Introduction to The Theory of Computation

Chap. 1

Summary

- Let's review some of the main ideas from *finite mathematics* and establish the *notation* used in the text.

  Important is the *proof techniques*:

  proof by deduction, induction or by contradiction.

- The main ideas of the course:
  Automata, Languages, Grammars, their definitions, and their relations. ➔ Extended to a number of different types of automata, languages and grammars.

- Some simple examples of the role these concepts in Computer Science, particularly in Programming Languages, digital design, and text processing.

  These issues will encounter applications of these concepts in a number of other CS courses, e.g.) CSci 465

Learning Objectives

- Define the 3 basic concepts in the Theory of Computation:
  - Machine (Automaton, Turing Machine),
  - Formal Language (Regular language, Context-Free language, etc.), and
  - Grammar (Regular grammar, Context-Free grammar, etc.)
- Evaluate expressions involving operations on strings and on languages.
- Generate strings from simple grammars.
- Construct grammars to generate simple languages.
- Describe the essential components of an automaton.
- Design grammars to describe simple programming constructs.

Basic Concepts:
Automaton -- Language -- Grammar

- *Automaton*: a formal construct that accepts input, produces output, may have some temporary storage, and can make decisions.
  - The abstract mathematical model of modern computer.
- *Formal Language*: a set of sentences formed from a set of symbols according to formal rules.
- *Grammar*: a set of rules for generating the sentences in a formal language.

  In addition,
- *Computability*: the types of problems computers can solve *in principle* – decidability, acceptability
- *Complexity*: the types of problems that can be solved in *practice* – time/space complexity

Mathematical Preliminaries

- *Sets*: basic notation, operations (union, intersection, difference, and complementation), disjoint sets, power set, partitions.
- *Functions and Relations*: domain, range, total function, partial function, order of magnitude, equivalence relations.
- *Graphs and Trees*: vertices, edges, walk, path, simple path, cycle, loop, root vertex, parent, child, leaves, depth, height.
- *Proof Techniques*: proof by deduction, proof by induction, proof by contradiction.

Proof by Deduction

- A proof where a statement is proved to be true based on well-known mathematical principles; i.e. establish facts through *reasoning* or make conclusions about a particular instance by referring to a general rule or principle.
- It may use the algebraic symbols and construct logical arguments from known facts to show that something is true for all instances.
- Example: Prove that the difference between the squares of any two consecutive integers is equal to the sum of those integers.

Proof) Choose any two consecutive integers, n and n+1.

Then, take the squares of these integers: n^2 and $(n+1)^2 = n^2 + 2n+1$.

The difference between these squares is $(n^2 + 2n+1) - n^2 = 2n+1$ (A)

The sum of the original two consecutive integers is: $n + (n+1) = 2n+1$ (B).

Therefore, the given claim is true since the above (A) and (B) are equal.

Q.E.D.

Proof by Induction

- A proof by which the truth of a number of statements can be inferred from the truth of a few specific instances.
- Suppose we have a sequence of statements P_1, P_2, ... , and we want to prove P_k to be true, for all $k \geq 1$. Suppose the following holds:
  1. For some $k \geq 1$, the starting statement(s) P_1, (P_2, ..., P_k) are true.
  2. The problem is s.t. for any $n \geq k$, the truths of P_1, P_2,..., P_n imply the truth of P_{n+1}.

Use induction to show that every statement in this sequence is true.

- Base case:
  For some $k \geq 1$, the starting statement(s) P_1, (P_2, ..., P_k) are true.
- Inductive Hypothesis (I.H.):
  Assume that P_1, P_2, ..., P_n , $n \geq k \geq 1$ are true.
- Inductive Step:
  Prove P_{n+1} is true using Inductive Hypothesis and Base case.

Therefore, the given statement P_k is true for all $k \geq 1$.

Proof by Induction (cont.)

- <u>Example</u>: A binary tree of height h has at most 2^h leaves.

Proof by Induction) Let $l(h)$ denote the maximum number of leaves of a binary tree of height h.

<u>Claim</u>: Show that $l(h) \leq 2^h$.

<u>Basis: $h = 0$</u>.

$l(0) = 1 = 2^0$ since a tree of height 0 has a root only, i.e. it has at most one leaf. Thus, $l(h) \leq 2^h$ for h=0.

<u>Inductive Hypothesis</u>: Assume that $l(h) \leq 2^h$ is true for $h = 0, 1, ..., n$.

<u>Inductive Step</u>: Let's prove that a binary tree of height h+1 has at most 2^{h+1} leaves, i.e. $l(h+1) \leq 2^{h+1}$.

To get a binary tree of height h+1 from one of height h, we can create it by merging at most two binary trees T_H, T_R, of height h, adding a new root.

Thus, $l(h+1) = l(T_H) + l(T_R) = l(h) + l(h) = 2 \cdot l(h)$.

Hence, $l(h+1) = 2 \cdot l(h) \leq 2 \cdot 2^h = 2^{h+1}$ by I.H. The claim is true for h+.

Therefore, $l(h) \leq 2^h$ for all $h \geq 0$.

i.e. A binary tree of height h has at most 2^h leaves for any height h. *Q.E.D.*

Proof by Contradiction

- A proof that determines the truth of a statement by assuming the proposition is false, then working to show its falsity until the result of that assumption is a contradiction.
- A disproof by counterexample also belongs to it.
- <u>Example:</u> Disprove that for any a, $b \in Z$, if $a^2 = b^2$, then $a = b$.

By CounterExample) Z is the set of all positive or negative integers.

If an a and b s.t. $a \neq b$ but $a^2 = b^2$, then the statement is disproved.

Choose any integer for a, then choose $b = -a$.

Then, $a^2 = b^2 = (-a)^2$, but $a \neq b$ $(= -a)$.

e.g.) $a = 4$, $b = -4 \rightarrow a^2 = b^2 \Leftrightarrow 4^2 = (-4)^2 = 16$, *but* $a \neq b$

Thus, the given statement is false: a is not necessarily equal to b.

Q.E.D.

Proof by Contradiction (cont.)

- <u>Example:</u> For all integers n, if n^3+5 is odd, then n is even.

Proof) Let n be any integer. Suppose that n^3+5 and n are both odd.

Then, there exist integers j and k s.t. $n^3+5= 2k+1$ and $n = 2j+1$.

Substituting for n we have:

$$\begin{aligned} 2k+1 &= n^3+5 = (2j+1)^3+5 \\ &= 8j^3 + 3(2j)^2 1 + 3(2j)(1)^2 + 1^3 + 5 \\ 2k &= 8j^3 + 12j^2 + 6j + 5 \end{aligned}$$

Dividing by 2 and rearrange it yields

$$* \ k - 4j^3 - 6j^2 - 3j = 5/2 \ \ **$$

-- impossible because 5/2 ** is a non-integer rational number while * is an integer by the closure properties for integer.

Thus, the assumption 'n is odd' is false, i.e. n must be even.

Formal Languages: Basic Concepts

- *Alphabet*: a set of symbols, i.e. $\Sigma = \{a, b\}$
- *String*: a finite sequence of symbols from Σ, such as $v = aba$ and $w = abaaa$
  - So, any string $u \in \Sigma^*$
  - Empty string: λ, ε
  - Substring, prefix, suffix
- *Operations on strings*:
  - Concatenation: $vw = abaabaaa$
  - Reverse: $w^R = aaaba$
  - Repetition: $v^2 = abaaba$ and $v^0 = \lambda$ *(empty string)*
- *Length of a string:* $|v| = 3$ and $|\lambda| = 0$

Formal Languages: Property

- Example 1.8: For the strings u, v, $|uv| = |u| + |v|$.

Proof by Induction)

First, let's define the length of a string recursively:

$|a| = 1$, $|ua| = |u| + 1$, for any $a \in \Sigma$ and any string u on Σ^*.

Base case: For all u of any length and all v of length 1, i.e. $|v|=1$,

$|uv| = |u| + 1 = |u|+|v|$ where $v \in \Sigma$. Holds.

Inductive Hypothesis (I.H.): Assume that $|uv| = |u| + |v|$

for all u of any length and all v of length $k \leq n$, i.e. $|v| \leq n$.

Inductive Step: For any v of $|v|= n+1$, rewrite v as $v = wa$ where $|w| = n$. Then, $|v| = |w| + 1$, $|uv| = |uwa|= |uw| + 1$.

By I. H. , $|uw| = |u| + |w|$ since $|w|= n$, so that

$|uv| = |uwa|= |uw| + 1 = |u| + |w| + 1 = |u| + |wa| = |u|+|v|$.

Therefore, $|uv| = |u| + |v|$ for all u and v of any length. Q.E.D.

Formal Languages: Definitions

- Σ^* = a set of *all strings* formed by concatenating *zero or more* symbols in Σ.
- Σ^+ = a set of all *non-empty strings* formed by concatenating symbols in Σ.

In other words, $\Sigma^+ = \Sigma^* - \{ \lambda \}$

- A *formal language* is any subset of Σ^*

Example 1.10: $\Sigma = \{a, b\}$

$L_1 = \{ a^n b^n \mid n \geq 0 \}$ and $L_2 = \{ ab, aa \}$

- A string in a language is also called a *sentence* of the language.

Formal Languages: Set Operations

- A language is a set of strings.

  Thus, set operations are defined as usual.

- If $L_1 = \{ a^n b^n \mid n \geq 0 \}$ and $L_2 = \{ ab, aa \}$ where $\Sigma = \{a, b\}$
  - *Union:* $L_1 \cup L_2 = \{ aa, \lambda, ab, aabb, aaabbb, \ldots \}$
  - *Intersection:* $L_1 \cap L_2 = \{ ab \}$
  - *Difference:* $L_1 - L_2 = \{ \lambda, aabb, aaabbb, \ldots \}$

    $= \{ a^n b^n \mid n = 0 \text{ or } n \geq 2 \}$
  - *Complement:* $\overline{L_2} = \Sigma^* - L_2 = \Sigma^* - \{ab, aa\}$
- Find $L_2 - L_1$?

Formal Languages: Other Operations

- *Reversal* of all strings in a language:
  - $L^R = \{ w^R \mid w \in L \}$
- *Concatenation* of strings from *two* languages, and
  - $L_1L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$
- Concatenation of strings from the *same* language:
  - $LL = \boldsymbol{L^2} = \{ xy \mid x \in L, y \in L \}$
- *Star-Closure*: $L^* = \boldsymbol{L^0} \cup L^1 \cup L^2 \cup L^3 \cup \ldots$

  where $\boldsymbol{L^0} = \{ \lambda \}$, $L^1 = L$, $L^2 = L \cdot L$, etc.
- *Positive Closure:* $L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots$

Example: Other Operations

- If $L_1 = \{ a^n b^n \mid n \geq 0 \}$ and $L_2 = \{ ab, aa \}$
  - *Reversal:* $L_2^R = \{ ba, aa \}$, $L_1^R = \{ b^n a^n \mid n \geq 0 \}$
  - *Concatenation:* $L_1L_2 = \{ ab, aa, abab, abaa, aabbab, aabbaa, \ldots \}$

    Concatenation: $L_2L_2 = \boldsymbol{L_2^2} = \{ abab, abaa, aaab, aaaa \}$
  - *Star-Closure:* $L_2^* = \boldsymbol{L_2^0} \cup L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
  - *Positive Closure:* $L_2^+ = L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
- Find $(L_2 - L_1)^R$?

Grammars: Definition

- A rule to describe the strings in a language.
- In English grammar:
  - *<sentence>* → *<noun phrase>* *< predicate>*,
  - *<noun phrase>* → *<article>* *<noun>*,
  - *<predicate>* → *<verb>*,
  - *<article>* → a | the,
  - *<noun>* → boy | dog,
  - *<verb>* → runs | walks.
- Example: a boy walks, the dog runs.

Grammars: Definition

- A *rule* to describe the strings in a language, i.e. a *syntax* of a language – not a semantics.
- Def. 1.1: A grammar G is defined as a quadruple

  G = (V, T, S, P) where

  V: a *finite* set of *variable* or *non-terminal symbols*

  T: a *finite* set of *terminal* symbols

  S ($\in$ V): a variable called the *start* symbol

  P: a *finite* set of *productions (i.e. rules)*
- Example 1.11:

  V = { S }

  T = { *a*, *b* }

  P = { S → *aSb*, S → λ } → L(G) = $\{ a^n b^n \mid n \geq 1 \}$

Grammars: *Derivation* of Strings

- Beginning with the *start symbol*, strings are derived by repeatedly replacing variable symbols with the expression on the right-hand side of any applicable production.
- Any applicable production can be used, in arbitrary order, until the string contains no variable symbols.
- Sample derivation using grammar in Ex. 1.11:

$$S \rightarrow aSb,\ S \rightarrow \lambda$$

$S \Rightarrow aSb$ (applying 1st production)

$\Rightarrow aaSbb$ (applying 1st production)

$\Rightarrow aabb$ (applying 2nd production)

The Language generated by a Grammar

- Def. 1.2: For a given grammar G=(V, T, S, P), *the language generated by G,*

$$L(G) = \{\, w \in T^* \mid S \Rightarrow^* w \,\}$$

is the set of all strings derived from the start symbol.

- To show a language L is generated by G: $L = L(G)$
  - Show every string in L *can be* generated by G ***and***

    $\forall w \in L \rightarrow \forall w \in L(G)$.
  - Show every string generated by G is in L.

    $\forall w \in L(G) \rightarrow \forall w \in L$.
- A given language can normally be generated by different grammars.

The Language generated by a Grammar

- For convenience, productions with the same left-hand sides are written on the same line:

$$S \to A \mid B \Leftrightarrow \quad S \to A,\ S \to B$$

- Example 1.13: For a given grammar G=(V, T, S, P) with productions $S \to SS \mid \lambda \mid aSb \mid bSa,$

  find $L(G) = ?$

  $L(G) = \{ w \mid \ ? \}$

Equivalence of Grammars

- Two grammars, G_1 and G_2, are *equivalent* if they generate the same language: $L(G_1) = L(G_2)$.
- For convenience, productions with the same left-hand sides are written on the same line: $S \to A \mid B$ (= $S \to A,\ S \to B$)
- Example 1.11:

  G_1= (V, T, S, P) where
  $V = \{ S \},\ T = \{ a, b \},$
  $P = \{S \to aSb \mid \lambda \ \}$
- Example 1.14:

  G_2= (V, T, S, P) where
  $V = \{ A, S \}, T = \{ a, b \},$
  $P = \{ S \to aAb \mid \lambda$
  $\quad A \to aAb \mid \lambda \ \}$

G_1 and G_2 are equivalent since $L(G_1) = L(G_2)$.

Automata

- An *Automaton* is an *abstract mathematical model* of a (von Neumann) digital computer.
- An automaton consists of
  - An *input* mechanism
  - A *control unit*
  - Possibly, a *storage* mechanism
  - Possibly, an *output* mechanism
- Control unit can be in any number of internal *states*, as determined by a *next-state* or *transition* function.

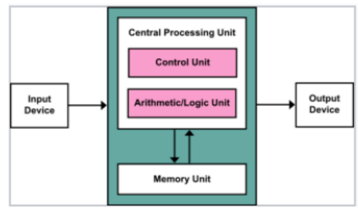
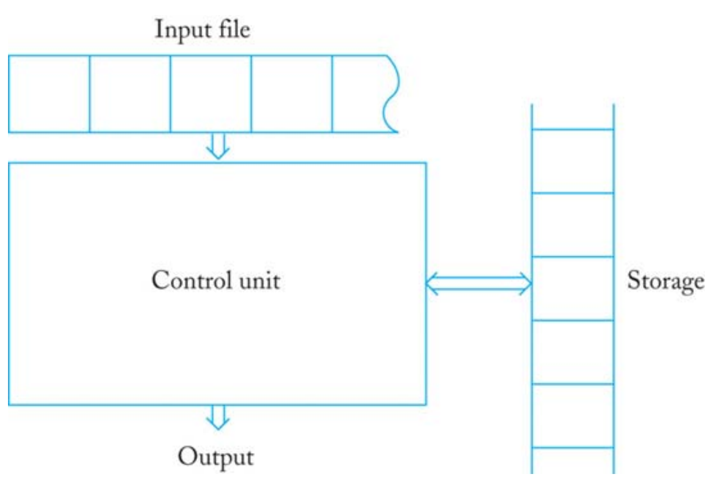


Figure: https://en.wikipedia.org/wiki/Von_Neumann_architecture

Diagram of a General Automaton



Application:
Grammars for Programming Languages

- The syntax of constructs in a programming language is commonly described with grammars.
- Assume that in a hypothetical programming language,
  - Identifiers consist of digits and the letters *a, b*, or *c*
  - Identifiers must begin with a letter
- Productions for a sample grammar:

  <id> → <letter> <rest>

  <rest> → <letter> <rest> | <digit> <rest> | λ

  <letter> → *a* | *b* | *c*

  <digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- Ref.) CSci 465. Principles of Translation. Compiler