1 Turing Machines (3.1)

First proposed by Alan Turing in 1936, the **Turing machine** is a much more accurate model of a general purpose computer. It can do everything that a real computer can do, but even it cannot solve certain problems[1].

1.1 The Idea

- The input string is *written* on the leftost squares of the tape. The rest of the tape is empty.
- We can read *and* write on the tape. The read/write head starts at the leftmost position on the tape.
- Computation proceeds according to the transition function. In other words, given the current state of machine, and the current symbol being read, the machine will
  - Transition to a new state.
  - Write a symbol to its current position, overwriting the existing symbol.
  - Moves the tape head L or R.
- Computation ends if and when it enters either the **accept** or the **reject** state. This means that we can have programs that can run forever.

1.2 Language of a Turing Machine

Given a Turing machine M, the language $L(M)$ is the set of all strings w such that the computation of M on w *halts* after entering the accept state. That is, $L(M) = \{w \mid w \text{ is accepted by } M\}$.

1.3 Formal Definition

As usual, the most important thing about the Turing machine is the transition function

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$$

That is, when the machine is in a certain state q and the head is over a tape square containing the symbol a, and if $\delta(q, a) = (r, b, L)$, then the machine writes the symbol b replacing the a, and goes to state r. The third component is either L or R, and indicates whether the head moves to the left or right after writing. In this case, the L indicates that we move the tape to the left.

Definition 1.1: Turing Machine

A **Turing machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ 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* $\sqcup$.
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$.
4. $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ is the transition function.
5. $q_0 \in Q$ is the start state.
6. $q_{\text{accept}} \in Q$ is the accept state.
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

[1] In a very real sense, these problems are beyond the theoretical limits of computation.

1.4 Configuration of a Turing Machine

As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location. A setting of these three items is called a **configuration** of the Turing machine. They are often represented in a special way. For a state q and two strings u and v over the tape alphabet Γ, we write uqv for the configuration where the current state is q, the current tape contents is uv, and the current head location is the first symbol of v.

For example, $\texttt{1011}q_7\texttt{01111}$ represents the configuration when the tape is 101101111, the current state is q_7, and the head is currently on the second 0.

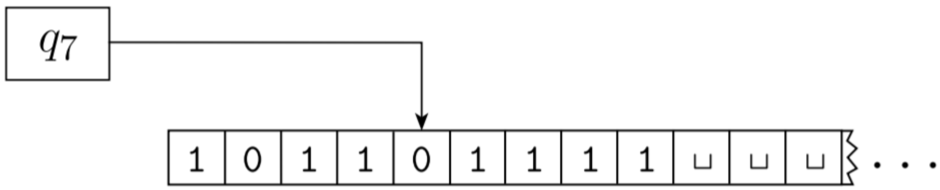


Figure: The configuration $\texttt{1011}q_7\texttt{01111}$.

1.4.1 Transitioning Between Configurations

Suppose that a configuration C_1 **yields** configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step. We can define this notion formally as follows: suppose we have $a, b, c \in \Gamma$, as well as $u, v \in \Gamma^*$ and states $q_i, q_j \in Q$. In that case, uaq_ibv and uq_jacv are two arbitrary configurations. Say that

$$uaq_ibv \text{ yields } uq_jacv$$

if, in the transition function, $\delta(q_i, b) = (q_j, c, L)$. This handles the case where the Turing machine moves leftward. For a rightward move, say that

$$uaq_ibv \text{ yields } uacq_jv$$

if, in the transition function, $\delta(q_i, b) = (q_j, c, R)$.

1.4.2 Start, Accepting, Rejecting, and Halting Configurations

The start configuration of M on input w is the configuration q_0w, which indicates that the machine is in the start state q_0 with its head at the leftmost position on the tape.

In an accepting configuration, the state of the configuration is q_{accept}.

In a rejecting configuration, the state of the configuration is q_{reject}.

Accepting and rejecting configurations are halting configurations and do not yield further configurations.

1.5 Deciders and Recognizers

We now briefly talk about the difference between deciders and recognizers.

1.5.1 Turing-Recognizable

A language is **Turing-recognizable** if some Turing machine recognizes it, i.e. if $L = L(M)$ for some Turing machine M. When we start a Turing machine on some input, the machine can either *accept*, *reject*, or *loop*. However, sometimes we don't want the machine to loop.

1.5.2 Turing-Decider

A Turing machine M is a **decider** Turing machine (either Turing-decidable or decidable) if it halts on all inputs (i.e. never loops).

L is **Turing-decidable** if some Turing machine that is a decider recognizes it.

1.5.3 Example 1: Turing Machine

Consider the language $L = \{w\texttt{\#}w \mid w \in \{\texttt{0}, \texttt{1}\}^*\}$, which is both not regular and not context-free.

1. Give an idea for a potential Turing machine that recognizes L.

   The idea is as follows

   - We want to zig-zag across tapes to corresponding positions on either side of `#` to check whether these positions agree. If they do not, or there is no `#`, then we reject. If they do, then cross them off.
   - Once all symbols to the left of the `#` are crossed off, check for any symbols to the right of `#`. If there are any, *reject*. Otherwise, accept.

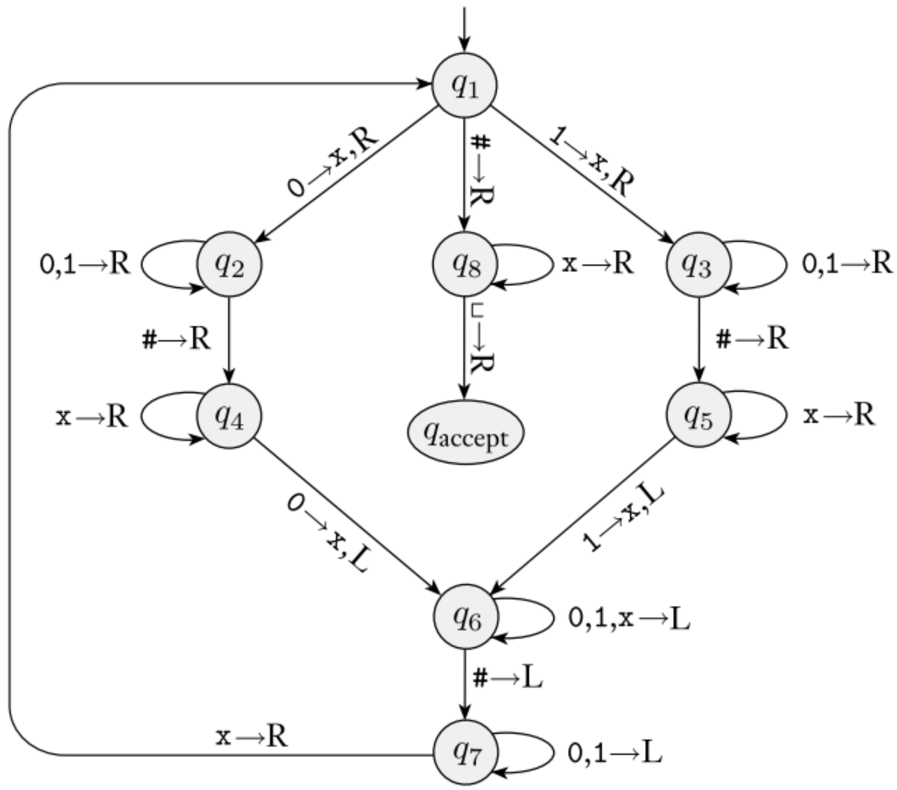
   To see what we mean, consider the following example

   ```
   0 1 # 0 1
   -> x 1 # 0 1
   -> x 1 # x 1
   -> x x # x 1
   -> x x # x x
   ```

2. Is this machine a decider?

   Yes, because it will halt (either accept or reject) no matter what the input is.

3. Draw the state diagram corresponding to the Turing machine.



To simplify the figure, we don't show the reject state or the transitions going to the reject state. Those transitions occur implicitly whenever a state lacks an outgoing transition for a particular symbol.

4. What is Q (the set of states)?

$$Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$$

5. What is Σ?

$$\Sigma = \{\texttt{0}, \texttt{1}, \texttt{\#}\}$$

6. What is Γ?

$$\Gamma = \Sigma \cup \{\texttt{b}, \texttt{x}\}$$

7. Given the string `01#01`, run through the Turing machine.

We begin with the initial configuration.

```
State:      q1
Tape:       0 1 # 0 1
  (Next:)   ^
Config:     q1 0 1 # 0 1
```

Here, we read in the `0` and move to q_2, replacing `0` with `x` and moving the tape to the right.

```
State:     q2
Tape:      x 1 # 0 1
  (Next:)    ^
Config:    x q2 1 # 0 1
```

At this point, we read in the `1` as well (without crossing anything out) and move the tape to the right.

```
State:     q2
Tape:      x 1 # 0 1
  (Next:)      ^
Config:    x 1 q2 # 0 1
```

For the same reason as above, we read in `#` as well, transitioning to q_4 and moving the tape to the right.

```
State:     q4
Tape:      x 1 # 0 1
  (Next:)        ^
Config:    x 1 # q4 0 1
```

Now, we read in the `0`, replacing it with a `x` and moving the tape left. We also transition to q_6.

```
State:     q6
Tape:      x 1 # x 1
  (Next:)      ^
Config:    x 1 q6 # x 1
```

We read in the `#`, moving the tape to the left and transitioning to q_7.

```
State:     q7
Tape:      x 1 # x 1
  (Next:)    ^
Config:    x q7 1 # x 1
```

We now keep looping at q_7 whenever we see a `0` or `1`. In our case, we only need to read one `1`, so we do that, while also moving the tape to the left.

```
State:     q7
Tape:      x 1 # x 1
  (Next:)  ^
Config:    q7 x 1 # x 1
```

We now read in the `x` and transition to q_1, moving the tape to the right.

```
State:      q1
Tape:       x 1 # x 1
  (Next:)     ^
Config:     x q1 1 # x 1
```

Now, we transition to q_3, reading in the `1` and replacing it with an `x` while also moving the tape to the right.

```
State:      q3
Tape:       x x # x 1
  (Next:)       ^
Config:     x x q3 # x 1
```

We now read in the `#`, moving the tape to the right and transitioning to q_5.

```
State:      q5
Tape:       x x # x 1
  (Next:)         ^
Config:     x x # q5 x 1
```

We now read in all of the `x`'s, moving the tape to the right. We only do this once as there is only one `x` to be read.

```
State:      q5
Tape:       x x # x 1
  (Next:)           ^
Config:     x x # x q5 1
```

We now read in a `1`, replacing it with a `x`, transitioning to q_6, and moving the tape to the left.

```
State:      q6
Tape:       x x # x x
  (Next:)         ^
Config:     x x # q6 x x
```

At q_6, we keep reading in the `x`'s. We only do this once, so we move the tape one to the left.

```
State:      q6
Tape:       x x # x x
  (Next:)       ^
Config:     x x q6 # x x
```

Now, we transition to q_7 since the next symbol is `#`, moving the tape to the left.

```
State:     q7
Tape:      x x # x x
  (Next:)    ^
Config:    x q7 x # x x
```

We transition to q_1, moving the tape to the right.

```
State:     q1
Tape:      x x # x x
  (Next:)      ^
Config:    x x q1 # x x
```

Since the next symbol to be read is a #, we transition to q_8, moving the tape to the right.

```
State:     q8
Tape:      x x # x x
  (Next:)        ^
Config:    x x # q8 x x
```

We now keep reading in any x's, moving the tape to the right. This is done twice.

```
State:     q8
Tape:      x x # x x
  (Next:)            ^
Config:    x x # x x q8
```

At this point, we are implicitly at a $\sqcup$. So, we move to q_{accept}. Thus, this string is accepted.