Languages and Grammars (informally)

Suppose we have the following "grammar" to generate sentences in English.

The set of English words we are allowed to used is the set {*a, the, large, hungry, rabbit, mathematician, eats, hops, quickly, wildly*}. Our sentences must satisfy the following rules.

1. A sentence (S) is made up of a noun phrase (NP) followed by a verb phrase (VP);

2. a noun phrase is made up of an article (ART) followed by an adjective (ADJ) followed by a noun (N), 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 (V) followed by and adverb (ADV), or

5. a verb phrase is made up of a verb (V);

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*; and

14. an adverb is *quickly*, or

15. an adverb is *wildly*.

From these rules we can form valid sentences using a series of replacements until no more rules can be used. For example, we can form the sentence *a large mathematician hops wildly.* However, we cannot generate the sentence *the mathematician eats a rabbit*.

Languages and Grammars (formally)

A **vocabulary** V is any nonempty finite set of elements called symbols.
A **string** (word, sentence, statement) over alphabet V is a finite sequence of symbols from V .

$V*$ is the set of all strings over V .

λ (or) is the empty or null string. λ ε V *.

V^+ is the set $V* - \{\lambda\}$.

A <u>language</u> over V is a subset of $V*$.

Informally a **grammar** provides a set of symbols and a set of productions or rules for generating the strings in a language. See our author's description of a grammar.
More formally, a **grammar G** is a 4-tuple (N , T , S , P) where:

1) N is a finite, nonempty set of symbols called **nonterminals**; $N \quad V \qquad N \ T =$
2) T is a finite, nonempty set of symbols called **terminals**; $T \quad V \qquad N \quad T = V$
3) S ε N is a symbol called the **start** symbol; and
4) P is a finite, nonempty set of **productions** or rules of the form $u \ \ v$ where u ε $(N \quad T)^+$ and v ε $(N \quad T)*$. We say u can be defined (or replaced) by v.

For any strings $w_0 = l\,z_0\,r$ and $w_1 = l\,z_1\,r$ ε V^*, we write $w_0 \quad w_1$ if $z_0 \quad z_1$ is a production in P. We say w_1 is directly derivable from w_0 (or derivable from w_0 in one step).

If w_n can be derived from w_0 in zero or more steps we write $w_0 \ {}^* \ w_n$, we say w_n is derivable from w_0 .
We say the grammar G generates the language $L(G)$ provided $L(G) = \{w \ \varepsilon \ T^* : S \ {}^* \ w\}$.

Observe that the symbols (including the symbols in N and in T , and the symbols used to represent the rules, etc.) also make up an alphabet and the rules are strings over that alphabet. Thus the set of rules of the grammar is a **metalanguage** , i.e., a language used to describe another language.

For the time being we shall be concerned with rules of the form $A \quad \beta$ where A ε N and $\beta \ \varepsilon \ V*$. Such grammars are called **context-free grammars** and the languages they generate are called **context-free languages**.

Example: $T = \{a\}$, $N = \{S\}$, $L(G) = \{a^n : n > 0\}$

$P = \{ S \ \ a, S \ \ aS \}$

Example: $T = \{a, b\}$, $N = \{S, B\}$, $L(G) = \{a^r b^s : r, s \_ 0\}$

$P = \{ S \ \ aS,\ S \ \ bB,\ S \ \ b,\ B \ \ bB,\ B \ \ b,\ S \ \ \lambda \}$

Languages and Grammars (cont'd)

Example $T = \{a, b\}$, $N = \{S\}$, L(G) = $\{a^n b^n : n > 0\}$
$P = \{ S \ \ ab,\ S \ \ aSb \}$

Example $T = \{a, b, c\}$, $N = \{S\}$, $L(G) = ?$
$P = \{ S \ \ aSa,\ S \ \ bSb, S \ \ cSc , S \ \ \lambda\}$

Example $T = \{a, b\}$, $N = \{S, A, B\}$, $L(G) = ?$
$P = \{ S \ \ aAB,\ A \ \ bBb,\ B \ \ A,\ B \ \ \lambda \}$

A derivation is a **leftmost** derivation if in each step the leftmost nonterminal is replaced.

Similarly a derivation is a **rightmost** derivation if in each step the rightmost nonterminal is replaced.

Parsing is the process of finding a sequence of rules by which a string u in L(G) is derived.

Informally, a derivation of a string can be represented by a hierarchical structure called a **parse tree**. A parse tree is an ordered tree in which each interior node is labeled with the left hand side of a rule and the children of the node represent the corresponding right hand side of the rule. In a context-free language every interior node is labeled with a nonterminal symbol and every leaf is labeled with a terminal symbol.

A grammar for which two (or more) distinct parse trees are possible for the same string is said to be **ambiguous**. The last example is an example of an ambiguous grammar.
(right) regular grammars are grammars with rules of the form $A \ \ vB$, or $A \ \ v$, where v ε T^* , A,B ε N.
Observe that regular grammars are also context-free.

Example: $T = \{a\}$, $N = \{S\}$, $L(G) = \{a^n : n > 0\}$

$P = \{ S \ \ a, S \ \ aS \}$

Example: $T = \{a, b\}$, $N = \{S, B\}$, $L(G) = \{a^r b^s : r, s \_ 0\}$

$P = \{ S \ \ aS,\ S \ \ bB,\ S \ \ b,\ B \ \ bB,\ B \ \ b,\ S \ \ \lambda \}$

In order that we understand the restrictions placed on context-free grammars it may be helpful to look at a **context-sensitive grammar**. A context-sensitive grammar rule is of the form $\alpha A\beta \ \ \alpha\gamma\beta$, where α, β, γ ε $(NT)^*$, A ε N

Example $T = \{$a, b, c$\}$, $N = \{S, A, B, C\}$, $L(G) = ?$
$P = \{ S \ \ A,\ A \ \ aABC, A \ \ abC , CB \ \ \ BC, bB \ \ \ bb, C \ \ c\}$

Backus-Naur Form

The **syntax** of a language is the form of its statements. In particular, the syntax of a programming language is the form of its expressions, statements, and program units.

The **semantics** of a language is the meaning of those statements. In particular, the semantics of a programming language is the meaning of its expressions, statements, and program units.

The smallest syntactic units in a language are called **lexemes** (or lexics). In particular, the lexemes of a programming language are the instances of its identifiers, constants, operators, keywords, reserved words, etc.

A **token** of a language is a category of its lexemes. In particular, identifiers and arithmetic operators of a programming language are tokens of the programming language.

A language **recognizer** is a device which given a string of symbols from an alphabet either accepts or rejects that string as a member of a language. In particular, the syntax analyzer of a compiler determines whether or not a program is in the programming language, i.e., the program is syntactically correct.

A language **generator** is a device which can generate the sentences in a language. A grammar is a language generator for a language.

The use of a formal grammar to define the syntax of a programming language is important to the user as well as the implementor. The user uses the grammar for correct use of program form, punctuation, and structure. The implementor uses the grammar in construction of the translator. Both the user and implementor have a common definition from which to resolve disputes.

The language generating the instances of the tokens of programming languages is in the class of regular languages. The programming languages themselves are for the most part context-free languages. The notation used to describe the syntax of programming languages is known as the **Backus-Naur form** (**BNF**). BNF is a metalanguage used to describe a programming language.

In Backus-Naur form the nonterminals are the names of abstractions and are often delimited by brackets, < > . For example <var>, <expr>, and <id> may be delimited nonterminals in a BNF description of a programming language. The start symbol may be <program> in a BNF description of a programming language, The terminals are the lexemes of the language. The symbol **::=** is often used instead of the arrow in a production . All right-hand sides of productions with the same left-hand side are listed in the same statement separated by the "or" symbol, **|**. For example, the productions S aSa, S bSb, and S λ are represented as:

<S> **::=** a <S>a **|** b<S>b **|** λ

or in our author's format: *S* a*S*a **|** b*S*b **|** λ.

A grammar for generating arithmetic expressions (see page 36):

expression	*expression operator expression*	
	|	– *expression*
	|	(*expression*)
	|	*identifier*
identifier	A | B | C	
operator		+ | – | * | /

Note that this grammar is an ambiguous grammar.