

```
input: '1011'
blank: ' '
start state: read
table:
    read:
        0:   {write: 0,   L: shift0}
        1:   {write: 1,   L: shift1}
        ' ': {write: ' ', L: tidy}

    shift0:
        [0,1,' ']: {write: 0, R: return}

    shift1:
        [0,1,' ']: {write: 1, R: return}

    return:
        [0,1,' ']: {R: read}

    tidy:
        [0, 1]: {write: 0, R: halt}

    halt:
```

## Turing Machine Language

The language recognised by a Turing machine can be any symbol specified in the tape language. However, I suspect that by this question you also mean how Turing machines can be built to accept certain languages, running through a kind of error-checking process to validate the input before accepting it and moving on to the function.[4]

There are two types of languages for Turing machines: *Recursively enumerable* and *decidable* (or *recursive*). Recursively enumerable languages are tricky to verify, as while we can check if they are accepted, there is no failure state. An invalid input will cause the Turing machine to loop infinitely, as due to the Entscheidungsproblem it is mathematically impossible to determine if a program will or will not halt.[1]

Decidable languages are far easier, as they have a defined rejection state that will, at the end of reading the input, definitively tell you if the input is valid language or not.

Now that we have got the basic definitions of Turing machine language out of the way, we can look at how this language is structured. Since drawing complete graphs for examples would be time-consuming and I have to get this done in around 24 hours, I will be using high-level abstraction. For what Turing machine state graphs look like, see the previous learning objective.

For example, a machine accepting the input $L = \{01^n0 | n \geq 0\}$ means that the input must consist of a zero, any number of ones equal or above zero, and another zero. A high-level description of how a Turing machine parses this:

1. If the scanned symbol is a 1 or blank, reject. If the symbol is a 0, move right and switch to the next state.
2. If the scanned symbol is a blank, reject. If the symbol is a 1, move right and keep the same state. If the symbol is a 0 move right and switch to the next state.
3. If the scanned symbol is 1 or 0, reject. If the symbol is a blank, accept.

This is a decidable language, as it has a clear reject and accept state that it will halt at when the algorithm is finished.

There are other permutations of Turing machines that accept different kinds of language, such as a multi-tape permutation that can simultaneously read and write $n$ amount of symbols, where $n$ is equal to the amount of tapes there are. Each read/write head may or may not be able to move independently depending on the specification.

## Resources

[1] A. M. Turing (1936). *On Computable Numbers With an Application to the Entscheidungsproblem* [Online]. Available: `https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf`
The original paper by Alan Turing, read to see how the original design was staked out and what exactly identifies a Turing machine.

[2] B. Eater (2018). *Making a computer Turing complete* [Video]. Available: `https://www.youtube.com/watch?v=AqNDk_UJW4k`
Youtube video by Ben Eater on the instruction set required to make a super basic computer Turing Complete. Watched to get a basic grasp on how they worked and how they related to current computing, with instruction cards the equivalent of operation codes.

[3] A. Lee (2016). *Turing machine visualization* [Website]. Available: `http://turingmachine.io/`
Turing machine visualiser that accepts code of a basic syntax that allows you to display and step through Turing machines. Used to practically test the turing machine in for Learning Objective 2.

[4] B. Xiaohui (2020). *MAS714: Algorithms and Theory of Computation* [Lecture]. Available: `https://www3.ntu.edu.sg/home/xhbei/MAS714/`
Lecture slides and an exercise on Turing machines provided by the Nanyang Technological University of Singapore.

- `https://www3.ntu.edu.sg/home/xhbei/MAS714/slides/slides17.pdf`
  Read through these lecture slides to understand how Turing machines are mathematically represented.

- `https://www3.ntu.edu.sg/home/xhbei/MAS714/homework/exercise2.pdf`
  An exercise on Turing machine languages that I will be using as an exam for this activity.