**Introduction to Grammars and Languages**

**(Class notes prepared by Ramon A. Mata-Toledo)**

**Prerequisites:** Read pages 8 through 33 of your textbook or pages 15 through 18 of the Aho and Ulmann’s book. The list of references is located at the end of this document. PDF files for some of these references are available on Blackboard under the folder References (under Course Documents). All these references have been placed “on reserve” under your instructor’s name at the East Library.

**Alphabet**

We will call an **alphabet** any nonempty finite set of symbols. An alphabet is generally denoted by the Greek letter Σ. The elements of any alphabet are assumed to be indivisible symbols.

**Example No. 1**

The set of letters of the English alphabet = {A, B, ..Z, a, b, c, ..z} form an alphabet.

A widely used alphabet is the binary alphabet Σ = {0, 1}

In certain computer languages such as the family of C languages some elements of the alphabet are the indivisible symbols such as **begin**, **end**, **while**, etc.

**Strings**

Given a particular alphabet we will define a **string** as a sequence of elements taken from that alphabet. Synonyms for a string are word and sentence.

**Example No. 2**

If we assume that Σ is the binary alphabet Σ = {0, 1} we can form the following strings:

0, 1, 00, 01, 10, 001, 010, and 011 (neither the comma nor the word are are part of the string)

Formally, a string can be written as where *k* ≥ 0, where 1 ≤ i ≤ k.

If we let *k* = 0 we have a special string called the empty string which is generally denoted by ε. This empty string is unique and has no symbols. Some properties of this particular string are indicated later on.

Warning.png Do not confound ε with a space or blank character. They are two different strings. In addition, do not confound the empty string with the empty set.

**Example No. 3**

Considering the alphabetwe can form individual string such as

+((a, a\*(a+a), ))+(( a+(a\*a), and (((a+a)))

**Length of a string**

We will define the **length** of a string as the number of characters that make up the string. The length of a string α is generally denoted by |α|. Some authors also use the notation lg(*x*).

**Example No. 4**

Given the alphabet  the length of some of the strings derived from this alphabet are

|a+(a\*a)| = 7 and |(((a+a)))| = 9

The length of the empty string is zero. That is, |ε| = 0

**Two particular sets of strings: and**

Given an alphabet Σ, we can define two new sets, namely,

The set of all strings of length one or more consisting of member of Σ is denoted by Σ+ (generally called “Sigma plus”). Notice that all elements of Σ+ have at least on symbol. Notice that the empty string cannot be an element of this set.

The set of all strings of length zero or more consisting of member of Σ is denoted by Σ\* (generally called “Sigma star” or the Kleene’s star in honor of the mathematician Stephen Kleene who was the first to use this type of notation).

Observe that is ε is an element of the set Σ\*.

The relation between these two particular sets is as follows:

Σ\* = Σ+ {ε}

**Example No. 5**

If we assume that Σ is the binary alphabet Σ = {0, 1} we have that

Σ\* = {ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, ….}

Σ+ = {0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, ….}

The rest of this page has been left intentionally blank

**Concatenation of strings**

Two strings, let’s say, w and x, may be connected to form a new string wx called the **concatenation** of the string w and x. The concatenation is an implicit operation between strings and has no special symbol for it.

Given an alphabet Σ, the following properties hold for the concatenation of symbols:

1. Concatenation is associative; i.e, for each x, y, z in Σ\*

*x*(*yz*) = (*xy*)*z*

1. Given the empty string ε and for each *x*, *y*, *z* in Σ\*
2. ε x = x
3. x ε = x
4. x εy = xy

Notice that the string ε is “invisible” when used in any concatenation operation.

1. Σ\* has left and right cancellation. That is, for each x, y, z in Σ\*
2. zx = zy implies x = y
3. xz = yz implies x = y
4. For each x, y, z in Σ\* we have that

|xy| = |x| + |y|

**Note No. 1**

Mathematically speaking these properties of the concatenation, as they apply to Σ\*, define Σ\* as a monoid where the empty string serves as the identity element. A semigroup consists of a set S with a binary associative operation · which is closed in S. A monoid, in turn, is a semigroup where the properties i and ii given above hold true. An binary operation is said to be closed when, applied to any two elements of a set S, the result of the operation is another element of the set S.

**Substrings, prefixes, and suffixes of a string**

Given a string w over an alphabet Σ, we will call a **substring** of w any part of the string w.

Let x, y, and z be arbitrary strings over some given alphabet Σ. We will say that x is a **prefix** of the string xy.

Likewise, we will say that y is a **suffix** of the string xy.

Any prefix or suffix of a string w is a substring of the string w.

**Example No. 6**

Consider the string xwyz defined over some alphabet Σ.

Some of the prefixes of this string are: x and xw. Likewise, some of the suffixes are y and yz.

Notice that the empty string is a substring, a prefix, and and a suffix of any string regardless of the alphabet.

**Prerequisites for remaining of this set of notes:** Read pages 16 and 17 of reference 1. Read pages 20 through 27 of Section 2 of reference 2.

**Languages**

A language over an alphabet Σ is any subset of Σ\*. Since this definition is too general we will restrict our attention to languages generated by a particular type of phrase-structured grammar.

**Grammars**

Grammars are the most important class of generators of languages. In this particular set of notes and for purposes of compilation we only consider grammars of the type known as phrase-structured grammars.

**Phrase-Structured Grammar**

A **Phrase-Structured grammar** is a mathematical system for defining a language, as well as a device for giving the sentences in the language a useful structure.

A grammar is defined as a 4-tuple  where:

* *N* is a finite nonempty set of **nonterminal symbols**; the symbols of this set are sometimes syntactic categories.
* is a finite nonempty set of **terminal symbols or tokens**; this set is the alphabet of the grammar.
* . That is, *N* and do not have common elements.
* *P* is a finite subset of . The elements of *P* are called **productions rules or productions** for short. Mathematically, productions are ordered pairs of the form (α, β) where and. However, productions are rarely written as ordered pairs, instead they are generally written as  where *α* may contain both terminal and nonterminal symbols and *β* may contain both terminal and nonterminal symbols or it may be the empty (or null) string. Productions define the rule of formation of sentences in the grammar.
* *S* is a distinguished nonterminal symbol called the **starting symbol**.

The term 4-tuple indicates that a phrase-structured grammar has four components. These components do not have to be in any particular order.

**Example No. 7**

Consider the grammar *G*o (pronounced “g not”) which allows us to write any valid arithmetic expressions involving additions and multiplications including parenthesized expressions. This grammar is defined next.

 where *P* consists of the productions which can be written as follows

*E* → *E* + *T*

*E* → *T*

*T* → *T* \* *F*

*T* → *F*

*F* → (*E*)

*F* → *a*

*E* → *E* + *T* | *T*

*T* → *T* \* *F* | *F*

*F* → (*E*) | *a*

or equivalently as

**Example No. 8 (** Example taken from reference No. 3)

Consider the grammar defined by the rules shown below.

1.- <sentence> → <subject> <verb> <noun>

2. -<subject> → <article> <noun>

3. - <object> → <article> <noun>

4. - <verb> → SEES

5. - <article> → THE

6. - <noun> → BOY

7. - <noun> → GIRL

As indicated in Example No. 7 when there is more than one production for the same nonterminal it is customary to substitute all these productions by a single production with the same left-hand side and the right-hand sides separated by a vertical bar. For instance, the productions of the grammar given above

<noun> → BOY

<noun>→ GIRL

can be replaced by the single production

<noun> → BOY | GIRL.

Likewise, the last two productions of Go can be replaced by the production F→ (E) | a.

**Given a set of productions how can we distinguish the elements of a grammar?**

The elements of a grammar are generally no explicitly differentiated as one would expect from the formal definition of grammar and as illustrated in Example No. 7. Instead, grammars are generally defined through their set of productions as indicated in Example No.8.

To help distinguish between nonterminal and terminal elements of the grammars some authors use only Latin-alphabet letters for nonterminals and lower case letters at the beginning of the Latin alphabet for terminals. This convention which may be used for all types of grammar will not be used in this class of notes. Another convention that can be used for all types of grammar is to enclose the nonterminal symbols in a particular set of characters not used in the language for any other purpose. This is illustrated in Example No. 8 where the nonterminal symbols are enclosed in “angled brackets.” All other symbols of Example No. 8, excluding the arrows, are considered terminal symbols. The angled-brackets themselves are not part of the set of nonterminal nor are considered terminal symbols. They are meta-symbols of the language.

In this course, unless indicated otherwise, we will use a convention that will allow us to "extract" from the set of productions all the elements that comprise the grammar when none of the previous two conventions are used. This new convention will be applied to grammars of the type Context Free (this concept will be defined later). The convention is a follows:

The nonterminal symbols or syntactic categories are all the “single symbols” that appear on the left-hand side of the productions of the grammar as shown in Example No. 7, where no brackets of any kind are used, the nonterminal symbols are those symbols that appear in the left hand side of the productions.

Terminal or alphabet symbols are all the individual symbols that appear only on the right-hand side of the productions. Notice that a nonterminal symbols may be made up of more than one character as shown in Example No. 8.

The starting nonterminal is the nonterminal appearing at the left hand side of the topmost production. If there is more than one production that has the same nonterminal on its left hand side, we can choose any of these productions as the starting production.

Using this convention with the grammar of Example No. 7 we can easily identify all the elements of Go.

**Sentential Forms**

A grammar defines a language in a recursive manner. Given a grammar a special kind of string called a **sentential form** can be defined as follows:

1. *S* is a sentential form
2. If *αβγ* is a sentential form and β→δ is in *P*, then *αδγ* is also a sentential form.

A sentential form containing no nonterminal symbols is called a **sentence** generated by *G*.

We will consider a sentential form of *G* as a type of "derivation" which contains, in general, both terminal and nonterminal symbols.

The language generated by the grammar *G*, denoted by L(*G*), is the set of all **sentences** generated by *G*. Remember that a sentence contains no nonterminal symbols.

The “empty” string is a particular type of sentential form.

**How to derive sentences of the language**

Beginning with the starting production, the rules of a grammar are used to define a special kind of string substitution. This substitution is accomplished by replacing a specific nonterminal in some given string of terminal and nonterminal (sentencial form) with the right-hand side of a production which has the specified nonterminal as its left-hand side. We say that the production has been "applied" to the nonterminal in the string. This process of substituting nonterminal symbols by their right-hand side is continued until the sentential form contains terminal symbols only.

When generating or deriving sentences for the language we will consider one of the two methods: linear derivations (leftmost or rightmost) and tree-derivations.

**Linear Derivations**

We will call a **leftmost linear** derivation to any derivation where, at each step, the leftmost nonterminal of the sentential form is substituted.

We will call a **rightmost linear** derivation to any derivation where, at each step, the rightmost nonterminal of the sentential form is substituted.

**Example No. 9**

The leftmost and rightmost derivation of the string “a + a” using the grammar of Example No. 7 are as shown below. The productions used in the derivation are indicated by the numbers “above the arrows”

Leftmost linear: 

Rightmost linear: 

**Derivation Trees**

Any sentence in a language can be associated with a derivation tree. Trees associated with a sentence may be constructed in "top-down" or "bottom-up" fashion. The construction of any derivation tree is carried out by interpreting the production rules as tree-building operations. The derivation tree that we will consider in this section applies only to top-down derivations.

A **derivation tree** *D* (or parse tree) for a context-free grammar can be formally defined as follows:

1. The root of *D* is labeled *S* (the starting nonterminal).
2. If *D*1, *D*2, …*D*k are the subtrees of the direct descendants of the root and the root of *D*i is labeled *X*i, then

*A* → *X*1*X*2…*X*k is a production in *P*. *D*i must be a derivation tree for the grammar if *X*i is a nonterminal and *D*i is a single node labeled *X*i if *X*i is a terminal.

1. Alternatively, if *D*1 is the only subtree of the root of *D* and the root of *D*1 is labeled , then A→ is a production of *P*.

Observe that there is a natural ordering on the nodes of an ordered tree. That is, the direct descendents of a node are ordered "from the left." This ordering can be formally defined as follows:

* Suppose that *n* is a node and *n*1, *n*2, …, *n*k are its direct descendants, then
* if *i* < *j*, then *n*i and all its descendants are to the left of *n*j and all its descendants.

**Note No. 2**

A leftmost linear derivation of a given sentence is "equivalent" to a top-down tree derivation.

Likewise, a bottom-up tree derivation is "equivalent" to a rightmost derivation "in reverse".

Once a derivation tree has been obtained there is not enough information to tell whether the derivation was top-down or bottom-up.

The rest of this page has been left intentionally blank

**Top-down derivation**

The top-down derivation of the string “a+a” is shown below. Notice that a top-down derivation is equivalent to a leftmost derivation. In a top-down derivation you always work with the leftmost node or “oldest child”.

**Example No. 10**

E

+

E

T

F

a

E

E

+

E

T

F

a

E

T

F

a

The rest of this page has been left intentionally blank

**Classification of Grammars**

Grammars are generally classified according to the format of their productions. As indicated before, in this course we will limit ourselves to phrase-structured grammars. The main categories of phrase-structured grammars are: Context-sensitive and context-free. Within the context-free grammars we will consider the following subcategories: right linear, left-linear and regular grammars. These grammars and the languages that they generate are often as the Chomsky hierarchy in honor of N. Chomsky who first considered grammars of this type.

**Context Sensitive grammars**

This is the most general type of all phrase-structured grammars. In this class of grammar, each production is of the form

α →β

The following conditions must hold:

* Both α and β are members of the set 
* α must contain at least one element of *N*. That is the right-hand side of the production may have at least one nonterminal.
* | α| ≤ |β|. That is the length of the left-hand side of the production must be less or equal to the length of the right-hand side. Notice that this implies that in context-sensitive grammar no production can generate the empty string.

**Example No. 11 (**Example taken from reference No. 5)

Consider the following grammar with starting nonterminal S where the sets of terminal and nonterminal symbols are defined as follows:

Σ = {a, b, c} N = {S, B, C}

The set of productions are:

* + - 1. S → aSBC 5. bB → bb
      2. S → aBC 6. bC → bc
      3. CB → BC 7. cC → cc
      4. aB → ab

The derivation of the sentence “aaabbbccc” is left as an exercise to the reader.

The language generated by this grammar contains sentences of the type anbncn for each n ≥1. As this example shows derivation of sentences for this type of grammars may be a complicated process. That is, it is not easy to easily determine what sentences are generated by the language.

The rest of this page has been left intentionally blank

**Context-free grammars**

This type of grammar is important in computer science because all programming languages for which we can always create a compiler fall in this category.

In a context-free grammar all productions are of the following form:

A → α

The following conditions need to be met for each production:

* A is a single nonterminal symbol. That is, A is an element of the set of Nonterminals.
* α is a member of . That is α string of symbols which may contain terminal or nonterminal symbols or it could be the empty string.

The grammars of Examples No. 7 and 8, and repeated here, are two grammars of type context-free.

1.- <sentence> → <subject> <verb> <object> 1. E → E + T

2. -<subject> → <article> <noun> 2. E → T

3. - <object> → <article> <noun> 3. T → T \* F

4. - <verb> → SEES 4. T → F

5. - <article> → THE 5. F → (E)

6. - <noun> → BOY 6. F → a

7. - <noun> → GIRL

**Right-Linear Grammars**

Each production of this type of context-free grammar has one of the following two forms:

A →xB or

A → x

where A and B are elements of N and x is an element of Σ\*.

**Example No. 12** (Example taken from reference No. 1)

The grammar with alphabet Σ = {0, 1} and the productions shown below is right-linear.

S →0S

S →1S

S →ε

This grammar generates the language (0,1}\*.

**Left-Linear Grammars**

Each production of this type of context-free grammar has one of the following two forms:

A →Bx or A → x where A and B are elements of N and x is an element of Σ\*.

**Example No. 13**

The grammar with alphabet Σ = {0, 1} and the productions shown below is left-linear.

S →S0

S →S1

S →ε

**Regular Grammars**

A right-linear grammar is called a **regular grammar** when the following conditions are met:

* All productions with the possible exception of S →ε are of the form A→xB or A→x where A and B are elements of N and x is an element of Σ.
* If S→ε is one of the productions, then S, the starting symbol, does not appear in the right-hand side of any other production.

**Example No. 14** (Example taken from reference No. 2)

The following grammar defines the set of number which contain a decimal point. The letter “d” stands for a decimal digit.

1. S → dB | +A | -A | .G
2. A → dB | .G
3. B → dB | .H | d
4. G → dH
5. H → dH | d

The rest of this page has been left intentionally blank

**Why is the classification important?**

As indicated in reference No. 2 “These grammar classifications are to some extent arbitrary. One may define many variations on the basic patterns given. However, these definitions lead to particularly simple classes of sentence recognizing machines or automata.” By automata we will understand some “machine” which determines if a given string can be derived from the set of productions of the grammar. That is, automata determine can determine if a given string is part of the language generated by the grammar. The process of determining if the given string is in the language is called parsing. These automata are called parsers.

The automata associated with some of the phrase-structured grammar are as follows:

1. The language generated by right-linear grammars can be recognized by finite-state automata (this term will be defined later on) where the “transitions” are associated with some nonterminal symbol.

2. The language generated by a context-free grammar can be recognized by finite-state automata controlled by a stack governed by certain operations. The stack can be indefinitely large. However, only a finite group of “stack members” are ever referenced in the finite description of the automata.

3. The language generated by context-sensitive grammars are accepted by a Turing machine which tape is not allowed to grow longer than the input string. The concept of Turing machine will be defined later on.

**Ambiguous Grammar**

A grammar G is said to be **ambiguous** if there exists at least one string in the language for which there is more than one derivation tree.

The grammar defined by the productions shown below is an ambiguous grammar:

E → E t E

E → x

This can be demonstrated by fact that the string “xtxtx” has two derivations trees. The derivation of these two trees is left as an exercise for the reader.

1. **Exercises**

1.- Let a grammar G be defined by

1. S → AB
2. A → Aa | bB
3. B → a | Sb

Give a derivation tree and a leftmost linear derivation for the sentence baabaab.

2.- Given the grammar

<S> → a <A> c <B>

<S> → <B> d <S>

<B> → a <S> c <A>

<B> → c<A> <B>

<A> → <B> a <B>

<A> → a <B> c

<A> → a

<B> → b

Which of the following strings can be derived from this grammar?

1. aacb
2. aababcbadcd
3. aacbccb
4. aacabcbcccaacdca
5. aacabcbcccaacbca

3.- Given the grammars

1.- <sentence> → <subject> <verb> <object> 1. E → E + T

2. -<subject> → <article> <noun> 2. E → T

3. - <object> → <article> <noun> 3. T → T \* F

4. - <verb> → SEES 4. T → F

5. - <article> → THE 5. F → (E)

6. - <noun> → BOY 6. F → a

7.- <noun> → GIRL

Find the leftmost derivation and top-down derivation of the sentences given below

THE GIRL SEES THE BOY (a + a) \* (a + a)

THE GIRL SEES THE GIRL a\* (a+a)

THE BOY SEES THE BOY (a + a) \* a

4.- Given the grammar

E → E t E

E → x

Find two different derivations trees and two leftmost derivations for the sentence *xtxtx*.

References.gif**References**

The books indicated below were used in the preparation of this class notes. References indicated with an asterisk are considered *classics* in the field of the theory of compilers and language theory. ACM has made references 1 and 5 free to the public.

1. The Theory of Parsing, Translation, and Compiling. Volume I: Parsing by A. V. Aho and J. D. Ullman. Prentice-Hall Inc, 1972. (\*)
2. Compiler Construction: Theory and Practice by W. A. Barrett, R. M. Bates, D. A. Gustafson, and J. D. Couch. 2nd Ed. Science Research Associates, Inc, 1986.
3. Compiler Design Theory by P. M. Lewis II, D. J. Rosenkrantz, and R. E. Stearns. Addison-Wesley Publishing Company, Inc, 1976. (\*)
4. Introduction to Formal Languages Theory by M. A. Harrison. Addison-Wesley Publishing Company, Inc, 1978. (\*)
5. Formal Languages and their relation to Automata by J. E. Hopcroft and J.D. Ullman. Addison-Wesley Publishing Company, Inc, 1969. (\*).