PART TWO

COMPUTABILITY THEORY

3

THE CHURCH–TURING THESIS

So far in our development of the theory of computation, we have presented several models of computing devices. Finite automata are good models for devices that have a small amount of memory. Pushdown automata are good models for devices that have an unlimited memory that is usable only in the last in, first out manner of a stack. We have shown that some very simple tasks are beyond the capabilities of these models. Hence they are too restricted to serve as models of general purpose computers.

3.1 TURING MACHINES

We turn now to a much more powerful model, first proposed by Alan Turing in 1936, called the ***Turing machine***. Similar to a finite automaton but with an unlimited and unrestricted memory, a Turing machine is a much more accurate model of a general purpose computer. A Turing machine can do everything that a real computer can do. Nonetheless, even a Turing machine cannot solve certain problems. In a very real sense, these problems are beyond the theoretical limits of computation.

The Turing machine model uses an infinite tape as its unlimited memory. It has a tape head that can read and write symbols and move around on the tape.

Initially the tape contains only the input string and is blank everywhere else. If the machine needs to store information, it may write this information on the tape. To read the information that it has written, the machine can move its head back over it. The machine continues computing until it decides to produce an output. The outputs *accept* and *reject* are obtained by entering designated accepting and rejecting states. If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

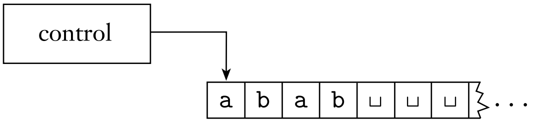

FIGURE 3.1
Schematic of a Turing machine

The following list summarizes the differences between finite automata and Turing machines.

1. A Turing machine can both write on the tape and read from it.
2. The read–write 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.

Let's introduce a Turing machine M_1 for testing membership in the language $B = \{w\#w|\ w \in \{\texttt{0},\texttt{1}\}^*\}$. We want M_1 to accept if its input is a member of B and to reject otherwise. To understand M_1 better, put yourself in its place by imagining that you are standing on a mile-long input consisting of millions of characters. Your goal is to determine whether the input is a member of B—that is, whether the input comprises two identical strings separated by a # symbol. The input is too long for you to remember it all, but you are allowed to move back and forth over the input and make marks on it. The obvious strategy is to zig-zag to the corresponding places on the two sides of the # and determine whether they match. Place marks on the tape to keep track of which places correspond.

We design M_1 to work in that way. It makes multiple passes over the input string with the read–write head. On each pass it matches one of the characters on each side of the # symbol. To keep track of which symbols have been checked already, M_1 crosses off each symbol as it is examined. If it crosses off all the symbols, that means that everything matched successfully, and M_1 goes into an accept state. If it discovers a mismatch, it enters a reject state. In summary, M_1's algorithm is as follows.

M_1 = "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, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
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*."

The following figure contains several nonconsecutive snapshots of M_1's tape after it is started on input `011000#011000`.

```
0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x x 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x x x x x x # x x x x x x ␣ ...
                          accept
```

FIGURE 3.2
Snapshots of Turing machine M_1 computing on input `011000#011000`

This description of Turing machine M_1 sketches the way it functions but does not give all its details. We can describe Turing machines in complete detail by giving formal descriptions analogous to those introduced for finite and pushdown automata. The formal descriptions specify each of the parts of the formal definition of the Turing machine model to be presented shortly. In actuality, we almost never give formal descriptions of Turing machines because they tend to be very big.

FORMAL DEFINITION OF A TURING MACHINE

The heart of the definition of a Turing machine is the transition function δ because it tells us how the machine gets from one step to the next. For a Turing machine, δ takes the form: $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$. That is, when the machine is in a certain state q and the head is over a tape square containing a symbol a, and if $\delta(q, a) = (r, b, \mathrm{L})$, 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 a move to the left.

DEFINITION 3.3

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. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ computes as follows. Initially, M receives its input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ on the leftmost n squares of the tape, and the rest of the tape is blank (i.e., filled with blank symbols). The head starts on the leftmost square of the tape. Note that Σ does not contain the blank symbol, so the first blank appearing on the tape marks the end of the input. Once M has started, the computation proceeds according to the rules described by the transition function. If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L. The computation continues until it enters either the accept or reject states, at which point it halts. If neither occurs, M goes on forever.

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. Configurations often are represented in a special way. For a state q and two strings u and v over the tape alphabet Γ, we write $u\,q\,v$ 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. The tape contains only blanks following the last 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 3.4 depicts a Turing machine with that configuration.