CSCI3390-Lecture 2:Turing Machines for Decision Problems; Decidability

September 6, 2018

1 Summary

- A variant of the Turing Machine model replaces the single halt state by two halted states, one accepting, and one rejecting. Such machines represent algorithms for decision problems.
- A language $L \subseteq \Sigma^*$ is called *decidable* (also *Turing-decidable*, *recursive*) if there is a Turing machine that on every input $w \in \Sigma^*$, halts in the accepting state if $w \in L$, and halts in the rejecting state is $w \notin L$.
- There is a weaker property, a kind of 'halfway decidable' behavior: What if the machine always gives the right answer (halts in the accepting state) if $w \in L$ and never gives a wrong answer? Such a machine might reject w if $w \notin L$, but might also run forever. Languages for which such machines exist are called *Turing recognizable* (also *recursively enumerable*).

2 Turing Machines for Decision Problems

2.1 Version 2 of the definition

Instead of having just a single halt state, let us allow our machines to have two such states, one called 'accept' and one called 'reject'. Thus the state-transition function now has the form

$$\delta : (Q - \{\textit{accept,reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\},$$

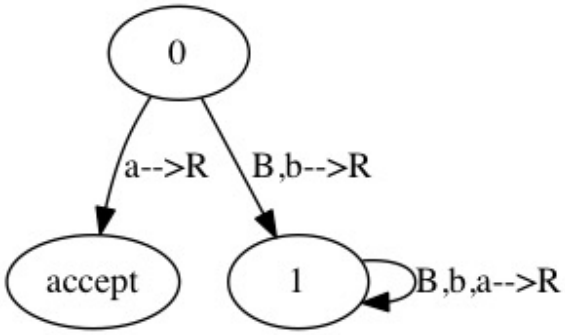

Figure 1: A Turing machine that recognizes, but does not decide the language consisting of all strings over $\{a, b\}$ whose first letter is a.

The definition is otherwise unchanged.

We say that a word w over the input alphabet Σ is *accepted* by the machine $\mathcal{M}$ if the $\mathcal{M}$, when started in the initial state with w on the tape and the current position at the first letter of w, eventually halts in the accept state. We define '*rejected* by the machine' similarly.

Very important: 'Rejected' is not the same thing as 'not accepted'! There is a third kind of behavior possible, in which the machine does not enter a halted state at all.

$L \subseteq \Sigma^*$ is *Turing-recognizable,* or *recursively enumerable,* means there is a Turing machine $\mathcal{M}$ such that for all $w \in \Sigma^*$, $\mathcal{M}$ accepts w iff $w \in L$.

$L \subseteq \Sigma^*$ is *Turing-decidable* (or *decidable*, or *recursive*) means there is a Turing machine $\mathcal{M}$ such that for all $w \in \Sigma^*$, $\mathcal{M}$ accepts w if $w \in L$ and $\mathcal{M}$ rejects w if $w \notin L$.

2.2 Examples

Look at the machine whose the state-transition diagram is in Figure 1. The input alphabet is $\{a, b\}$. The machine accepts any input whose first letter is a. If the first letter is b, it moves to the right forever. Thus this machine recognizes the language

$$L = \{av : v \in \{a, b\}^*\},$$

but it does not decide the language.

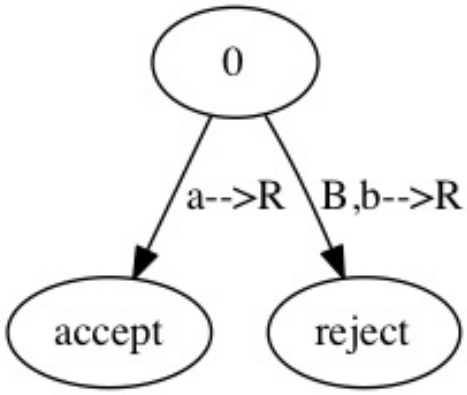

Figure 2: A Turing machine that decides the same language.

Thus L is Turing recognizable. Is it decidable? Yes (*of course* we can decide if the first letter is a) but we need a different machine. This is illustrated in Figure 2.

There is a convention we adopt for these diagrams to de-clutter them: If a transition is not specified–that is, if the diagram fails to indicate what to do for a given state-letter pair (q, γ), then the machine rejects. In other words, if no value is given for $\delta(q, \gamma)$, then we consider its value to be $(\textit{reject}, \gamma, R)$. With this convention, we can redraw the diagram as in Figure 3.

Here is a more substantial example. We describe a Turing machine that decides the language consisting of all $w \in \{a, b\}^*$ in which the number of occurrences of a is equal to the number of occurrences of b. The idea is to make multiple passes over the input, crossing out a's and b's in pairs. Each pass begins at the left-hand end of the tape. The head moves right until it finds an a or a b. If it finds an a, it crosses it out (marks the cell with an X) and enters the 'I'm looking for a b' state, and continues its rightward scan. When it finds a b, it crosses the b out, returns to the left-hand end, and repeats. The analogous procedure is carried out if it first finds a b.

What if it is looking for a b (after having crossed out an a) and doesn't find one? Then there must have been more a's than b's, so the machine rejects. The same thing happens if it is looking for an a and doesn't find one. If it is starting a fresh pass, looking for either a or b, and fails to find one, then all the letters have been crossed out in pairs, and the machine accepts. This description allows us to construct our machine: The states are 'looking for a letter' (0), 'looking for an a'

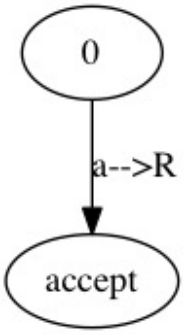

Figure 3: The same machine as Figure 2, but with the convention that transitions to the reject state do not have to be shown.

(1), 'looking for a b' (2), and 'going left to start' (3). The state-diagram is shown in Figure 4.

Complexity Analysis. Later in this course we will be concerned with the space and time complexity of algorithms. For Turing machines, this means how many tape cells the machine uses to complete a computation, and how many steps the computation requires, usually as a function of the input length. For the example above, the number of tape cells used for inputs of length n is just n—this machine does not require use any cells other than the ones on which the original input is written. [1] The time complexity is more interesting: If the input consists entirely of a's or entirely of b's, then the machine makes just a single rightward pass over the input, and completes the computation in n steps. In the worst case, if the string contains equal numbers of a's and b's, say $\frac{n}{2}$ of each, the machine makes $\frac{n}{2}$ rightward and leftward passes in crossing out all the matching pairs, and one last rightward pass. Each pass requires *at most* n steps, so the total number of steps is no more than

$$n \cdot (2 \cdot \frac{n}{2} + 1) = n^2 + n.$$

To see that this worst-case upper bound cannot be substantially improved, we can do a more detailed analysis and find a 'matching' lower bound: Suppose the input

[1] You can reasonably argue that the right answer is $n + 2$, since the machine has to look at the cells just before the first input symbol and just after the last one. We really won't worry about this distinction, since we are more interested in the simpler fact that the space required is roughly proportional to the input size.

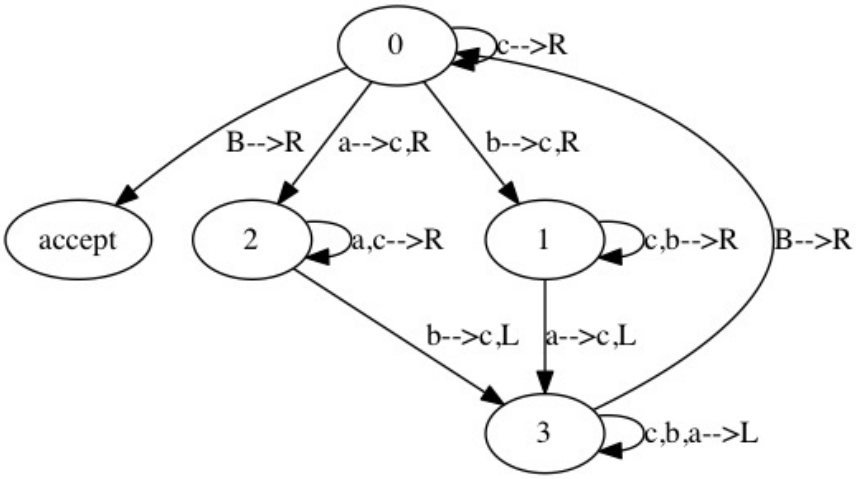

Figure 4: A Turing machine that decides the set of words over $\{a, b\}$ having an equal number of a's and b's.

string is

$$a^{n/2}b^{n/2},$$

that is, $\frac{n}{2}$ a's followed by an equal number of b's. Each rightward pass has to visit *at least* $\frac{n}{2}$ cells before returning to the start, so the total number of steps on this input is at least

$$\frac{n}{2} \cdot (2 \cdot \frac{n}{2} + 1) = \frac{n^2 + n}{2}.$$

So, basically, the worst-case running time is roughly proportional to n^2.