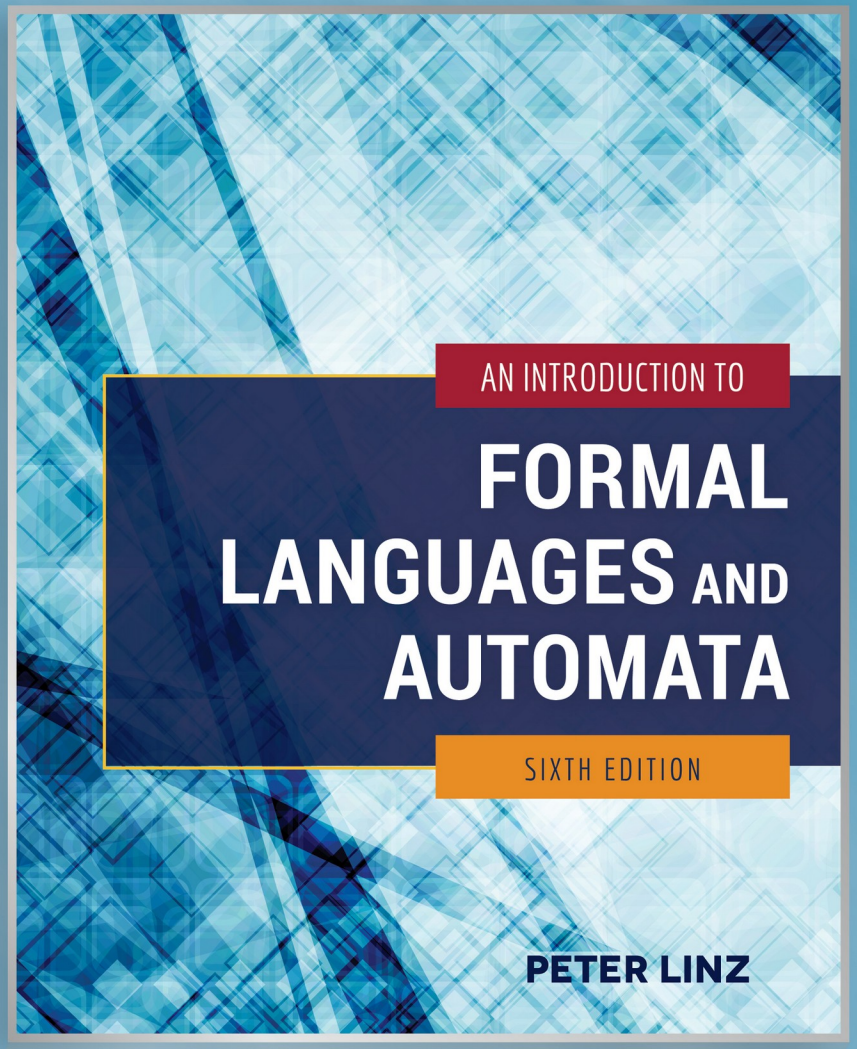
Chapter 1

INTRODUCTION TO THE THEORY OF COMPUTATION



Learning Objectives

At the conclusion of the chapter, the student will be able to:

- Define the three basic concepts in the theory of computation: automaton, formal language, and grammar.
- Solve exercises using mathematical techniques and notation learned in previous courses.
- Evaluate expressions involving operations on strings.
- Evaluate expressions involving operations 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.

Theory of Computation
Basic Concepts

- *Automaton*: a formal construct that accepts input, produces output, may have some temporary storage, and can make decisions
- *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, the theory of computation is concerned with questions of computability (the types of problems computers can solve in principle) and complexity (the types of problems that can solved in practice).

Review of 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, level, height.
- *Proof Techniques*: proof by induction and proof by contradiction.

Formal Languages
Basic Concepts

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

Formal Languages
Definitions

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

Examples: $L_1 = \{ a^n b^n : n \geq 0 \}$ 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. Therefore, set operations are defined as usual.
- If $L_1 = \{ a^n b^n : n \geq 0 \}$ and $L_2 = \{ ab, aa \}$
  - Union: $L_1 \cup L_2 = \{ aa, \lambda, ab, aabb, aaabbb, \ldots \}$
  - Intersection: $L_1 \cap L_2 = \{ ab \}$
  - Difference: $L_1 - L_2 = \{ \lambda, aabb, aaabbb, \ldots \}$
  - Complement: $L_2 = \Sigma^* - \{ ab, aa \}$
- Practice: Find $L_2 - L_1$

Formal Languages
Other Operations

- New languages can be produced by reversing all strings in a language, concatenating strings from two languages, and concatenating strings from the same language.
- If $L_1 = \{ a^n b^n : n \geq 0 \}$ and $L_2 = \{ ab, aa \}$
  - Reverse: $L_2^R = \{ ba, aa \}$
  - Concatenation: $L_1 L_2 = \{ ab, aa, abab, abaa, aabbab, aabbaa, \ldots \}$

    The concatenation $L_2 L_2$ or $L_2^2 = \{ abab, abaa, aaab, aaaa \}$
  - Star-Closure: $L_2^* = L_2^0 \cup L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
  - Positive Closure: $L_2^+ = L_2^1 \cup L_2^2 \cup L_2^3 \cup \ldots$
- Practice: Find $(L_2 - L_1)^R$

Grammars
Definition

- Precise mechanism to describe the strings in a language
- Def. 1.1: A grammar G consists of:
  - V: a finite set of *variable* or non-terminal symbols
  - T: a finite set of *terminal* symbols
  - S: a variable called the *start* symbol
  - P: a set of *productions*
- Example 1.11:

$$V = \{ S \}$$

$$T = \{ a, b \}$$

$$P = \{ S \rightarrow aSb, S \rightarrow \lambda \}$$

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 Example 1.11:

$S \Rightarrow aSb$ (applying first production)

$\Rightarrow aaSbb$ (applying first production)

$\Rightarrow aabb$ (applying second production)

The Language Generated by a Grammar

- Def. 1.2: For a given grammar G, the language generated by G, L(G), is the set of all strings derived from the start symbol
- To show a language L is generated by G:
  - Show every string in L can be generated by G
  - Show every string generated by L is in G
- A given language can normally be generated by different grammars

Equivalence of Grammars

- Two grammars are equivalent if they generate the same language
- For convenience, productions with the same left-hand sides are written on the same line
- Example 1.14:

  V = { A, S }, T = { a, b }, and productions

  $S \rightarrow aAb \mid \lambda$

  $A \rightarrow aAb \mid \lambda$

- The grammars in examples 1.11 and 1.14 can be shown to be equivalent

Automata

- An *automaton* is an abstract model of a 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.

Diagram of a General Automaton

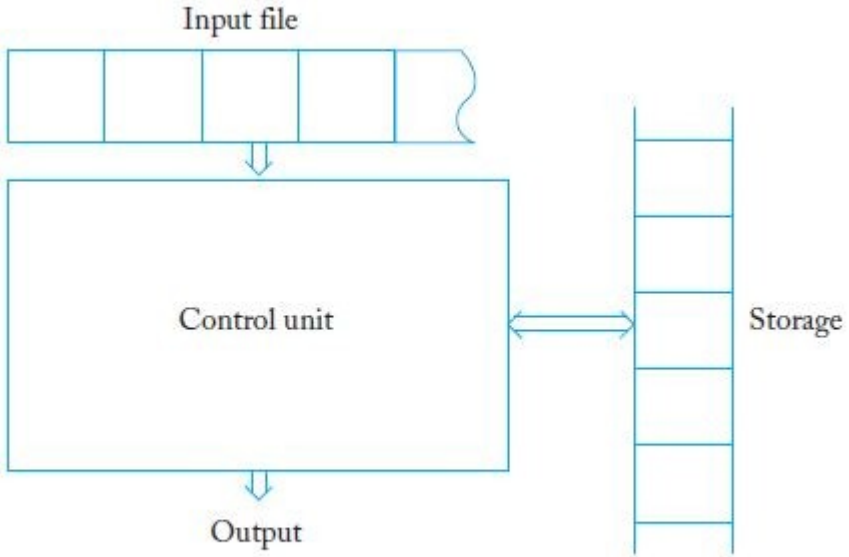


FIGURE 1.4

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