CS340: Theory of Computation Sem I 2017-18

Lecture Notes 14: Turing Machines

Raghunath Tewari IIT Kanpur

So far we have seen the following:

Finite Automaton: Finite control (set of states). Input is read in one direction. No memory.

Pushdown Automaton: Finite control (set of states). Input is read in one direction. Memory is restricted to be in the form of a stack (however the stack length is unbounded).

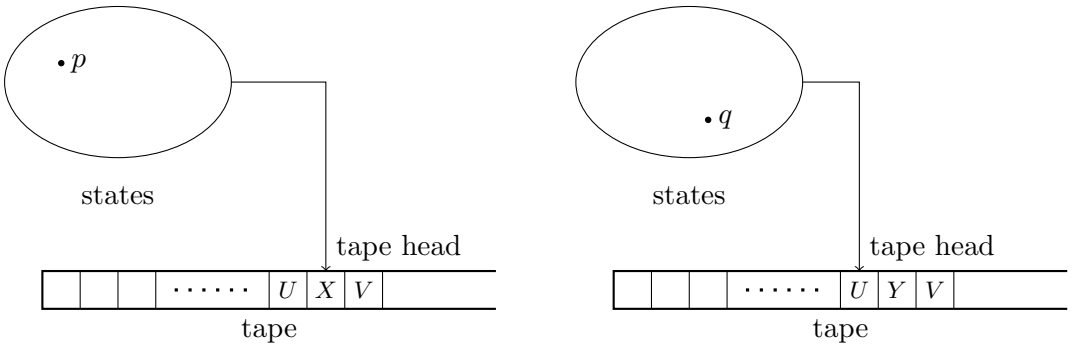
1 Turing Machine

1.1 The model

- Introduced by Alan Turing in 1936.
- Theoretical model. It can model the computation of any computing device.
- Finite control. Infinite memory in the form of a tape (all cells are accessible). Input can be read in both directions.
- Has a designated accept state and reject state. The computation halts when the Turing machine enters one of these two states.
- A transition rule of a Turing machine has the following form

$$\delta(p, X) = (q, Y, L).$$

This means that from state p, on reading the symbol X on the tape, the machine moves to state q, replaces X with Y and moves the tape head to the left. This is illustrated in the diagram below.



Before transition After transition

Therefore, the transition function has the following form

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

- The input to a Turing machine is written in the tape itself. We can do this since the tape can be read in both directions. We assume that the input alphabet Σ is a subset of the tape alphabet Γ.
- Also additionally we assume that there is a blank symbol, $\sqcup \in \Gamma$ and $\sqcup \notin \Sigma$.
- If the tape head ever tries to move left from the leftmost cell then it stays at its current position.

Formalizing, we get the following.

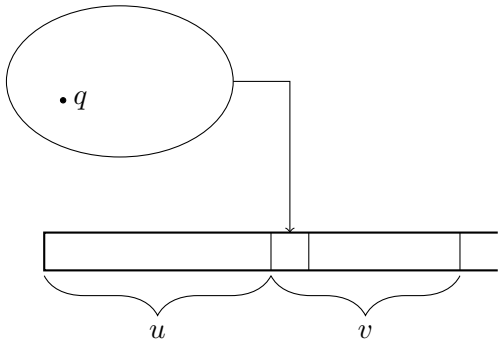
Definition 1.1. A Turing machine (denoted as TM in short) is the tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$, where

- Q is the set of states,
- Σ is the input alphabet,
- Γ is the tape alphabet, such that $\Sigma \subseteq \Gamma$ and there exists a symbol $\sqcup \in \Gamma \setminus \Sigma$,
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
- q_0 is the start state,
- q_A is the accept state,
- q_R is the reject state, where $q_A \neq q_R$.

1.2 Configuration of a TM

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ be a TM.

- A *configuration* of the TM M is a snapshot of the TM at a given instant. It consists of
  - i) the current state,
  - ii) the current tape contents, and
  - iii) the current position of the tape head.
- We denote a configuration as uqv, where $u, v \in \Gamma^*$ and $q \in Q$, if
  - i) q is the current state,
  - ii) uv is the current tape contents, and
  - iii) the tape head points to the first symbol in v (as shown below).



The configuration uqv.

- A configuration C_1 *yields* a configuration C_2 in one step (denoted as $C_1 \Rightarrow C_2$), if M can go from C_1 to C_2 in one transition step.

- Some standard configurations

  Start configuration: q_0w, where w is the input to the Turing machine. That is the head is pointing to the leftmost tape symbol.

  Accept configuration: Any configuration of the form uq_Av, where $u, v \in \Gamma^*$.

  Reject configuration: Any configuration of the form uq_Rv, where $u, v \in \Gamma^*$.

- M *accepts* a string w, if $\exists$ a sequence of configurations $C_0, C_1, \ldots, C_k$, such that,

  i) C_0 is the start configuration,

  ii) for $1 \leq i \leq k$, $C_{i-1} \Rightarrow C_i$, and

  iii) C_k is an accept configuration.

- M *rejects* a string w, if $\exists$ a sequence of configurations $C_0, C_1, \ldots, C_k$, such that,

  i) C_0 is the start configuration,

  ii) for $1 \leq i \leq k$, $C_{i-1} \Rightarrow C_i$, and

  iii) C_k is a reject configuration.

- Observe that there can be strings on which M neither accepts nor rejects (basically goes into an infinite loop).

- M is said to be a *halting* TM, if $\forall w \in \Sigma^*$, M either accepts w or rejects w.

- $L(M) = \{w \mid M \text{ accepts } w\}$.

- A language L is said to be *Turing-recognizable* if there exists a TM M such that $L = L(M)$.

- A language L is said to be *Turing-decidable* (or simply *decidable*) if there exists a halting TM M such that $L = L(M)$.

Remark. Turing-recognizable languages are also called *recursively enumerable* languages and Turing-decidable languages are also called *recursive* languages.

1.3 An Example

Consider the non-CFL

$$L = \{a^n b^n c^n \mid n \geq 0\}.$$

We give the intuition behind the construction of a TM for L.

1. First make a pass through the input and check whether it is of the form $a^*b^*c^*$ or not. If not then enter reject state else return to the leftmost cell.

2. Do the following until all occurrences of one of the three symbols are completely crossed out.

   (i) Cross the leftmost uncrossed a.

   (ii) Move to the leftmost uncrossed b and cross it.

(iii) Move to the leftmost uncrossed c and cross it.

(iv) Move back to the leftmost uncrossed a.

3. Make one more pass through the tape. If all symbols are crossed out then accept else reject.

Exercise 1. Give the formal definition of a TM for the above language.

Note 1. As you can see even for a simple language, designing a TM can be a cumbersome experience. We will usually give a high level description for a decidable or Turing-recognizable language. The important point is that *anything that can be computed can be computed using a TM.*

1.4 Variants of the TM

Let us consider some variants of the TM and argue about their computational power.

1. **Multitape TM:** The TM is allowed to have k tapes. It has a separate head in each of the k tapes. In each transition step, each tape replaces the bit under its respective head and moves left/right.

   We can simulate all the k tapes using a single tape. The contents of the different tapes are written side by side, separated by some marker symbol. Two issues arise in this case.

   - What if the contents of a tape increases?

     For each additional symbol added to a tape we shift the contents on the right side, to the right by one cell.
   - How do we keep track of the head positions for each individual tape?

     For each tape symbol, say a, in the original TM we add another symbol $\overline{a}$. This new symbol is used to keep track of the head position in each tape's portion.

2. **Two stacks:** The two stacks store the contents of the tape on the left side and right side of the tape head respectively.

3. **Queue:** Can be simulated using two stacks.

Exercise 2. Read Chapters 3.1, 3.2.

2 Nondeterministic Turing Machine

Given a TM M and an input w to M, from any configuration C, M can only move to a unique configuration C'. However we can also define a nondeterministic variant in which the machine can move to multiple configurations simultaneously.

- **Transition function**: A nondeterministic Turing machine is defined in the same way as a deterministic TM, except for the transition function. The transition function of a nondeterministic Turing machine is defined as

  $$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L,R\}}.$$

  In other words, from a single *(state, symbol)* pair the machine can have multiple transitions.

- **Configuration**: Given a nondeterministic TM M and an input w to M, from any configuration C, M can move to multiple configurations in one step.

- **Acceptance criterion**: The machine accepts if *some* sequence of transitions leads to an accept state.

2.1 Examples

2.1.1 Checking Non-Primality

Consider the language

$$L_C = \{0^p \mid p \text{ is a composite}\}.$$

We will design a nondeterministic TM for L_C as follows.

Input: A string w

1. Nondeterministically write l_1 a's and l_2 b's on the tape, where l_1 and l_2 are two numbers such that $2 \leq l_1, l_2 < p$.
2. Repeat the following until all a's are crossed off.
   i. Alternately cross off all b's and equal number of 0's in the string w.
   ii. If no 0's remain then reject.
   iii. Uncross all b's.
   iv. Cross off one a.
3. Accept if all 0's in w are crossed off, else reject.

2.1.2 Subset Sum

Given a set of integers S and a number k, is there a subset $S' \subseteq S$ such that the sum of the elements in S' is k?
We can formally state the above problem in a language form.

$$L_{SS} = \{\langle S, k\rangle \mid \exists S' \subseteq S \text{ and } \sum_{a \in S'} a = k\}$$

where $\langle S, k\rangle$ is an encoding of the set S and the number k.
The algorithm is simple. We nondeterministically guess a subset S' of S and compute its sum. If it equals k then accept else reject.

2.2 Configuration Graph

The concept of configuration graph is very important to the study of languages.

Definition 2.1. Given a Turing machine M (may be deterministic or nondeterministic) and an input x to M, the *configuration graph* of M on x, denoted as $G_{M,x}$ is defined as follows:

- the vertices of $G_{M,x}$ are the configurations of the machine M on input x, and

- there is an edge from a configuration C_1 to a configuration C_2 if there is a one step transition of M from C_1 to C_2, on the input x.

A *computation path* is a path in $G_{M,x}$ starting from the start configuration.

From the definition we can immediately observe the following Proposition.

Proposition 1. *M accepts x if and only if there is a path from the start configuration to an accept configuration in $G_{M,x}$.*

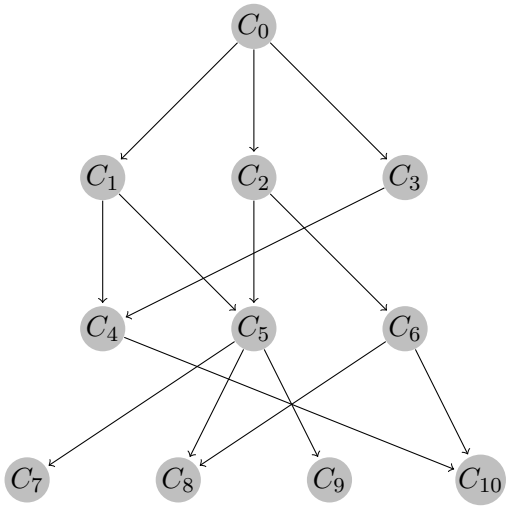
We will now discuss some properties regarding the configuration graph. Let M be a TM and x be an input to M.

- The definition of configuration graph only makes sense if both a machine and an input is provided.
- If M is deterministic then the outdegree of $G_{M,x}$ is at most one. On the other hand, if M is nondeterministic then the outdegree can be arbitrary.
- A vertex in $G_{M,x}$ has outdegree zero if and only if it corresponds to an accept or reject configuration.
- Technically the configuration graph can be infinite. This is because a configuration depends on the tape contents which can be unbounded. However later when we move to complexity theory, we will be able to bound the space used on the tape, as a function of the input size. As a result, the configuration graph will also be finite in such cases.
- Although by definition $G_{M,x}$ contains all possible configurations, but we will usually be interested in only those configurations in $G_{M,x}$ that are reachable from the start configuration.

2.2.1 Examples

Below we give examples of configuration graphs of deterministic and nondeterministic TMs, showing only the vertices that are reachable from the start configuration (start configuration is C_0).

Configuration graph of a deterministic machine.



Configuration graph of a nondeterministic machine.

2.3 Equivalence of Deterministic and Nondeterministic TMs

Theorem 2. *The class of languages accepted by deterministic TMs is the same as the class of languages accepted by nondeterministic TMs.*

One direction is trivial since deterministic TMs are also nondeterministic TMs.
To see the other direction, consider a NDTM N. We will give the sketch of the design of a deterministic TM M, such that $L(M) = L(N)$. Let x be an input. The machine M essentially does a breadth first search on the configuration graph $G_{N,x}$ by traversing it in a level wise manner.

i) M maintains a queue data structure which initially has a start configuration.

ii) In every iteration, M dequeues a configuration and enqueues all its children into the queue.

iii) If an accept configuration is encountered at any stage then them M halts and accepts.

iv) If the queue becomes empty (that is all configurations reachable from the start configuration are traversed) then M halts and rejects.

Note 2. We both BFS and DFS are graph traversal algorithms, we cannot do a DFS to traverse $G_{N,x}$. This is because even for an input which is in $L(N)$, N might have a computation path of infinite length. DFS will remain stuck on this path forever.

2.4 Definition of Algorithm

Hypothesis 3 (Church Turing Thesis)**.** *Anything that can be computed can be computed using a Turing machine.*

The Church-Turing thesis essentially says that our notion of algorithms as being a step by step procedure of solving a problem, is the same as that of Turing machine algorithms.

Exercise 3. Read Chapter 3.3.