Turing Machines

A much more powerful model of computation than finite automata is provided by Turing machines. While we can (and will) also picture them as state machines, in contrast to finite automata Turing machines have unbounded, (potentially) infinite memory. We now give a precise, yet informal definition of a Turing machine.

A *Turing machine* consists of an infinite memory tape divided into discrete cells which can be accessed by a read/write head for reading and writing symbols one at a time. It can be programmed by specifying a finite number of transition rules of the following form

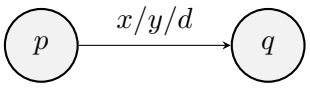
$$R(p, x, y, d, q)$$

which is interpreted as: If the machine is in state p and its head is located at a cell containing the symbol x, then the head overwrites x by the symbol y, moves to the next cell in direction d, and the machine transitions to state q.

- The states of the machine are finitely many specified by the programmer, including the starting state "0" and the halting state "HALT" (for decision problems there are two halting states: "ACCEPT" and "REJECT").
- The symbols on the tape are "0" or "1" (although machines can also use larger alphabets), or the special symbol "#" used to indicate an empty cell.
- The direction d may be either "Left" or "Right".

Initially, the machine is in the initial state, the tape contains the input string, and the head is located at the first cell of the input string. When the machine halts, the output string is read from the tape.

Just like finite automata, Turing machines can also be represented by state machines: We draw a circle for each state; We denote the initial state by an incoming arrow (out of nowhere); For each rule $R(p, x, y, d, q)$, we draw the transition from p to q, by an arrow from state p to state q labelled by $x/y/d$.



Turing machines are a theoretical model of computation. A more realistic model is the von Neumann architecture, which is the basis for modern computers. Turing machines are significant as one of the first attempts to formalise the concept of computation. They are on the one hand easy to specify and can be handled with mathematical rigour, and on the other hand are programmable in a rather intuitive fashion. Note that many variations of the definitions above exist, but they all are essentially equivalent.

Turing machines enable the notion of being computationally complete: A model of computation is (Turing) complete if it can simulate any Turing machine. Normal algorithms are computationally complete, and all programming languages are. The Church-Turing thesis conjectures that any function whose values can be computed by an algorithm can be computed by a Turing machine, and therefore that if any real-world computer can simulate a Turing machine, it is Turing equivalent to a Turing machine.

For specifying and simulating Turing machines we will use the following online simulator: `http:/turingmachine.vassar.edu/`.

Task 1 Specify a Turing machine that inverts a given binary string by turning each "0" into a "1" and each "1" into a "0". For example, on input "11100" the machine should output "00011".

Task 2 Modify your machine from Task 1 (if necessary) to move back its read/write head to the beginning of the output before halting.

Task 3 Specify a Turing machine that reverses a given binary string. For example, the reversal of "11100" is "00111".

Hint: One way of solving this task is to repeatedly do the following: read the last symbol from the input string, memorise it, remove it from the cell, move to the end of the output string (initially empty), write the symbol, and then move back to the end of the input string. To follow this scheme, you need to memorise the symbol read in each iteration in a special state (say, "READ0" and "READ1"). You also need to make sure that you reliably detect the start and end of the input string, no matter how long it is. Remember that the input string is surrounded by empty cells (with the special symbol "#"). It might be helpful to separate the rest of the input string and the output string constructed so far with an empty cell as well.

Task 4 Construct a Turing machine for adding two natural numbers in unary notation (the number n is represented by n 0s). In the input string, the two numbers are separated by a single empty cell ("#").