Turing Machines

Notes on Automata and Theory of Computation

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Recursively Enumerable Languages

- The set of regular languages is a proper subset of the set of context-free languages.
- While context-free grammar appears to be able to model natural languages and programming languages, some very simple languages cannot be characterized by cfg, e.g.

$$\{a^n b^n c^n, n \geq 0\},\ \{ww, w \in \{a,b\}^*\}.$$

- We introduce the set of **recursively enumerable (RE)** languages. It includes the set of context-free languages and contains the above examples.

Automata and Languages

- RE languages are defined by **Turing machines (TM)**. That is, a language is RE if it is accepted by a Turing machine.
- To draw analogy, note that regular languages and context-free languages can equivalently be defined with automata, i.e., the finite automata and the pushdown automata.
- We begin our study beyond context-free languages and pushdown automata with Turing machines.

Turing Machine

- A TM uses a *tape* as storage.
- The tape is divided into **cells**. A cell holds one tape symbol.
- A read-write **head** is above some cell.
- In one **move**, the head reads the symbol beneath it, writes a symbol to the current cell, moves left or right, and the machine is in another state.
- Initially, the input is stored on the tape surrounded by blanks, and the head is above the first symbol of input.
- It keeps going until no moves can be made or a final state is entered.

Formal Definition

- A TM M is defined by

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

  where
  - Q is the set of states
  - Σ is the input alphabet
  - Γ is the **tape alphabet**
  - δ is the transition function
  - $\Box \in \Gamma$ is the **blank symbol**
  - q_0 is the initial state
  - F is the set of final states

Notational Conventions

- input symbol: lower-case letters at the beginning of alphabet e.g., a, b, c
- tape symbol: capital letters near the end of alphabet e.g., X, Y, Z
- string of input symbols: lower-case letters near the end of alphabet, e.g., w, x, y, z
- string of tape symbols: Greek letters, e.g., α, β, γ
- state: p, q and nearby letters

Transition Function

- Domain and range of a transition function

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

- In words, based on the current state and tape symbol, a TM does three things: transits state, writes a symbol to the current cell, and moves the head left or right.
- A TM is said to **halt** if it reaches a configuration for which δ is not defined. This is possible because δ is a partial function in general.
- It helps to look at some examples (Ex. 9.1-2) to get the ideas.

Standard Turing Machines

- There are quite a few models of Turing machines. Some of them are equivalent in their descriptive powers.
- A TM is said to be a **standard TM** if it has the following features.
  - The tape is unbounded in *both* directions.
  - It is *deterministic* in the sense that at most one move is defined in δ for any configuration.
  - There is *no* input file or output device. Everything is on the tape.
- We will be talking about standard TMs unless specified otherwise.

Instantaneous Description

- The configuration of a TM at an instant is completely specified by state, tape content, and head position.
- We can denote a configuration by

$$\alpha\, q\beta \quad \text{or} \quad X_1X_2\ldots X_{k-1}\, qX_k\ldots X_n,$$

  meaning
  - the current state is q,
  - the tape content is $X_1X_2\ldots X_n$, i.e., $\alpha\beta$,
  - the head is above the cell for X_k.
- This notation is called **instantaneous description (ID)**.

Moves

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a TM. A **move** from one ID to the next is denoted by

$$\alpha_1 \, p\alpha_2 \;\vdash\; \beta_1 \, q\beta_2.$$

- The transition function decides moves,

$$X_1 \ldots \; pX_k X_{k+1} \ldots X_n \vdash X_1 \ldots Y \; qX_{k+1} \ldots X_n$$
$$\Leftrightarrow \;\; \delta(p, X_k) = (q, Y, R),$$
$$X_1 \ldots X_{k-1} \; pX_k \ldots X_n \vdash X_1 \ldots \; qX_{k-1}Y \ldots X_n$$
$$\Leftrightarrow \;\; \delta(p, X_k) = (q, Y, L).$$

Moves at the Boundaries

- In an instantaneous description, we need not specify the blank symbols extending to the left and the right.
- However, if the head is above a blank cell, then we need to signal that in ID. In particular,
  - If $\delta(p, X_1) = (q, Y, L)$ and the head is at the left end, then

$$pX_1X_2 \ldots X_n \vdash q\Box YX_2 \ldots X_n$$

  - Similarly, if $\delta(p, X_n) = (q, Y, R)$

$$X_1X_2 \ldots X_{n-1}\, pX_n \vdash X_1X_2 \ldots X_{n-1}Y\ q\Box$$

Transition Graph

- The transition function of a TM can be represented by a table or a graph.
- In a transition graph, each state is represented by a vertex. An edge from state p to state q is labelled by one or more items of X, Y, D, where X is the scanned symbol, Y is the replacing symbol and D is the direction of move.
- An edge with multiple labels can be replaced by multiple edges, each with a single label.

Halting

- We represent a sequence of moves by $\overset{*}{\vdash}$. For example,

$$\alpha_1\ p\beta_1 \overset{*}{\vdash} \alpha_2\ q\beta_2.$$

- M is said to **halt** if it is in a configuration for which the transition function is undefined.
  - A TM can halt in a final state: we can make a TM halt whenever a final state is entered by making the transition function undefined in any final state.
  - A TM can also halt in a non-final state.

Computation

- A sequence of moves that eventually makes a TM halt is called a **computation**.
- When a TM finishes a computation, we know whether or not the input is accepted. An input is accepted if it leads the TM to a final state and halt.
- *A TM may never halt for some inputs*. In such cases, the TM is said to be in an **infinite loop**, for which we use the following notation

$$\alpha\, p\beta \overset{*}{\vdash} \infty.$$

By definition, these inputs are not accepted by the TM. Ex. 9.3 is an example for infinite loop.

Language of a TM

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a TM. The language recognized (accepted) by M is defined by

$$L(M) = \{w \in \Sigma^+ : q_0 w \overset{*}{\vdash} \alpha\, q_f \beta, q_f \in F, \alpha, \beta \in \Gamma^*\}.$$

  Note
  - The final tape content is irrelevant in the definition.
  - λ is not in $L(M)$.
- By definition, $L(M)$ is recursively enumerable for any M.

Infinite Loop

- By definition, if a string w makes a TM to be in an infinite loop, then it is not in $L(M)$.
- There are three cases when running M on w.
  - M halts in a final state. $w \in L(M)$.
  - M halts in a non-final state. $w \notin L(M)$.
  - M does not halt after a very long time. We *cannot decide* whether or not $w \in L(M)$.

  It is the last case that makes things interesting. We may not be able to decide whether M is just doing an extremely long computation or it is indeed in an infinite loop.

Algorithm

- A TM M that *halts on any inputs* is said to be an **algorithm**.
- For an input, an algorithm either halts in a final state or halts in a non-final state. The possibility of entering an infinite loop is eliminated from consideration.
- Common understanding of an algorithm is a procedure that solves a problem. Here, if M is an algorithm, it solves the problem of whether $w \in L(M)$ for any w.

Recursive Language

- Given a language L, if there exists an algorithm M (which halts on any input) such that $L = L(M)$, then L is said to be **recursive**.
- The set of recursive languages is a subset of the set of RE languages.
  - According to the above definition, a recursive language is RE.
  - Is there an RE language that is not recursive?
- We will have a more detailed discussion later.

TM for 00^*

- Let $M = \{\{q_0, q_1\}, \{0,1\}, \{0,1\}, \delta, q_0, \Box, \{q_1\}\}$, with
$$\delta(q_0, 0) = (q_0, 0, R),\ \delta(q_0, \Box) = (q_1, \Box, R).$$
- M halts without acceptance whenever 1 is read. It halts with acceptance if $\Box$ is read.

TM for $\{a^n b^n\}$ and $\{a^n b^n c^n\}$

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, x, y, \Box\}$, $F = \{q_4\}$.
- The idea is to replace a by x and b by y. q_0 signals equal number of x and y and moving right, q_1 signals an unmatched x and moving right, q_2 signals equal number of x and y and moving backwards, q_3 signals no more a cannot be found before first y.
- TM for $\{a^n b^n c^n\}$ can be constructed similarly.
- Note that one is cfl but the other is not. That is, TM can recognize some languages that cannot be recognized by npda.

TM as a Transducer

- When a TM M is used as an acceptor, we are not concerned about the tape content when a computation finishes. We only need to know the state M is in, to decide whether the input is in $L(M)$.
- A TM can be a **transducer**. In this case, we care about the tape content when a computation finishes.
- Modern digital computers act more like transducers than acceptors.
- A TM transducer M of a function $f(w)$ is such that

$$q_0 w \overset{*}{\vdash}_M q_f f(w),\ q_f \in F.$$

Computable Functions

- A function f with domain D is said to be **computable** or **Turing-computable** if there exists a TM transducer M such that
$$q_0 w \overset{*}{\vdash}_M q_f f(w),\ q_f \in F,$$
for all $w \in D$.
- All basic mathematical functions (operations) are Turing-computable, as well as composite functions of basic functions.

Addition

- A TM can be designed to compute $x + y$ for positive integers x and y.
- First, we need a representation for positive integers. We use the **unary notation**

$$w \in \{1\}^+, \ |w(x)| = x.$$

- The designed TM should carry out the following computation for any x, y

$$q_0 w(x) 0 w(y) \overset{*}{\vdash} q_f w(x + y) 0.$$

- See Example 9.9.

Copy

- A TM can be designed for the copy function. Using the unary notation the designed TM should carry out the following computation for any w

$$q_0 w \overset{*}{\vdash} q_f ww.$$

- See Example 9.10.

Test of Condition

- We design a TM M that, given positive integers x, y, halts in q_y if $x \geq y$ and in q_n if $x < y$. That is,

$$\begin{cases} q_0 w(x) 0 w(y) \overset{*}{\vdash} q_y w(x) 0 w(y), & \text{if } x \geq y, \\ q_0 w(x) 0 w(y) \overset{*}{\vdash} q_n w(x) 0 w(y), & \text{if } x < y. \end{cases}$$

- Essentially we are matching the 1's to the left of 0 to the 1's to the right of 0. This is similar to recognizing $a^n b^n$.

Conditional Statement

- Now we can implement a conditional statement,

$$f(x,y) = \begin{cases} x+y, & \text{if } x \geq y, \\ 0, & \text{if } x < y. \end{cases}$$

- We have a comparer C, an adder A, and an eraser E. x, y are compared, then either added or erased.
- The implementation goes like

$$\begin{cases} q_{C,0}w(x)0w(y) \overset{*}{\vdash} q_{A,0}w(x)0w(y) \overset{*}{\vdash} q_{A,f}w(x+y)0, & \text{if } x \geq y, \\ q_{C,0}w(x)0w(y) \overset{*}{\vdash} q_{E,0}w(x)0w(y) \overset{*}{\vdash} q_{E,f}0, & \text{if } x < y. \end{cases}$$

Pseudo-code

- In designing or describing a computer program, pseudo-codes are useful for outlining the main ideas.
- When using pseudo-codes, we assume that we can translate the description to a programming language.
- The same can be said about TM. We assume that we can implement pseudo-codes by TMs. In particular, function calls can be implemented by TM.

Turing Thesis

- Turing thesis claims that any computation that can be carried out by mechanical means can be performed by a TM.
- This is not something provable. It is indeed a *definition* for mechanical computation: a computation is mechanical iff it can be done by a TM.
- According to the Turing thesis, if we can do something with a computer program, then we can do it by a TM.
- Thus, to show something is computable by TM, we can simply give a pseudo-code or block diagram. This saves us from the trivial task of constructing TM.

Equivalent Classes of Automata

- Two automata are said to be equivalent if they accept the same language.
- Two classes of automata are said to be equivalent if every automaton of the first class is equivalent to an automaton in the second class, and vice versa.
  - For example, the classes of dfa's and nfa's are equivalent.
  - If the converse is not sure to be true, we say that the second class is *at least as powerful* as the first.

Multitrack-tape TM

- We want to show that some variants of TM have the same descriptive powers as standard TM.
- We start with a TM with a tape of multiple tracks, called a multitrack-tape TM.
- This is equivalent to a standard TM using the Cartesian product of track alphabets as the tape alphabet. That is,

  $$\Gamma = \Gamma_1 \times \Gamma_2 \times \ldots$$

  So everything that can be done by a multitrack-tape TM can be done by a standard TM.

TM with Stay Option

- Instead of always moving left or right, the head can stay in a move with the stay option.
- The transition function is modified to be

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

- The class of TMs with stay option is equivalent to the class of standard TMs. A TM M with stay option can be simulated by a standard TM $\widehat{M}$: A move of M involving S can be simulated by two moves in $\widehat{M}$,

$$\delta_M(p, a) = (q, b, S) \Rightarrow \begin{cases} \delta_{\widehat{M}}(p, a) = (r, b, R) \\ \delta_{\widehat{M}}(r, *) = (q, *, L) \end{cases}$$

TM with Semi-infinite Tape

- A semi-infinite tape has a left boundary. The head above the tape cannot move further left at the boundary.
- A standard TM M can be simulated by a TM $\widehat{M}$ with semi-infinite tape. It follows that the class of TMs with semi-infinite tapes is equivalent to the class of standard TMs.
- The simulating $\widehat{M}$ has a two-track semi-infinite tape.
- The upper (lower) track stores the content of M's tape to the right (left) of some reference point.

Simulation

- The set of states of $\widehat{M}$ is partitioned into two subsets $\hat{q}_j$s, $\hat{p}_j$s. From the state of $\widehat{M}$ we know which part of tape M is working on.

$$\delta_M(q_i, a) = (q_j, c, L) \Rightarrow \begin{cases} \delta_{\widehat{M}}(\hat{q}_i, (a, *)) = (\hat{q}_j, (c, *), L) \\ \delta_{\widehat{M}}(\hat{p}_i, (*, a)) = (\hat{p}_j, (*, c), R) \end{cases}$$

- End markers are used to facilitate the transition between the two regions. For a move to the left passing the reference point, we have

$$\delta_{\widehat{M}}(\hat{q}_j, (\#, \#)) = (\hat{p}_j, (\#, \#), R)$$

- See Figure 10.4 for illustration.

Off-line TM

- A standard TM has no input file. An off-line TM permits the use of input files (read-only).
- The transition function depends on the state, tape symbol and the input symbol.
- The class of off-line TM is equivalent to the class of standard TM. A off-line TM M can be simulated by a standard TM $\widehat{M}$ with a four-track tape:
  - track 1: input content of M
  - track 2: input position of M
  - track 3: tape content of M
  - track 4: head position of M

Multitape TM

- A multitape TM may have more than one tapes, each with its own head.
- The transition function specifies the moves of all tapes

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

- The class of multitape TM is equivalent to the class of standard TM. An n-tape TM can be simulated by a standard TM with a $2n$-track tape. Each track keeps track of either head position or tape content.
- It is often easier to work with a multitape TM, e.g. to accept $\{a^n b^n\}$ (Example 10.1).

Multidimensional TM

- A multidimensional TM has a tape extending infinitely in more than one direction.
- The transition function for a 2-D TM is

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

  That is, the head can move left, right, up, or down in one transition.
- The class of multidimensional TMs is equivalent to the class of standard TMs.

Simulation

- We can simulate a multidimensional TM M with a standard TM $\widehat{M}$ with a two-track tape.
- We need an address scheme for cells in the multidimensional tape. This is not difficult to do with a reference point.
- To simulate one move of M, $\widehat{M}$ computes the new address of the cell under the head. It then modifies the position track, and the content track.
- The simulation of 2-D TM by a 2-track TM is illustrated in Figure 10.13.

Nondeterministic TM

- A nondeterministic TM permits multiple choices of next move. The value of the transition function is a set,

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

- A string w is accepted by a nondeterministic TM M if there exists a sequence of candidate moves such that

$$q_0 w \overset{*}{\vdash} x_1 q_f x_2, \; q_f \in F.$$

- Following some candidates may lead to a non-final halt state or an infinite loop, but they are irrelevant to acceptance.

Parallelized View

- A nondeterministic TM has the ability to replicate itself when necessary.
  - When there are more than one candidate moves are, the TM produces as many replicas as needed and gives each replica the task to follow one candidate.
  - If any of the replicas ever succeeds in reaching a final state, then the input string is accepted.
- Conceptually, we are exploring the possibilities simultaneously.

Simulation

- A nondeterministic TM M can be simulated by a (deterministic) TM $\widehat{M}$ with a 2-D tape.
- Every two horizontal tracks represents one replica. One track is used for tape content and the other is used for head position and internal state.
- $\widehat{M}$ looks at an active configuration and updates the tape content as new replicas are created.
- Note that a depth-first search for a successful candidate is not a good idea.