CSCI 2670, Fall 2012

Introduction to Theory of Computing

Department of Computer Science
University of Georgia
Athens, GA 30602

Instructor: Liming Cai
www.cs.uga.edu/∼cai

Lecture Note 4

The Church-Turing Thesis

Chapter 3. The Church-Turing Thesis

A part of **Part Two: Computability Theory** consisting of

Chapter 3. The Church-Turing Thesis
Chapter 4. Decidability
Chapter 5. Reducibility
Other advanced topics

What is **Part Two** about?

- to investigate computability, i.e.,
  what problems are computable.
- but this requires a precise definition on
  computability
- to achieve the definition, we need
  a formal computation model

Chapter 3 - defines Turing machine as the computation model
Chapter 4 - study some problems that are decidable/not decidable
Chapter 5 - study the equivalence between decidable problems

- We will study two kinds of computations:

  <u>Algorithms: Turing computations that eventually halts</u>

  <u>Turing computation that may not halt</u>

- Computable problems are those solvable by algorithms

  (1) computable decision problems: *decidable languages*

  (2) computable search problems: *computable functions*

3.1 Turing Machines

A *Turing machine* consists of

(1) an infinite *input tape*, where the input is placed,
(2) a *read head* moving left and right on input and beyond, and can read and write on the input tape,
(3) a set of states for transitions, and
(4) *accepting* and *rejecting* states, for the machine to *halt*.

Turing machines can do what PDA/CFG cannot do.

E.g., to recognize language $\{a^n b^n c^n \mid n \geq 0\}$

how?

the read head can count the numbers of a, b, and c, by

- scanning back and forth, and
- marking the read ones with special symbols.

Another example: $\{w\#w \mid w \in \Sigma^*\}$

- This looks different from $\{ww \mid w \in \Sigma^*\}$
- But can it be recognized by a PDA?
- Turing machines can recognize it!

Formal definition of a Turing machine

Definition 3.3

A *Turing machine* is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet, not including the *blank symbol* $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{accept} \neq q_{reject}$.

- about input tape:

    when it starts, the input tape has content $x_1x_2\ldots x_n$, and the rest of the tape consists of blank symbols $\sqcup$'s

- about transition function δ:

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

    cannot move off the leftmost position, even if L is used.

- about halting:

    entering q_{accept} and q_{reject} halts the machine immediately,

    infinite loop, if neither state is entered.

Use *configuration* to define the machine status at any moment

- consisting of current state,
- current read head position,
- current tape content,

e.g., configuration `1 0 1 1`q_7`0 1 1 1 1`, assume $\Sigma = \{0, 1\}$,

e.g., initial input content, assume $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{\#, \$, \&, \sqcup\}$

`a a a a b b b b c c c c`

a few steps later:

`# #`q_4`a a $ $ b b & & c c`

Note that the read head points to the symbol to the right of the state id.

Formalizing the intuitive way a Turing machine computes:

Configuration C_1 **yields** configuration C_2
if the machine can go from C_1 to C_2 in a single step.

Formally, let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, and $q_i, q_j \in Q$. We say

$$uaq_ibv \text{ yields } uq_jacv$$

if $\delta(q_i, b) = (q_j, c, L)$, and

$$uaq_ibv \text{ yields } uacq_jv$$

if $\delta(q_i, b) = (q_j, c, R)$.

The *start configuration* is $q_0 w$
where w is the input.

An configuration is *accepting configuration*
if its state is the accepting state.

An configuration is *rejecting configuration*
if its state is the rejecting state.

Both accepting and rejecting configurations are halting configurations.

Definition. A Turing machine M *accepts* an input w if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$, exists, such that

1. C_1 is the start configuration of M on input w,
2. C_i yields C_{i+1}, for $i = 1, 2, \ldots, k-1$, and
3. C_k is an accepting configuration.

The collection of strings accepted by M is

the language of M, or
the language recognized by M.
Denoted $L(M)$.

Definition 3.5 A language A is *Turing recognizable* if $A = L(M)$ for some Turing machine M. (A is also called **recursively enumerable**)

Note that on an input w, a Turing machine may accept and halt, reject and half, or never halt.

A Turing machine *decides* if it halts. It is called *decider*.

Definition 3.6 A language A is *Turing decidable* or simply *decidable* if $A = L(M)$ for some Turing machine decider M. (A is also called **recursive**)

Examples of Turing machines

Example 3.7, Computing a power of 2: 2^n.

to recognize language $\{0^{2^n} \mid n \geq 0\}$,

consists of strings: `0`, `00`, `0000`, `00000000`.

idea: 2^n can be repeatedly divided by 2, with the result 1.

if any phase of division has odd number of 0's left,

- accept if the number is 1,
- reject otherwise.

Turing machine (high-level description) for recognition of language $\{0^{2^n} \mid n \geq 0\}$ (page 143).

M_2 on the input string w:

1. Scan left to right the tape, crossing off every other 0,
2. If in step 1, the tape contains a single 0, accept.
3. If in step 1, the tape contains k 0s, for odd $k \geq 3$, reject.
4. Return the head to the leftmost position,
5. Goto step 1.

Formally $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$

$Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$,

$\Sigma = \{0\}$,

$\Gamma = \{0, \times, \sqcup\}$,

Start state is q_1,

Accept state is q_{accept}; reject state is q_{reject}, and

δ is described with a state diagram (Figure 3.8).

Two kinds of labels on transition edges:

(1) $a \longrightarrow b, M$, and
(2) $a \longrightarrow M$

where $a, b \in \Gamma$, $M \in \{L, R\}$.

Explanation for Figure 3.8.

- from q_1 to q_2, mark the first 0 as $\sqcup$ to indicate the leftmost
- if no - q_2 move right, skips all $\times$'s, if no more 0, go to q_{accept}, or
- cross the first encountered 0, go to q_3
- q_3 together with q_4 skip all $\times$'s, cross every other 0
- if not even number of 0, from q_4 go to q_{reject}
- reach the rightmost, move left, go to q_5
- move left, q_5 skips all $\times$'s and 0's
- reach the leftmost, go to q_2

Configuration changes for input `0000` at the bottom of page 144.

Example 3.9 $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ to recognize language $\{w\#w \mid w \in \{0,1\}^*\}$.

$Q = \{q_1, q_2, \ldots, q_8, q_{accept}, q_{reject}\}$,

$\Sigma = \{0, 1, \#\}$,

$\Gamma = \{0, 1, \#, \times, \sqcup\}$,

Start state is q_1,

δ is given as the diagram Figure 3.10.

Explanation for Figure 3.10

- from q_1, go to q_2 if read 0; go to q_3 if read 1, cross current symbol
- from q_2, move right, skip all 0's and 1's, until reach #, go to q_4,
- from q_4, move right, skip all $\times$'s until 0, go to q_6, move left
- from q_6, move left, skip all 0's, 1's, $\times$'s, until #, go to q_7, move left
- from q_7, move left, skip all 0's, 1's until $\times$, go to q_1, move right

- from q_3, symmetrically similar to from q_2.

- from q_1, if # (no more 0's or 1's on its left), go to q_8, move right
- from q_8, if no more 0's or 1's, go to q_{accept}

Can you trace configuration changes for input `010#010` ?

Turing machines can do arithmetics

e.g., addition of two integers

- represent integers, 1^x (unary) represents x

  input $1^x + 1^y = 1^z$, accept iff $x + y = z$.

  e.g., `111+11=11111`, how would a TM do it?

- binary representation, $x, y, z \in \{0, 1\}^*$

  input $x + y = z$, accept iff $x + y = z$.

  e.g., `011+10=101`, how would a TM do it?

3.2 Variants of Turing Machines

Multitape Turing machines

- Have k tapes, each with a head to read and write
- $k-1$ tapes start out blank
- Transition function $\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

Theorem 3.13 Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof idea:

- assigning space for k tapes using delimiter, e.g., #
- shift right to get more space

Nondeterministic Turing machines

- Transition function $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$

Theorem 3.16 Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:

- computation of NTM is a tree (possiblyinfinite)
- each node is a configuration, the root is the start configuration
- from a parent, there can be m children configuration
- breadth-first-search (why depth-first-search may not work?)

Technical details for the proof:

- A DTM simulates the computation of a given NTM on input
- search through the computation/configuration tree
- use input tape, address tape
    - address store current path from the root to the current level
    - in the form of 1 2 1 3 2 3 at level 6, excluding the root level
    - assuming 3 branches
- simulation tape (simulate deterministically, given the path)

Enumerators

A Turing machine that prints strings on the *output* tape, separated by a special symbol, say #, is called an **enumerator**.

- It starts from an empty input tape.

- The collection of all strings printed on the output tape is the language enumerated by the machine.

- The order of strings printout can be in any order.

- The collection can be infinite.

Theorem 3.21 A language can be recognized by a TM if and only if there is an enumerator enumerates it.

Proof ideas for:

(1). Turing enumerable $\Longrightarrow$ Turing recognizable.

- assume E an enumerator for language L,

- construct M to recognize input w by
  run E, and compare w with all strings output by E
  if w ever appears in the output of E, M accepts w

- if w is in $L(E)$, it will get printed on the tape eventually, so will be accepted by M.

strings outputted may repeat

(2) Turing recognizable $\Longrightarrow$ Turing enumerable.

- assume M a Turing machine that recognizes language L.

- construct E to enumerate $L(M)$ by
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots,$
    simulate M on w
    if M accepts w, print w out.

  (but would this work?)

  - **Let** $s_1 = \epsilon$**,** $s_2 = 0$**,** s_3**,** $s_4 = 00$**,** $\ldots$,

  - **for** $i = 1, 2, \ldots$**,**
    run M i **steps on each of** $s_1, s_2, \ldots, s_i$
    if M **accepts** s_j**, print** s_j**.**

Other equivalent models

Turing machines with two-way tapes

Turing machines with multi-cells at each position of tapes

Parallel models

- PRAM
- Boolean circuits
- parallel computers (shared memory, message passing)
- distributed systems

Other models

quantum computers
bio-computers
chemical computers

3.3 Definition of Algorithm

But first, an old slide (Slide 13):

Definition 3.5 A language A is *Turing recognizable* if $A = L(M)$ for some Turing machine M. (A is also called **recursively enumerable**)

Note that on an input w, a Turing machine may accept and halt, reject and half, or never halt.

A Turing machine *decides* if it halts. It is called *decider*.

Definition 3.6 A language A is *Turing decidable* or simply *decidable* if $A = L(M)$ for some Turing machine decider M. (A is also called **recursive**)

There are two kinds of Turing machines

(1) those always can halt on all inputs

- the instruction set built in such a machine is called an **Algorithm**

(2) those may not halt on some inputs

Church-Turing Thesis:

Everything computable is computable by a Turing machine.

Three formal systems:
The first-order logic,
λ-calculus, and
Turing machines

Three programming (language) paradigms:
logic programming languages: prolog
functional programming languages: LISP
procedural programming languages: C, etc

Q1: What problems cannot be solved by **algorithms**?

i.e., What problems may not correspond to

decidable/recursive languages?

Q2: What problems cannot be solved by Turing machines (that may not halt)?

i.e., What problems may not correspond to

Turing recognizable/recursively enumerable languages?

Examples of problems that have Algorithms

Problem 1: Testing if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$ has an integral root

That is to test if there is an integer solution for the equation $6x^3 - 3x^2 + 24x - 17 = 0$

There is a finite process to decide the question, given such a polynomial.

- enumerating all integers 0, -1, 1, -2, 2, ...
- the process is not infinite because
  the root should be within the range $[-k\frac{c_{max}}{c_1}, +k\frac{c_{max}}{c_1}]$

Problems 2: Testing if a given graph is connected (Example 3.23, page 157)

$$A = \{\langle G \rangle \mid G \text{ is connected } \}$$

A is decidable, i.e., there is an algorithm (TM that halts) for it.

on input $\langle G \rangle$, encoding of G,

1. select the first node and mark it
2. repeat the following until no new nodes are marked
3. for each node in G, mark it if it shares an edge with a marked node
4. if all nodes are mark, *accept*, otherwise *reject*

Illustration by Figure 3.24 (page 158)

Examples of problems that do not have Algorithms

Problem 3. Testing if a polynomial has an integral root

$$\text{e.g., } 6x^3yz^2 + 3xy^2 - x^3 - 10$$

- the polynomial may contain multiple variables (e.g., x, y, z)
- it is not possible to get bounds for these variables
- an enumerating process similar to the algorithm for Problem 1 may not halt

- it is not decidable
- But it is solvable by a TM (which may not stop)
- The corresponding encoded language is recursively enumerable/Turing recognizable.

We will see more undecidable languages

We will also see some languages
that are not even Turing recognizable