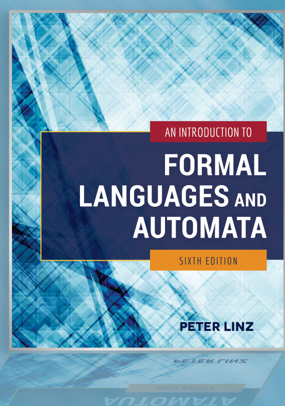
Chapter 11

A HIERARCHY OF FORMAL LANGUAGES AND AUTOMATA



Recursive and Recursively Enumerable Languages

- A language L is *recursively enumerable* if there exists a Turing machine that accepts it (as we have previously stated, rejected strings cause the machine to either not halt or halt in a nonfinal state)
- A language L is *recursive* if there exists a Turing machine that accepts it and is guaranteed to halt on every valid input string
- In other words, a language is recursive if and only if there exists a membership algorithm for it

Unrestricted Grammars

- An *unrestricted grammar* has essentially no restrictions on the form of its productions:
  - Any variables and terminals on the left side, in any order
  - Any variables and terminals on the right side, in any order
  - The only restriction is that λ is not allowed as the left side of a production
- A sample unrestricted grammar has productions

$S \rightarrow S_1B$
$S_1 \rightarrow aS_1b$
$bB \rightarrow bbbB$
$aS_1b \rightarrow aa$
$B \rightarrow \lambda$

Unrestricted Grammars and Recursively Enumerable Languages

- Theorem 11.6: Any language generated by an unrestricted grammar is recursively enumerable
- Theorem 11.7: For every recursively enumerable language L, there exists an unrestricted grammar G that generates L
- These two theorems establish the result that unrestricted grammars generate exactly the family of recursively enumerable languages, the largest family of languages that can be generated or recognized algorithmically

Context-Sensitive Grammars

- In a context-sensitive grammar, the only restriction is that, for any production, length of the right side is at least as large as the length of the left side
- Example 11.2 introduces a sample unrestricted grammar with productions

$S \to abc \mid aAbc$
$Ab \to bA$
$Ac \to Bbcc$
$bB \to Bb$
$aB \to aa \mid aaA$

Characteristics of Context-Sensitive Grammars

- An important characteristic of context-sensitive grammars is that they are **noncontracting**, in the sense that in any derivation, the length of successive sentential forms can never decrease
- These grammars are called context-sensitive because it is possible to specify that variables may only be replaced in certain contexts
- For instance, in the grammar of Example 11.2, variable A can only be replaced if it is followed by either b or c

Context-Sensitive Languages and Linear Bounded Automata

- Theorem 11.8 states that, for every context-sensitive language L not including λ, there is a linear bounded automaton that recognizes L
- Theorem 11.9 states that, if a language L is accepted by a linear bounded automaton M, then there is a context-sensitive grammar that generates L
- These two theorems establish the result that context-sensitive grammars generate exactly the family of languages accepted by linear bounded automata, the context-sensitive languages

The Chomsky Hierarchy

- The linguist Noam Chomsky summarized the relationship between language families by classifying them into four language types, type 0 to type 3
- This classification, which became known as the *Chomsky Hierarchy*, is illustrated in Figure 11.3

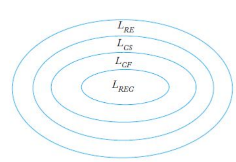


FIGURE 11.3

An Extended Hierarchy

- We have studied additional language families and their relationships to those in the Chomsky Hierarchy
- By including deterministic context-free languages and recursive languages, we obtain the extended hierarchy in Figure 11.4

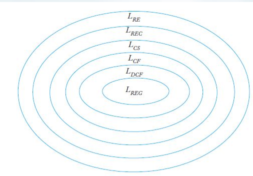
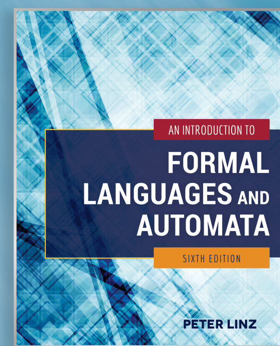


FIGURE 11.4

Chapter 12

LIMITS OF ALGORITHMIC COMPUTATION



Computability and Decidability

- Are there questions which are clearly and precisely stated, yet have no algorithmic solution?
- As stated in chapter 9, a function f is *computable* if there exists a Turing machine that computes the value of f for all arguments in its domain
- Since there may be a Turing machine that can compute f for part of the domain, it is crucial to define the domain of f precisely
- The concept of decidability applies to computations that result in a "yes" or "no" answer: a problem is *decidable* if there exists a Turing machine that gives the correct answer for every instance in the domain

The Turing Machine Halting Problem

- The Turing machine *halting problem* can be stated as: Given the description of a Turing machine M and an input string w, does M perform a computation that eventually halts?
- The domain of the problem is the set of all Turing machines and all input strings w
- Any attempts to simulate the computation on a universal Turing machine face the problem of not knowing if/when M has entered an infinite loop
- By Theorem 12.1, there does not exist any Turing machine that finds the correct answer in all instances; the halting problem is therefore undecidable

The Halting Problem and Recursively Enumerable Languages

- Theorem 12.2 states that, if the halting problem were decidable, then every recursively enumerable language would be recursive
- Assume that L is a recursively enumerable language and M is a Turing machine that accepts L
- If H is a Turing machine that solves the halting problem, then we can apply H to the accepting machine M
  - If H concludes that M does not halt, then by definition the input string is not in L
  - If H concludes that M halts, then M will determine if the input string is in L
- Consequently, we would have a membership algorithm for L, but we know that one does not exist for some recursively enumerable languages, therefore contradicting our assumption that H exists

Reducing One Undecidable Problem to Another

- A problem A is *reduced* to a problem B if the decidability of A follows from the decidability of B
- An example is the *state-entry problem*: given any Turing machine M and string w, decide whether or not the state q is ever entered when M is applied to w
- If we had an algorithm that solves the state-entry problem, it could be used to solve the halting problem
- However, because the halting problem is undecidable, the state-entry problem must also be undecidable

Undecidable Problems for Recursively Enumerable Languages

- As illustrated before, there is no membership algorithm for recursively enumerable languages
- Recursively enumerable languages are so general that most related questions are undecidable
- Usually, there is a way to reduce the halting problem to questions regarding recursively enumerable languages, such as
  - Is the language generated by an unrestricted grammar empty?
  - Is the language accepted by a Turing machine finite?