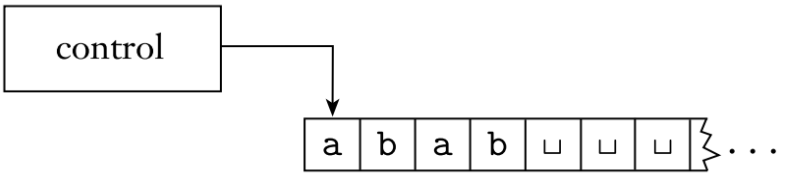
Turing Machines

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.

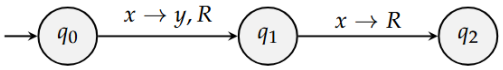
Definition: A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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:

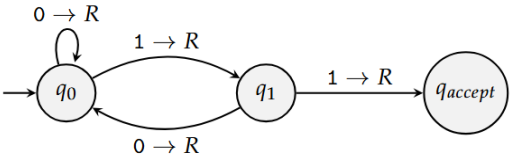
1. A Turing machine can both read from the tape and write on it.
2. The tape head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

Designing Turing Machines: Consider the following state diagram.

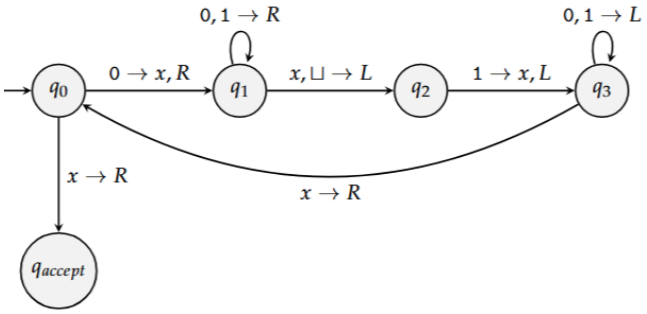


In the diagram, the label x → y, R appears on the transition from q_0 to q_1. This label signifies that when in state q_0 with the head reading x, the machine goes to state q_1, writes y, and moves the head to the right. In other words, $\delta(q_0, x) = (q_1, y, R)$. For simplicity, we use the shorthand 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.

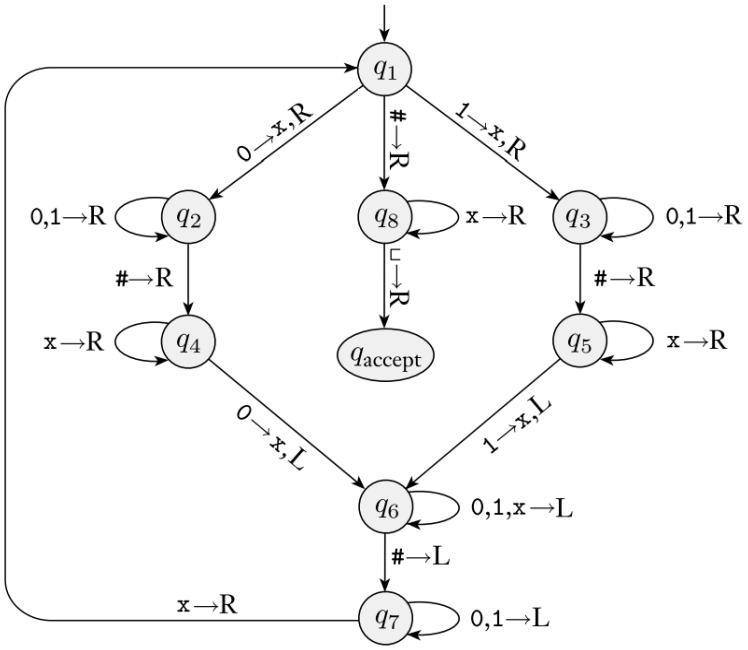
Problem 1: L(M) → {w ∈ {0,1}* | w contains 11}



Problem 2: L(M) → {0^n1^n | n > 0}



Problem 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."

Configuration of Turing Machines: As a Turing machine computes, changes occur in the machine's state, the tape contents, and the head's position. A particular setting of these three components is called a configuration of the Turing machine. For a state q and two strings u and v, we write uqv to denote a configuration where the current state is q, the tape content is uv, and the head is positioned on the first symbol of v. For example, $011q_701$ represents a configuration where the tape contains 01101, the current state is q_7, and the head is on the second 0.

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.

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 language and the machine is called recognizer. Moreover, if the machine halts on all inputs, the language is called Turing decidable or recursive language and the machine is called decider.

Notations for Encoding: The input to a Turing machine is always a string. If we want to provide an object other than a string as input, we must first represent that object as a string. A Turing machine may be programmed to decode the string representation of the object. Our notation for the encoding of an object O into its string representation is ⟨O⟩. The first line of the algorithm

describes the input to the machine. If the input description is the encoding of an object as in ⟨O⟩, the Turing machine first implicitly checks whether the input properly encodes the object in the desired form or not.

Decidable Languages: One way to prove that a language is decidable is to construct a decider for it. Let A_{DFA} be the language of all string representations of any DFA D and any string w, such that D accepts the string w. The decider for A_{DFA} decides whether a pair of strings ⟨D, w⟩ is in the language of A_{DFA} or not.

Problem 4: Let, $A_{DFA} \rightarrow \{\langle D, w\rangle \mid D \text{ is a DFA that accepts input string } w\}$. Prove that A_{DFA} is decidable.
Proof: The following TM M_{DFA} decides A_{DFA}:
M_{DFA} = "On input s:

1. Check that s has the the form ⟨D, w⟩ where D is a DFA and w is a string; reject if not.
2. Simulate D on input w.
3. If the simulation ends in an accept state, accept. If it ends in a non-accepting state, reject."

Since the simulation always ends after $|w|$ steps, M_{DFA} is a decider.
If $w \in L(A)$, then M_{DFA} will accept. If $w \notin L(A)$, then M_{DFA} will reject.

Note: We generally use the shorthand "On input ⟨D, w⟩" to denote that the Turing machine has verified that the input string is in the form ⟨D, w⟩.

Problem 5: Let, $A_{REX} \rightarrow \{\langle R, w\rangle \mid R \text{ is a regular expression that generates the string } w\}$. Prove that A_{REX} is decidable.
Proof: The following TM M_{REX} decides A_{REX}.
M_{REX} = "On input ⟨R, w⟩:

1. Convert regular expression R to an equivalent NFA N using the Thompson's construction method.
2. Convert NFA N to an equivalent DFA D using the subset construction method.
3. Simulate TM M_{DFA} from problem 4 on input ⟨D, w⟩.
4. If M_{DFA} accepts, accept; if M_{DFA} rejects, reject."

Since the simulation always ends after $|w|$ steps, M_{REX} is a decider.
If $w \in L(A)$, then M_{REX} will accept. If $w \notin L(A)$, then M_{REX} will reject.

Universal Turing Machines: Let, $A_{TM} \rightarrow \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$. The following Turing machine U recognizes A_{TM}.

U = “On input ⟨M, w⟩:

1. Simulate M on input w.
2. If M reaches its accept state, accept; if M reaches its reject state, reject.”

The Turing machine U is an example of the universal Turing machine that was first proposed by Alan Turing in 1936. This machine is called universal because it is capable of simulating any other Turing machine using the description of that machine. The universal Turing machine played an important role in the development of stored-program computers such as the IAS machine.

The Halting Problem: The halting problem is the problem of determining whether a Turing machine halts by accepting or rejecting a given input. Let $HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$.

Theorem 1: $HALT_{TM}$ is undecidable.
Proof: Assume that the TM H decides $HALT_{TM}$. The following is a description of H:
H = “On input ⟨M, w⟩:

1. Simulate TM M on input w.
2. If M halts on w, accept; if M loop infinitely on w, reject.”

We construct a Turing machine D with H as a subroutine. D calls H to determine what M does when the input to M is its own description ⟨M⟩. D does the opposite, i.e. it halts if M loops infinitely and loops infinitely if M halts. The following is a description of D:
D = “On input ⟨M⟩:

1. Simulate TM H on input ⟨M, ⟨M⟩⟩.
2. If H rejects, D halts. If H accepts, D loops infinitely.

In summary,

$$D(\langle M\rangle) = \begin{cases} \text{halts if M loops infinitely} \\ \text{loops infinitely if M halts} \end{cases}$$

When we run D with its own description ⟨D⟩ as input, we get:

$$D(\langle D\rangle) = \begin{cases} \text{halts if D loops infinitely} \\ \text{loops infinitely if D halts} \end{cases}$$

However, this contradicts our assumption that H solves the halting problem for any Turing machine and its input. Therefore, our assumption must be wrong. Hence, H must not exist and $HALT_{TM}$ must be undecidable.