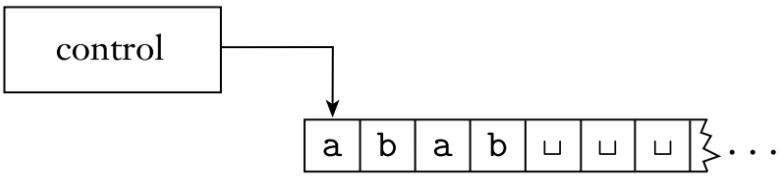
Turing Machines

There exists a much more powerful model of computation, first proposed by Alan Turing in 1936, called the Turing machine. A Turing machine is a much more accurate model of a general purpose computer.

Description: A Turing machine consists of three parts:

1. A finite-state control that issues commands,
2. An infinite tape with input and blank cells,
3. A tape head that can read and write a single tape cell.



At each step, the Turing machine

1. Reads and writes a symbol to the tape cell under the tape head,
2. Changes state,
3. Moves the tape head to the left or to the right.

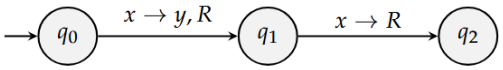
Definition: A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the blank symbol ,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Difference with finite automata:

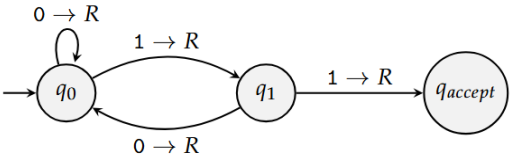
1. A Turing machine can both read from the tape and write on it.
2. The tape head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

Designing Turing machines: Consider the following state diagram.

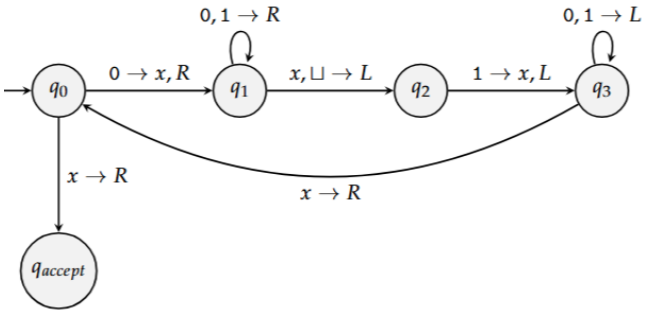


In the diagram, the label x → y, R appears on the transition from q_0 to q_1. This label signifies that when in state q_0 with the head reading x, the machine goes to state q_1, writes y, and moves the head to the right. In other words, $\delta(q_0, x) = (q_1, y, R)$. For simplicity, we use the shorthand x → R in the transition from q_1 to q_2 to mean that the machine moves to the right when reading x in state q_1 but doesn't alter the tape, so $\delta(q_1, x) = (q_2, x, R)$. Moreover, we generally omit the reject state and the transitions going to the reject state from the state diagram.

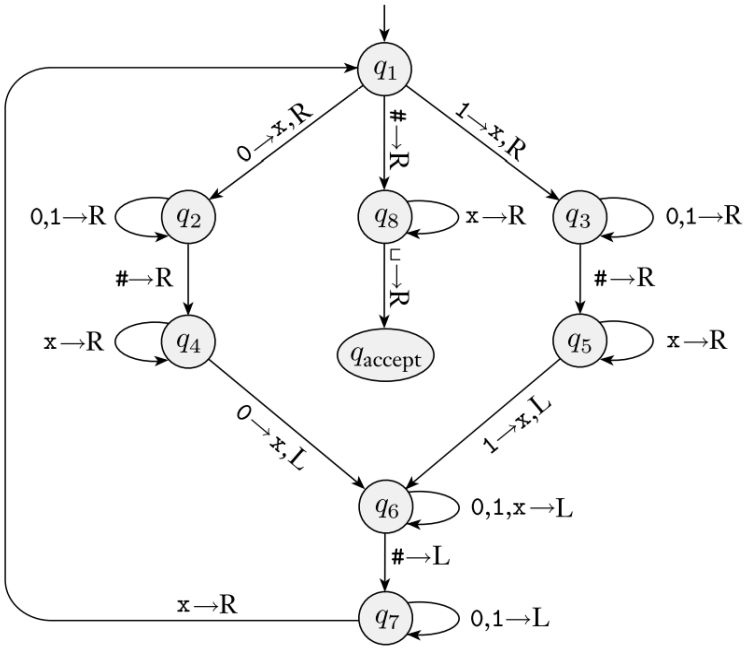
Example 1: L(M) → {w ∈ {0,1}* | w contains 11}



Example 2: L(M) → {0^n1^n | n > 0}



Example 3: L(M) → {w#w | w ∈ {0,1}*}



We can formally describe a particular Turing machine by specifying each of its seven parts. However, going to that level of detail can be cumbersome for all but the tiniest Turing machines. Mostly, we informally describe the Turing machines which sketches the way it functions but does not give all its details. The algorithm for the above-defined machine (M) can be summarized as follows:
M = "On input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, reject. Cross off symbols as they are checked.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, reject; otherwise, accept."

Turing recognizable and decidable: When we start a Turing machine on an input, three outcomes are possible: accept, reject, or loop. By loop it means that the machine simply does not halt. A Turing machine can fail to accept an input by rejecting, or by looping.

The language accepted by a Turing machine is called Turing recognizable or recursively enumerable. Moreover, if the machine halts on all inputs, the language is called Turing decidable or recursive.

Universal Turing machines: Turing proved that we can build a Turing machine (U) that acts as an interpreter for other Turing machines. In other words, U's input tape can contain a description of another Turing machine, which is then simulated step by step. Such a machine U is called a Universal Turing machine.

Closure of Turing recognizable and decidable languages:
Turing recognizable and decidable Languages are closed under union, intersection, concatenation, and Kleene star operations. Though Turing decidable languages are closed under complement operation, Turing recognizable languages are not.

Church–Turing thesis: The Church–Turing thesis is a fundamental claim that is stated as: Any effectively calculable function can be computed by a Turing machine. Church used the λ-calculus to define algorithms whereas Turing did it with his Turing machines.