CHAPTER 13

Turing Machines

13.1 WHY YOU SHOULD CARE

The Turing machine is the standard model for all computation. It is capable of accepting all regular languages, context-free languages, and indeed any language computation that is believed to be possible by a computer. All programming languages are called Turing-complete, which is an informal way of saying that they can do all that a Turing machine can do. Thus understanding how Turing machines work is important for any computer scientist.

13.2 AN EXAMPLE TURING MACHINE

What makes the DFA an unsatisfactory model of computation is that it necessarily has finite memory and thus cannot count arbitrarily high, leading it to be unable to accept simple languages like

$$L = \{0^n 1^n : n \geq 0\}.$$

To get around this deficiency, the Turing machine adds an infinitely long tape (stretching forever to the left and right) that has data on it in the form of cells, each of which holds a single character and can be modified by a head that moves along the tape one cell at a time. This unbounded memory allows us to model all known computation. The input string now starts on the tape with the head pointed to the leftmost character. All other cells are filled with a special blank symbol (usually B).

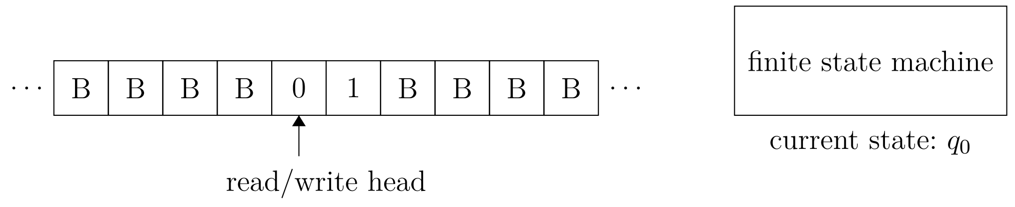
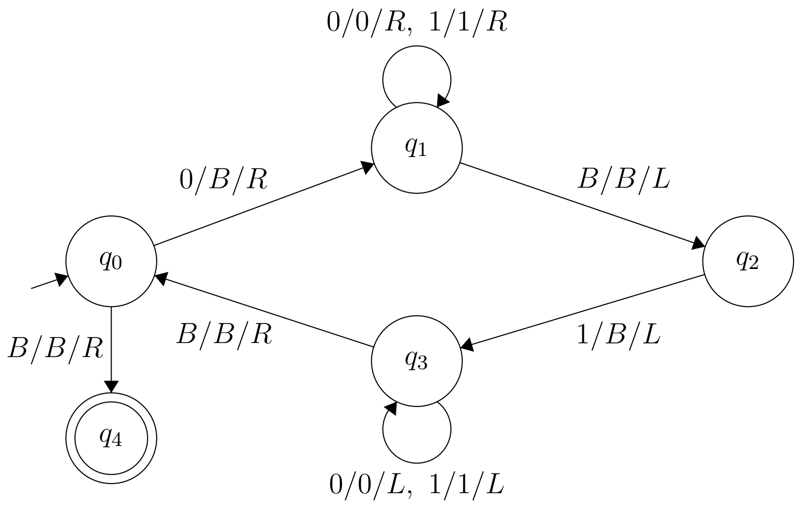


Figure 13.1: A Turing machine with the input 01 on its doubly infinite tape

Example 13.1

Turing machine for $\{0^n 1^n : n \geq 0\}$:



As with a DFA, we start at a start state (q_0 in this example) and transition between states. The difference now is that we transition based on the state and the character in the cell at the head position. Moreover, we have to specify what gets written to the cell and whether the head should move to the left (denoted by L) or the right (denoted by R). For example, notice that the state q_0 has an arrow out of it with 0/B/R to q_1. This means that if the Turing machine is at state q_0 and it sees a 0 on the tape at its head position, it will write a B at that position, move the head to the right, and switch to state q_1. Equipped with this understanding, we can now follow the algorithm being executed by the Turing machine above. The machine will erase (replace with a B) the first 0 and move to state q_1 which will traverse the rest of the string. When the end of the string is reached (indicated by a B to the right), we switch to state q_2. From q_2, the rightmost 1 is erased and we move to q_3 which moves the head back to the beginning of the string. Continuing in this manner, we can erase 0s and 1s from the two ends of the string. If the string is really in the language, then we will eventually erase the entire string at which point we follow the transition from q_0 to q_4 on a blank and accept.

Here are several differences from DFAs that are important to understand:

- Not all state/symbol combinations need to be defined for a Turing machine. If there is no rule for a given combination during the computation (for example, seeing a 1 on the tape at q_0), then the machine simply halts and rejects.
- The transitions that are defined are still deterministic—we don't allow a state/symbol combination to appear multiple times in this model of the Turing

machine. There are non-deterministic models of Turing machines, but we will not study them in this book.

- Unlike DFAs for which we go through the string only once, for a Turing machine we can traverse the string multiple times and go back to characters we have processed previously. In fact, most interesting problems will require us to make several passes over the input or use the blank tape cells in their computation.
- A Turing machine accepts a string simply by reaching an accepting state. As soon as an accepting state is reached, the computation halts and accepts.

In particular, the last item means that we do not have to "consume" the entire string to accept/reject it. For example, a Turing machine for Σ^* can have a single accepting state (and no others) or a Turing machine for $\emptyset$ can be one with a single starting state that is not accepting (and no others).

We will next see some notation for tracing the steps of a Turing machine. The trace will contain all the characters on the tape (other than the blank symbols that go off infinitely to the left and right) with the current state written immediately to the left of the position of the head.

Example 13.2

For example, the trace for the earlier Turing machine on the input 0011 is given below:

$$q_0 0011 \Rightarrow q_1 011 \Rightarrow 0q_1 11 \Rightarrow 01q_1 1 \Rightarrow 011q_1 B \Rightarrow 01q_2 1 \Rightarrow 0q_3 1 \Rightarrow q_3 01 \Rightarrow$$
$$q_3 B01 \Rightarrow q_0 01 \Rightarrow q_1 1 \Rightarrow 1q_1 B \Rightarrow q_2 1 \Rightarrow q_3 B \Rightarrow q_0 B \Rightarrow q_4 B \text{ (ACCEPT)}$$

Example 13.3

The trace for the earlier Turing machine on the input 000111 starts out as below:

$$q_0 000111 \Rightarrow q_1 00111 \Rightarrow 0q_1 0111 \Rightarrow 00q_1 111 \Rightarrow 001q_1 11 \Rightarrow 0011q_1 1 \Rightarrow 00111q_1 B$$
$$\Rightarrow 0011q_2 1 \Rightarrow 001q_3 1 \Rightarrow 00q_3 11 \Rightarrow 0q_3 011 \Rightarrow q_3 0011 \Rightarrow q_3 B0011 \Rightarrow q_0 0011 \Rightarrow \ldots$$

At this point, the state of the Turing machine is identical to that in the starting point in the previous example and the computation goes on to accept.

Example 13.4

On the other hand, the string 011 will not be accepted:

$$q_0 011 \Rightarrow q_1 11 \Rightarrow 1q_1 1 \Rightarrow 11q_1 B \Rightarrow 1q_2 1 \Rightarrow q_3 1 \Rightarrow q_3 B1 \Rightarrow q_0 1 \text{ (REJECT)}$$

The language of a Turing machine is the set of strings accepted by it. Notice that this first example Turing machine will halt and accept or halt and reject on all strings, but it is possible that a Turing machine can reject by running forever. This detail will be important later.

13.3 FORMAL DEFINITION OF A TURING MACHINE

A Turing machine can be defined using a 7-tuple as follows.

Definition 13.1

Any Turing machine can be defined by a tuple of the form $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where

- Q is the set of states,
- Σ is the input alphabet,
- Γ (upper case gamma) is the tape alphabet (note: $\Sigma \subseteq \Gamma$ and $B \in \Gamma$),
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function—this takes as inputs the current state and the tape alphabet the head is pointing to and outputs the next state, what to write at the head position, and whether to move left (L) or right (R),
- q_0 is the start state,
- B is the special blank symbol, and
- F is the set of accepting states.

Example 13.5

For the Turing machine in Example 13.1, the tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is given by

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{0, 1, B\}$,
- δ is the transition function from the figure ($\delta(q_0, 0) = (q_1, B, R)$, $\delta(q_1, 0) = (q_1, 0, R), \ldots$),
- q_0 is the starting state,
- B is the blank symbol, and
- $F = \{q_4\}$.

13.4 RECOGNIZING ADDITION

Our second example will demonstrate how Turing machines can recognize addition with unary numbers. Unary, or tally numbering, is a very simple numerical encoding that has a single symbol (usually 1 or 0) and each positive integer is represented by that number of that symbol.

Example 13.6

In unary, the number 4 is represented by 1111 and 7 is represented by 1111111.

The following language allows us to recognize or check addition in unary by using 0s for the first number, 1s for the second, and 0s for the sum:

$$L_+ = \{0^n 1^m 0^{n+m} : n, m \geq 0\}.$$

To design a Turing machine for this language, we can use an algorithm similar to the previous example. We can match 0s at the start of the string with 0s at the end by erasing them. Once this is done we erase matched pairs of 1s from the beginning with 0s on the end until the entire string is erased. If the end result of the computation is a blank tape, then the computation should accept. If the string is not of the given form, the computation should reject.

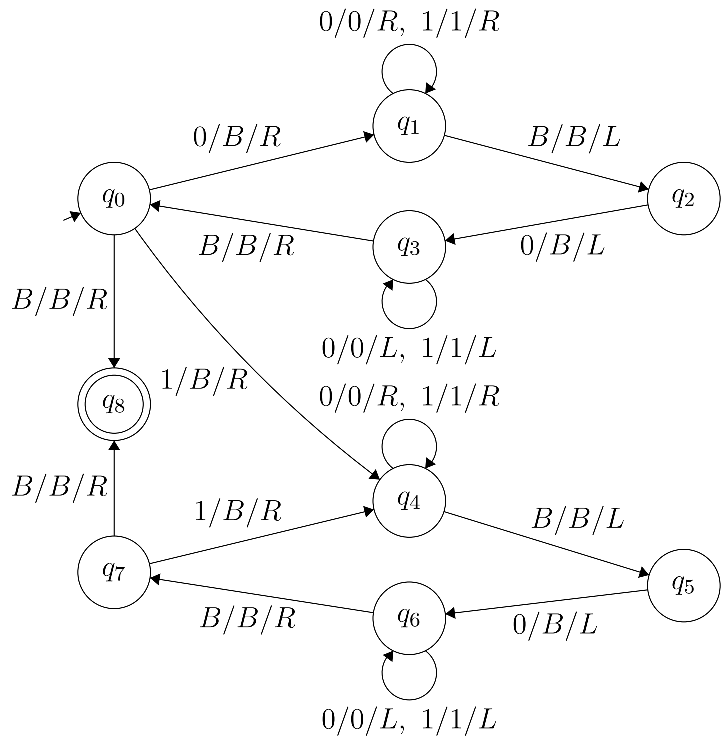


Figure 13.2: A Turing machine for $\{0^n 1^m 0^{n+m} : n, m \geq 0\}$.

Example 13.7

A Turing machine for

$$L_+ = \{0^n 1^m 0^{n+m} : n, m \geq 0\}$$

is given in Figure 13.2. This Turing machine executes the algorithm outlined earlier. The states q_0, q_1, q_2, q_3 erase the leftmost 0 and the rightmost 0 repeatedly until there are no more 0s at the start of the string. The states q_4, q_5, q_6, q_7 then in a similar fashion erase the leftmost 1 and the rightmost 0 repeatedly. If the input string is of the correct form, then the entire string will be erased and the computation will accept. In all other cases it will reject.
Note that this Turing machine correctly handles special cases such as strings of the form $0^n 1^m 0^{n+m}$, where $n = 0$, $m = 0$, and both $n = 0$ and $m = 0$. Trace through examples of these cases to see why this is true. As with coding, you want to pay attention to special cases like these and make sure your algorithm is working correctly for them.

Example 13.8

The trace for the above Turing machine on the input 0100 (demonstrating that $1 + 1 = 2$) looks like this:

$$q_00100 \Rightarrow q_1100 \Rightarrow 1q_100 \Rightarrow 10q_10 \Rightarrow 100q_1B \Rightarrow 10q_20 \Rightarrow 1q_30 \Rightarrow q_310$$
$$\Rightarrow q_3B10 \Rightarrow q_010 \Rightarrow q_40 \Rightarrow 0q_4B \Rightarrow q_50 \Rightarrow q_6B \Rightarrow q_7B \Rightarrow q_8B \text{ (ACCEPT)}$$

Example 13.9

The rejecting trace on the input 0011000 (demonstrating that $2+2 \neq 3$) looks like this:

$$q_00011000 \Rightarrow q_1011000 \Rightarrow 0q_111000 \Rightarrow 01q_11000 \Rightarrow 011q_1000 \Rightarrow 0110q_100$$
$$\Rightarrow 01100q_10 \Rightarrow 011000q_1B \Rightarrow 01100q_20 \Rightarrow 0110q_30 \Rightarrow 011q_300 \Rightarrow 01q_3100$$
$$\Rightarrow 0q_31100 \Rightarrow q_301100 \Rightarrow q_3B01100 \Rightarrow q_001100 \Rightarrow q_11100 \Rightarrow 1q_1100 \Rightarrow 11q_100$$
$$\Rightarrow 110q_10 \Rightarrow 1100q_1B \Rightarrow 110q_20 \Rightarrow 11q_30 \Rightarrow 1q_310 \Rightarrow q_3110 \Rightarrow q_3B110$$
$$\Rightarrow q_0110 \Rightarrow q_410 \Rightarrow 1q_40 \Rightarrow 10q_4B \Rightarrow 1q_50 \Rightarrow q_61 \Rightarrow q_6B1 \Rightarrow q_71 \Rightarrow q_4B$$
$$\Rightarrow q_5B \text{ (REJECT)}$$

You should also verify that the Turing machine works correctly when $n = 0$ (for example, by accepting strings such as 10 and 1100 and rejecting strings such as 110 and 100), when $m = 0$ (for example, by accepting strings such as 00 and 0000 and rejecting strings such as 0 and 000), and when both n and m are zero (i.e., when the input is λ).

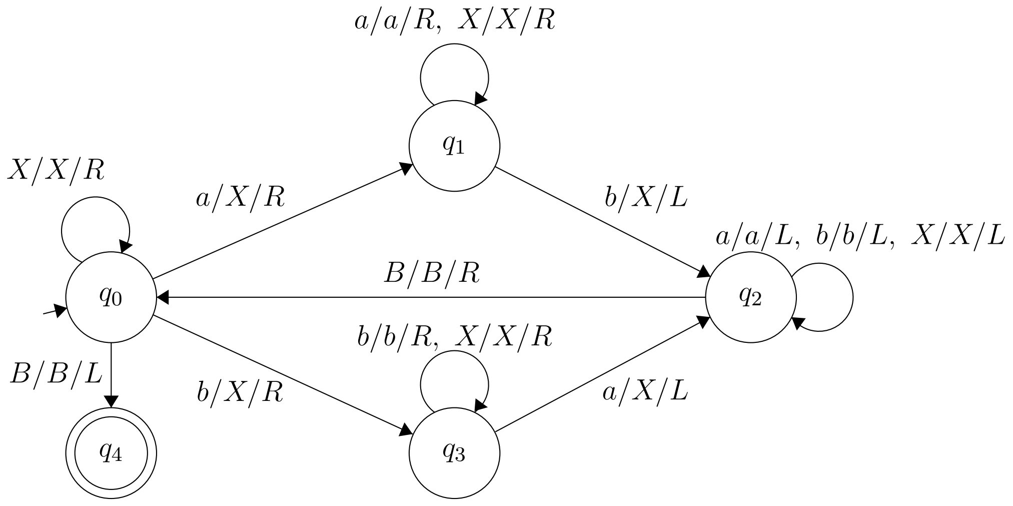
13.5 CONDITIONAL BRANCHING WITH A TURING MACHINE

We will next see an example of a Turing machine for a more complex language, the set of strings with equal numbers of as and bs. This example will illustrate the idea of branching with a Turing machine. The algorithm this time will mark off the leftmost a or b with a new symbol and then move to the right through the string to find a matching complementary symbol (b or a, respectively). Once this has been marked off, the Turing machine returns to the start of the string and repeats this procedure again. The computation terminates successfully when all the as and bs have been marked off. If this does not happen, then the computation rejects.

Example 13.10

Here is a Turing machine for the language

$$L_= = \{w \in \{a, b\}^* : w \text{ has an equal number of a's and b's}\}.$$



This example introduces a new symbol, X, that is not in the input alphabet but is in the tape alphabet. The Turing machine works by branching depending on whether the first character is an a or a b. If it is an a, it replaces it with an X and takes the upper branch (q_1) to move to the right until it finds a b that can be replaced with an X. If it is a b, it also replaces this with an X and takes the lower branch (q_3) to move to the right until it finds an a that can be replaced with an X. In both cases, the state q_2 will then move the head to the left until it finds the start of the string and repeats the procedure again (from q_0) for the next a or b that has not been replaced with an X. If all the characters in the string have been replaced by Xs, the Turing machine goes to the accepting state. Again, note that the special case of the empty string is handled correctly as we accept when we see a B on q_0.

13.6 TURING MACHINES CAN ACCEPT ALL REGULAR LANGUAGES

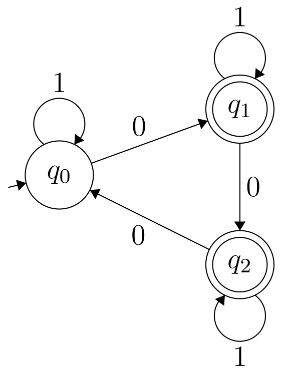
We will show that every regular language has a Turing machine that accepts it by simulating the actions of the DFA with its finite state. We'll start with an example and then generalize this idea to any regular language.

Example 13.11

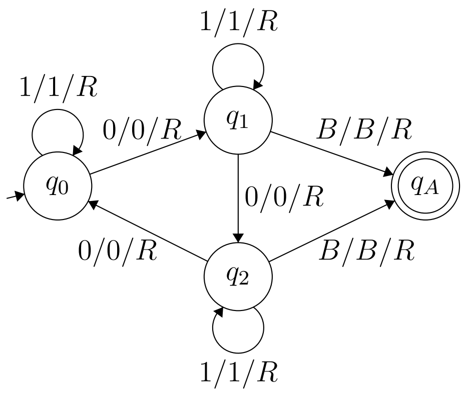
The regular language

$$\{s \in \{0,1\}^* : \text{ the number of 0s in } s \text{ is not divisible by 3}\}$$

is accepted by the following DFA:



We can convert this DFA into a Turing machine with the same language:



Notice that (1) we changed each DFA transition into a Turing machine transition with the same states and symbol, leaving the tape unchanged and always moving right, (2) we made the accepting states of the DFA into non-accepting ones (as this might lead to a premature acceptance from the Turing machine), and (3) we added a new accepting state q_A that we transition to from each accepting state (q_1 and q_2 in this example) on a B since we should accept precisely the strings that end at an accepting state and a B indicates the end of the string.

Theorem 13.1

Every regular language has a Turing machine that accepts it.

Proof

Let L be any regular language. By the definition of regular languages, L must have a DFA for it $D = (Q, \Sigma, \delta, q_0, F)$. We will generate a Turing machine that accepts the same language as D.
Define the Turing machine $T = (Q \cup \{q_A\}, \Sigma, \Sigma \cup \{B\}, \delta', q_0, B, \{q_A\})$ as follows:

- For each transition $\delta(q_s, c) = q_d$ ($q_s, q_d \in Q, c \in \Sigma$) for the DFA, add the transition $\delta'(q_s, c) = (q_d, c, R)$ for the Turing machine.
- For each accepting state of the DFA $q \in F$, add the transition $\delta'(q, B) = (q_A, B, R)$ for the Turing machine.

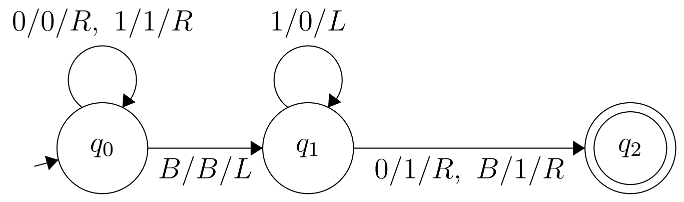
The behavior of the constructed Turing machine mimics that of the DFA precisely since all the head moves are to the right, iterating through the input string starting from the start state. Moreover, once the end of the string is reached (signaled with a B symbol), the Turing machine has a transition to its accepting state precisely when the string would have been accepted by the DFA (i.e., from one of the accepting states of the DFA). Thus, the Turing machine accepts the language of the DFA. This means that the regular language L also has a Turing machine that accepts it.

13.7 TURING MACHINES AS COMPUTERS OF FUNCTIONS

Besides computing languages, Turing machines can also compute functions. Specifically, such a the Turing machine starts with its input on the tape and writes the output of the function before it halts.

Example 13.12

Turing machine to increment a binary counter:



This Turing machine works by moving to the rightmost character of the string and then replacing 1s with 0s until it encounters a 0 that it changes to a 1. A special case is if it encounters no 0s, then it carries a 1 to the next blank cell to the left.

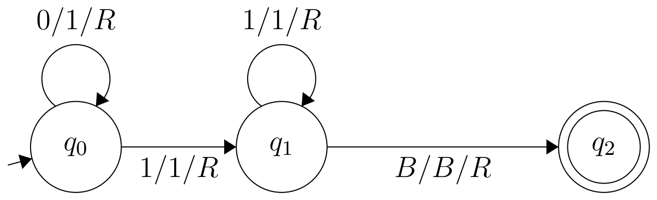
Example 13.13

For example, the previous Turing machine will increment from 1000 to 1001, from 1011 to 1100, and from 1111 to 10000.

The next simple example shows how to add two unary numbers (as defined in Section 13.4), represented by 0^n and 1^m. To compute the sum for 0^n1^m we need to create the output 1^{n+m}. This is very simple to do by converting all the initial 0s to 1s as in the following example:

Example 13.14

Turing machine to add two unary numbers:



Example 13.15

For example, the previous Turing machine will convert the input 00111 into 11111, showing that it can add $2 + 3$ to get 5.

13.8 CHAPTER SUMMARY AND KEY CONCEPTS

- The **Turing machine** is the standard model for all possible computation. Any computational system that can do all that a Turing machine does is called **Turing-complete**.
- A Turing machine has a **doubly-infinite tape** with a read/write **tape head** that can move along the tape. It also has **finite control** in the form of states with transitions.
- A Turing machine is **formally defined** by a tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where Q is the set of **states**, Σ is the **input alphabet**, Γ is the **tape alphabet**, δ is the **transition function**, q_0 is the **start state**, B is the **blank symbol**, and F is the set of **accepting states**.
- A Turing machine can be used for computational operations such as **recognizing arithmetic operations** and performing **conditional branching**.
- Every regular language has a Turing machine that accepts it.
- Turing machines can also be used to **compute functions** with string inputs and outputs.