Lecture 13: Turing Machine

Instructor: Ketan Mulmuley
Scriber: Yuan Li

February 19, 2015

Turing machine is an abstract machine which *in principle* can simulate any computation *in nature*.

Church-Turing Thesis: any function that is computable in nature is Turing-machine-computable.

1 Definition

The most powerful machine we've seen so far is (nondeterministic) PDA which consists of a read-only tape, a head, and a stack. Compared to the laptop we are using, it lacks the capability of random access writing. As long as we have a tape which supports random-access writing, there is no reason to have two different tapes.

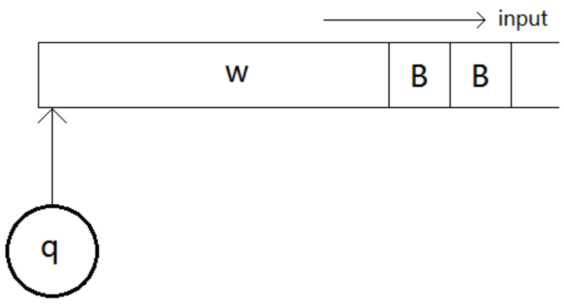

Figure 1: Turing machine with input w

Formally, Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F \subseteq Q),$$

where Q is a *finite* set of states, Σ the input alphabet, $\Gamma \supseteq \Sigma \cup \{B\}$ the tape alphabet (B is a special symbol denotes "blank"), δ the transition function, q_0 the initial state, $F \subseteq Q$ the set of final accepting states, and

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

The transition function δ describes the program. If $\delta(q, a) = (p, b, L)$, then at current state q, input symbol a, machine M will go to state p, replace a by b, and the head moves left; if $\delta(q, a) = (p, b, R)$, then at current state q, input symbol a, machine M will go to state p, replace a by b, and the head moves right; if $\delta(q, a)$ is undefined, then M halts, and the input is rejected. If the current state q is in F, then machine M accepts w. If the machine never halts, then the input is assumed to be rejected.

At the first glance, it is not clear how powerful Turing machine is. Following is Turing's argument why the machine can simulate *anything* computable by human-like computer.

Assume someone is working at a desk with a pencil and an eraser (see Figure 4). Instead of turning the pages, he can put the pages one by one (looks like a tape), and moves left or right as he wants.

He is working on *one* page at a time. Depending on the *state* of his *head*, where the number of states is finite assuming his head has finite number of neurons, and the symbols he is reading on that page, he may erase some symbols and write down some new symbols. After he is done with this page, he may move left or right continue working on a new page. We assume each page has finite number of pixels.

If two assumptions above hold (head has finite number of neurons and each page has finite number of pixels), then the person is equivalent to a Turing machine.

Therefore, we come to the conclusion that the machine can compute anything computable by human-like computer. By the way, Turing invented this machine when he was an undergraduate. There is another computation system called lambda calculus first formulated by Alonzo Church, which turns out to be equivalent to Turing machine. But I have never seen anyone thinking like lambda calculus.

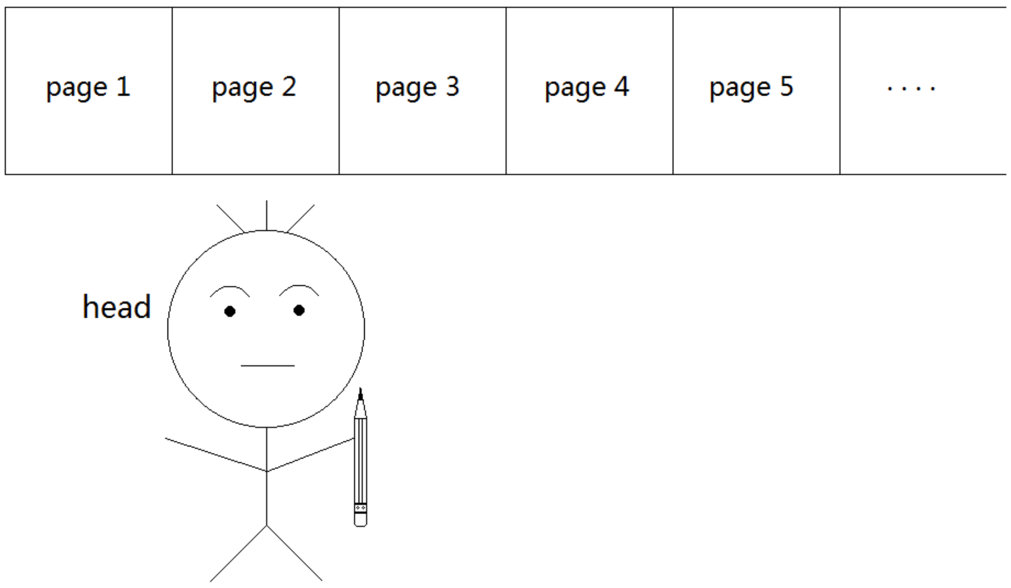

Figure 2: Someone works like a Turing machine

2 Instantaneous Description

In order to formally define the language accepted by a Turing machine, we need the concept of instantaneous description. Instantaneous description is a snapshot of Turing machine at runtime, which is $\alpha_1 q \alpha_2$, where $\alpha_1, \alpha_2 \in \Gamma^*$, $q \in Q$. String α_1 is the tape content to the left of the head, q is the state, α_2 is the tape content to the right of the head (the tape head is on the first symbol of α_2).

Define $I_1 \mapsto_M I_2$ if I_2 is the next ID of I_1 of Turing machine M. Formally, let

$$I_1 = x_1 x_2 \dots x_{i-1} q x_i x_{i+1} \dots x_n$$

If $\delta(q, x_i) = (p, y, R)$, then

$$I_1 \mapsto_M x_1 x_2 \dots x_{i-1} y p x_{i+1} \dots x_n$$

If $\delta(q, x_i) = (p, y, L)$, then

$$I_1 \mapsto_M x_1 x_2 \dots x_{i-2} p x_{i-1} y x_{i+1} \dots x_n$$

when $i > 1$; the machine M halts if $i = 1$.

Let $\mapsto_M^*$ be the reflexive transitive closure of $\mapsto_M$. The initial ID is $q_0 w$ (If α_1 or α_2 is empty, we do not mention it). The language accepted by M is

$$L(M) = \{w : q_0 w \mapsto_M^* \alpha_1 p \alpha_2 \text{ for some } p \in F, \alpha_1, \alpha_2 \in \Gamma^*\}.$$

3 Examples

Let $L = \{0^n 1^n : n \geq 1\}$, which is a CFL. We will design a Turing machine M accepting L. Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F \subseteq Q),$$

where $Q = \{q_0, q_1, \ldots, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, X, Y, B\}$, $F = \{q_4\}$, and δ is shown in the following figure.

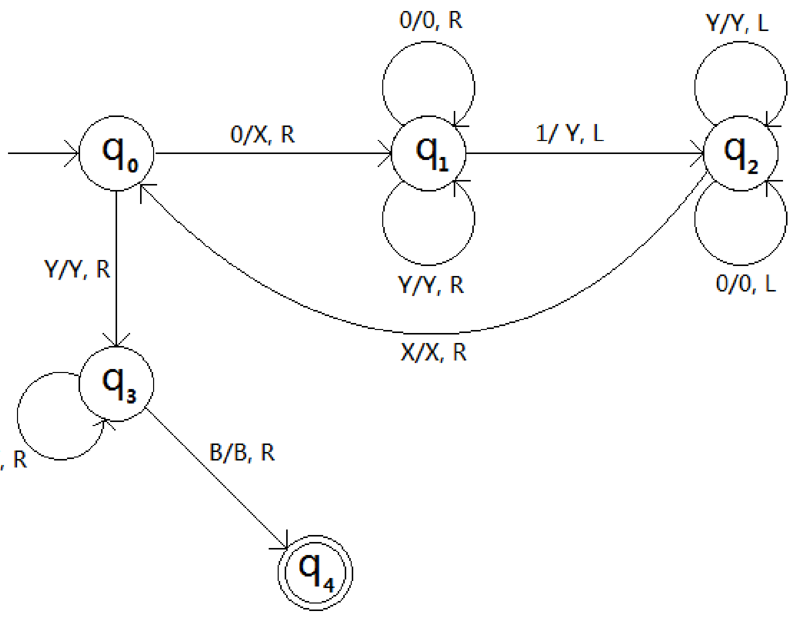

Figure 3: Turing machine accepts $\{0^n 1^n : n \geq 1\}$

Turing machine M works as follows: it marks the first 0 as X, then goes to the first 1 and marks it as Y; it goes left to the second 0, and marks it as X, then goes to the second 1 and marks it as Y, etc. Until all the 0's and 1's are marked, it moves to the right most symbol and then accepts. All undefined transitions means stuck, and the input is rejected.

	0	1	X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, R)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	—	—	—	—	—

Designing a Turing machine accepting certain language is like programming. Turing machine is the *minimal* programming language.

Let us take another example which is not a CFL. Let $L = \{0^n 1^n 2^n : n \geq 1\}$. The Turing machine works similar with the above it: it marks the first 0 as X, the first 1 as Y, the first 2 as Z; then it goes back to the second 0, and marks it as X, the second 2 as Y, the second 2 as Z, and so on.

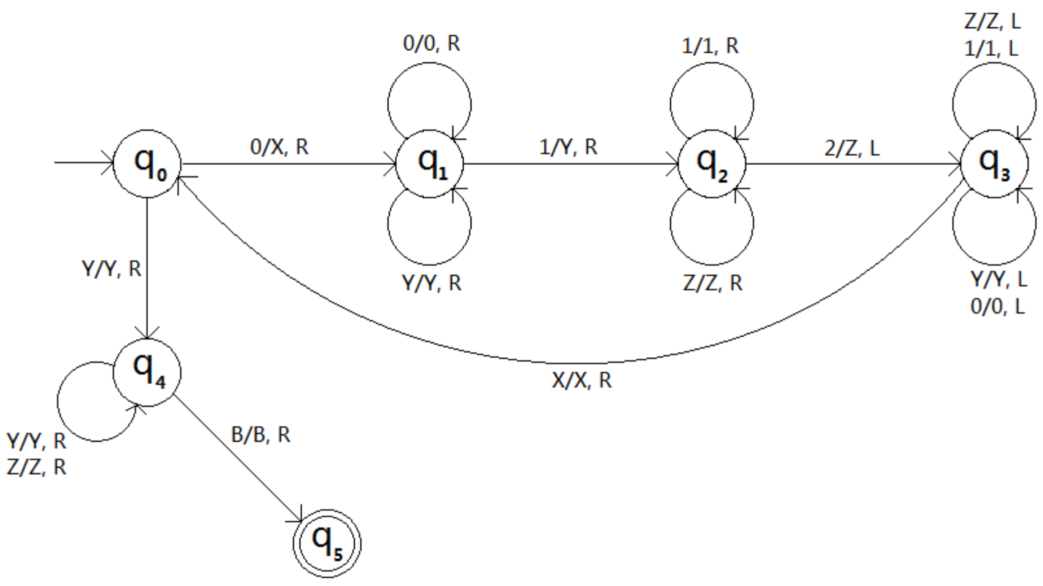

Figure 4: Turing machine accepts $\{0^n 1^n 2^n : n \geq 1\}$