13.1 Languages and Grammar

Formal Language

Formal language is a language that is specified by a well-defined set of rules of syntax.

Formal Grammar

A *formal grammar* G is any compact, precise definition of a language L. A grammar implies an algorithm that would generate all legal sentences of the language.

Phrase-structure Grammar

First, some definitions.

Vocabulary

A *vocabulary* (or *alphabet*) V is a finite, nonempty set of elements called *symbols*.

Word

A *word* (or *sentence*) over V is a string of finite length elements of V.

Empty String λ

The empty string or null string, denoted by λ, is the string containing no symbols.

Set of Words & Set of Language

The set of all words over V is denoted by V^*. A *language* over V is a subset of V^*.

Phrase-Structure Grammar

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.

Derivability

Let $G = (V, T, S, P)$ be a phrase-structure grammar. Let $w_0 = lz_or$ (that is, the concatenation of l, z_o, and r) and $w_1 = lz_1r$ be strings over V. If $z_o \to z_1$ is a production of G, we say that w_1 is *directly derivable* from w_0 and we write $w_0 \Rightarrow w_1$. If $w_0, w_1, \ldots, w_n$ are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots, w_{n-1} \Rightarrow w_n$, then we say that w_n is *derivable from* w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a *derivation*.

Language Generated by G, $L(G)$

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,

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

Types of Grammars

Type	Restrictions on Productions $w_1 \to w_2$
0	No restrictions
1	$w_1 = lAr$ and $w_2 = lwr$, where $A \in N, l, r, w \in (N \cup T)^*$ and $w \neq \lambda$; or $w_1 = S$ and $w_2 = \lambda$ as long as S is not on the right-hand side of another production
2	$w_1 = A$, where A is a nonterminal symbol
3	$w_1 = A$ and $w_2 = aB$ or $w_2 = a$, where $A \in N, B \in N$, and $a \in T$; or $w_1 = S$ and $w_2 = \lambda$

Derivation Trees

A derivation 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.

Backus-Naur Form

The *Backus-Naur form (BNF)* is used to specify the syntactic rules of many computer languages, including Java. The productions in a type 2 grammar have a single nonterminal symbol as their left-hand side. Instead of listing all the productions separately, we can combine all those with the same nonterminal symbol on the left-hand side into one statement. Instead of using the symbol $\to$ in a production, we use the symbol ::=. We enclose all nonterminal symbols in brackets, $\langle\rangle$, and we list all the right-hand sides of productions in the same statement, separating them by bars.

An example of BNF
$\langle identifier \rangle ::= \langle lcletter \rangle | \langle identifier \rangle \langle lcletter \rangle$
$\langle lcletter \rangle ::= a|b|c| \cdots |z$

13.1 pg. 856 # 5

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, 1, A, B, S\}$, $T = \{0, 1\}$, and set of productions P consisting of $S \to 0A, S \to 1A, A \to 0B, B \to 1A, B \to 1$.

a) Show that 10101 belongs to the language generated by G.

$S \Rightarrow 1A \Rightarrow 10B \Rightarrow 101A \Rightarrow 1010B \Rightarrow 10101$

b) Show that 10110 does not belong to the language generated by G.

Notice the two adjacent 1s in the string. By looking at our set of productions, P does not contain any rules that allow two 1s to be adjacent to each other.

c) What is the language generated by G?

By looking at our set of production rules, we can easily see that our string must first start with either 0 or 1 because of $S \to 0A$ and $S \to 1A$. The question now becomes what comes after the first symbol. We first consider the rules $A \to 0B$, and $B \to 1A$. By looking at these rules, we know that the symbols that follow the first symbol will alternate between 0 and 1. So we get $101A$ or $001A$. We also know that our string can only terminate by using the rule $B \to 1$. In addition, we know that each 1 is preceded by a 0. So this means we will have one or more 01's following the first symbol. The language generated by G is $\{0(01)^n | n \geq 1\} \cup \{1(01)^n | n \geq 1\}$.

13.1 pg. 856 # 13

Find a phrase-structure grammar for each of these languages.

a) the set consisting of the bit strings 0, 1, and 11

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, 1, S\}$, $T = \{0, 1\}$, and set of productions P consisting of $S \to 0, S \to 1, S \to 11$.

b) the set of bit strings containing only 1s

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{1, S, A\}$, $T = \{1\}$, and set of productions P consisting of $S \to 1A, A \to 1A, A \to \lambda$.

c) the set of bit strings that start with 0 and end with 1

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, 1, S, A\}$, $T = \{0, 1\}$, and set of productions P consisting of $S \to 0A1, A \to 0A, A \to 1A, A \to \lambda$.

d) the set of bit strings that consist of a 0 followed by an even number of 1s.

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, 1, S, A\}$, $T = \{0, 1\}$, and set of productions P consisting of $S \to 0A, A \to 11A, A \to \lambda$.

13.1 pg. 856 # 17

Construct phrase-structure grammars to generate each of these sets.

a) $\{0^n | n \geq 0\}$

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, S\}$, $T = \{0\}$, and set of productions P consisting of $S \to 0S, S \to \lambda$.

b) $\{1^n 0 | n \geq 0\}$

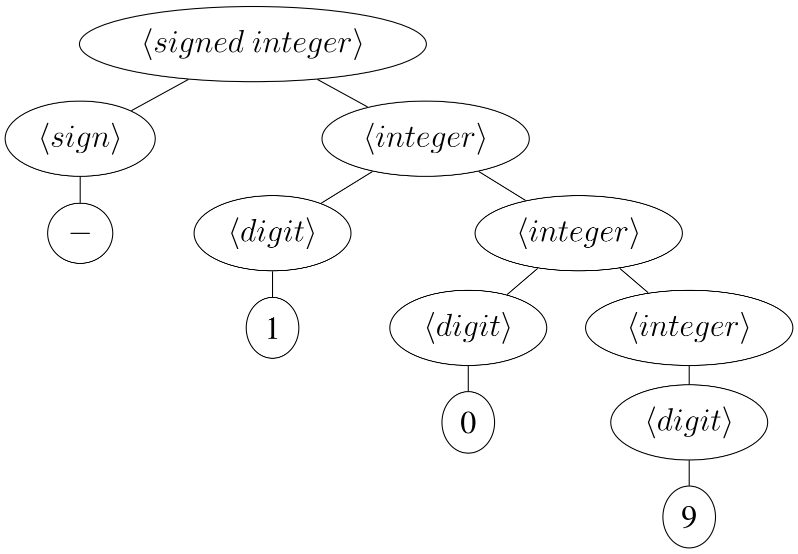
Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, 1, S, A\}$, $T = \{0, 1\}$, and set of productions P consisting of $S \to A0, A \to A1, A \to \lambda$.

c) $\{(000)^n | n \geq 0\}$

Let $G = (V, T, S, P)$ be the phrase-structure grammar with $V = \{0, S\}$, $T = \{0\}$, and set of productions P consisting of $S \to 000S, S \to \lambda$.

13.1 pg. 857 # 27

Construct a derivation tree for -109 using the given grammar.
$\langle signed\, integer \rangle ::= \langle sign \rangle \langle integer \rangle$
$\langle sign \rangle ::= + | -$
$\langle integer \rangle ::= \langle digit \rangle | \langle digit \rangle \langle integer \rangle$
$\langle digit \rangle ::= 0|1|2|3|4|5|6|7|8|9$



13.1 pg. 857 # 31

Give production rules in Backus-Naur form for an identifier if it can consist of

a) one or more lowercase letters.

$\langle identifier \rangle ::= \langle lcletter \rangle | \langle identifier \rangle \langle lcletter \rangle$
$\langle lcletter \rangle ::= a|b|c| \cdots |z$

b) at least three but no more than six lowercase letters.

$\langle identifier \rangle ::= \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle |$
$\langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle |$
$\langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle |$
$\langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle \langle lcletter \rangle$

$\langle lcletter \rangle ::= a|b|c| \cdots |z$

c) one to six uppercase or lowercase letters beginning with an uppercase letter.

$\langle identifier\rangle ::= \langle ucletter\rangle |$
$\langle ucletter\rangle\langle letter\rangle |$
$\langle ucletter\rangle\langle letter\rangle\langle letter\rangle |$
$\langle ucletter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle |$
$\langle ucletter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle |$
$\langle ucletter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle\langle letter\rangle$

$\langle letter\rangle ::= \langle ucletter\rangle | \langle lcletter\rangle$
$\langle ucletter\rangle ::= A|B|C|\cdots|Z$
$\langle lcletter\rangle ::= a|b|c|\cdots|z$

d) a lowercase letter, followed by a digit or an underscore, followed by three or four alphanumeric characters (lower or uppercase letters and digits).

$\langle identifier\rangle ::= \langle lcletter\rangle\langle digitorus\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle |$
$\langle lcletter\rangle\langle digitorus\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle\langle alphanumeric\rangle$

$\langle digitorus\rangle ::= \langle digit\rangle | \_$
$\langle alphanumeric\rangle ::= \langle letter\rangle | \langle digit\rangle$
$\langle letter\rangle ::= \langle ucletter\rangle | \langle lcletter\rangle$
$\langle ucletter\rangle ::= A|B|C|\cdots|Z$
$\langle lcletter\rangle ::= a|b|c|\cdots|z$
$\langle digit\rangle ::= 0|1|2|\cdots|9$