Ceng 124
Discrete Structures

2018-2019 Spring Semester

by Sibel T. Özyer, Spring 2019

Topics

- 13.1 Languages and Grammars

Introduction

- The grammar of English tells us whether a combination of words is a valid sentence.
- For instance, *the frog writes neatly* is
- a valid sentence, because it is formed from a noun phrase, *the frog*, made up of the article *the* and the noun *frog*, followed by a verb phrase, *writes neatly*, made up of the verb *writes* and the adverb *neatly*.
- We are concerned only with the **syntax**, or form, of the sentence, and not its **semantics**, or meaning.
- *swims quickly mathematics* is not a valid sentence because it does not follow the rules of English grammar.

Introduction (cont.)

- The syntax of a **natural language**, that is, a spoken language, such as English, French, German, or Spanish, is extremely complicated.
- Research in the automatic translation of one language to another has led to the concept of a **formal language**, which, unlike a natural language, is specified by a well-defined set of r
- Rules of syntax are important not only in linguistics, the study of natural languages, but also in the study of programming languages.
- We will describe the sentences of a formal language using a grammar.

Introduction (cont.)

- 1. a **sentence** is made up of a **noun phrase** followed by a **verb phrase**;
- 2. a **noun phrase** is made up of an **article** followed by an **adjective** followed by a **noun**,
- or
- 3. a **noun phrase** is made up of an **article** followed by a **noun**;
- 4. a **verb phrase** is made up of a **verb** followed by an **adverb**, or
- 5. a **verb phrase** is made up of a **verb**;
- 6. an **article** is *a*, or
- 7. an **article** is *the*;
- 8. an **adjective** is *large*, or
- 9. an **adjective** is *hungry*;
- 10. a **noun** is *rabbit*, or
- 11. a **noun** is *mathematician*;
- 12. a **verb** is *eats*, or
- 13. a **verb** is *hops*;
- 14. an **adverb** is *quickly*, or
- 15. an **adverb** is *wildly*.

Introduction (cont.)

- From these rules we can form valid sentences using a series of replacements until no more rules can be used. For instance, we can follow the sequence of replacements:
- **sentence**
- **noun phrase verb phrase**
- **article adjective noun verb phrase**
- **article adjective noun verb adverb**
- *the* **adjective noun verb adverb**
- *the large* **noun verb adverb**
- *the large rabbit* **verb adverb**
- *the large rabbit hops* **adverb**
- *the large rabbit hops quickly*

Phrase-Structure Grammars

A *vocabulary* (or *alphabet*) V is a finite, nonempty set of elements called *symbols*. A *word* (or *sentence*) over V is a string of finite length of elements of V. The *empty string* or *null string*, denoted by λ, is the string containing no symbols. The set of all words over V is denoted by V^*. A *language over* V is a subset of V^*.

Phrase-Structure Grammars (cont.)

- A grammar has a **vocabulary** V , which is a set of symbols used to derive members of the language.
- Some of the elements of the vocabulary cannot be replaced by other symbols. These are called **terminals**, and the other members of the vocabulary, which can be replaced by other symbols, are called **nonterminals**.
- The sets of terminals and nonterminals are usually denoted by T and N, respectively.
- There is a special member of the vocabulary called the **start symbol**, denoted by S, which is the element of the vocabulary that we always begin with.

Phrase-Structure Grammars (cont.)

A *phrase-structure grammar* $G = (V, T, S, P)$ consists of a vocabulary V, a subset T of V consisting of terminal symbols, a start symbol S from V, and a finite set of productions P. The set $V - T$ is denoted by N. Elements of N are called *nonterminal symbols*. Every production in P must contain at least one nonterminal on its left side.

Example 1

- Let $G = (V, T, S, P)$, where $V = \{a, b, A, B, S\}$, $T = \{a, b\}$, S is the start symbol, and $P = \{S \to ABa, A \to BB, B \to ab, AB \to b\}$.
- G is an example of a phrase-structure grammar.

Example 2

- $P = \{S \to ABa,\ A \to BB,\ B \to ab,\ AB \to b\}$.
- The string $Aaba$ is directly derivable from ABa in the grammar in Example 1 because $B \to ab$ is a production in the grammar.
- The string $abababa$ is derivable from ABa because ABa

  $\Rightarrow Aaba \Rightarrow BBaba \Rightarrow Bababa \Rightarrow abababa$, using the productions $B \to ab, A \to BB$,

  $B \to ab$, and $B \to ab$ in succession.

Language Generated by G

Let $G = (V, T, S, P)$ be a phrase-structure grammar. The *language generated by* G (or the *language of* G), denoted by $L(G)$, is the set of all strings of terminals that are derivable from the starting state S. In other words,

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

Example 3

- Let G be the grammar with vocabulary $V = \{S, A, a, b\}$, set of terminals $T = \{a, b\}$, starting symbol S, and productions $P = \{S \rightarrow aA, S \rightarrow b, A \rightarrow aa\}$. What is $L(G)$, the language of this grammar?
- Solution: From the start state S we can derive aA using the production $S \rightarrow aA$. We can also use the production $S \rightarrow b$ to derive b. From aA the production $A \rightarrow aa$ can be used to derive aaa. No additional words can be derived. Hence, $L(G) = \{b, aaa\}$.

Example 4

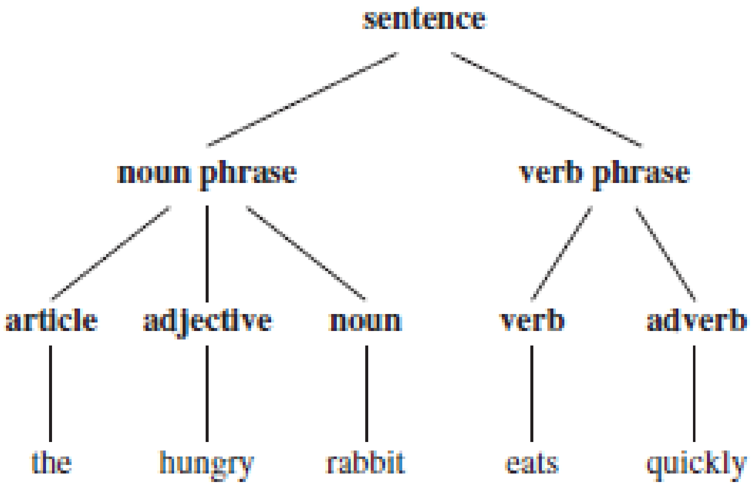
- Let G be the grammar with vocabulary $V = \{S, 0, 1\}$, set of terminals $T = \{0, 1\}$, starting symbol S, and productions $P = \{S \rightarrow 11S, S \rightarrow 0\}$. What is $L(G)$, the language of this grammar?
- Solution: From S we can derive 0 using $S \rightarrow 0$, or $11S$ using $S \rightarrow 11S$. From $11S$ we can derive either 110 or $1111S$. From $1111S$ we can derive 11110 and $111111S$.
- At any stage of a derivation we can either add two 1s at the end of the string or terminate the derivation by adding a 0 at the end of the string. We surmise that $L(G) = \{0, 110, 11110, 1111110, \ldots\}$, the set of all strings that begin with an even number of 1s and end with a 0.

Derivation Trees

- Aderivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a **derivation**, or **parse tree**.
- The root of this tree represents the starting symbol. The internal vertices of the tree represent the nonterminal symbols that arise in the derivation. The leaves of the tree represent the terminal symbols that arise.
- If the production $A \to w$ arises in the derivation, where w is a word, the vertex that represents A has as children vertices that represent each symbol in w, in order from left to right.

A Derivation Tree

Construct a derivation tree for the derivation of *the hungry rabbit eats quickly.*



A Derivation Tree (cont.)

- The problem of determining whether a string is in the language generated by a context-free grammar arises in many applications, such as in the construction of compilers. Two approaches to this problem are indicated in example 5.

Example 5

- Determine whether the word *cbab* belongs to the language generated by the grammar $G = (V, T, S, P)$, where $V = \{a, b, c, A, B, C, S\}$, $T = \{a, b, c\}$, S is the starting symbol, and the productions are
  - $S \rightarrow AB$
  - $A \rightarrow Ca$
  - $B \rightarrow Ba$
  - $B \rightarrow Cb$
  - $B \rightarrow b$
  - $C \rightarrow cb$
  - $C \rightarrow b.$

Solution

- Because there is only one production with S on its left-hand side, we must start with $S \Rightarrow AB$. Next we use the only production that has A on its left-hand side, namely,
- $A \rightarrow Ca$, to obtain $S \Rightarrow AB \Rightarrow CaB$. Because *cbab* begins with the symbols *cb*, we use the production $C \rightarrow cb$. This gives us $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB$. We finish by using the production $B \rightarrow b$, to obtain $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB \Rightarrow cbab$.
- The approach that we have used is called **top-down parsing**, because it begins with the starting symbol and proceeds by successively applying productions.
- There is another approach to this problem, called **bottom-up parsing.** In this approach, we work backward. Because *cbab* is the string to be derived, we can use the production $C \rightarrow cb$, so that $Cab \Rightarrow cbab$. Then, we can use the production $A \rightarrow Ca$, so that $Ab \Rightarrow Cab \Rightarrow cbab$. Using the production $B \rightarrow b$ gives $AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab$. Finally, using $S \rightarrow AB$ shows that a complete derivation for *cbab* is $S \Rightarrow AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab$.