



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Unit 3.

Generative Grammars

Basic concepts of languages theory

- **Parsing or syntactic analysis** is the process of analyzing a **string** of **symbols**, either in natural languages or in computer languages, conforming to the rules of a **formal grammar**.
- The **formal language theory** considers a language as a mathematical object.
- A language is just a set of strings (sentences). To formally define a language we need to formally define what are the strings admitted by the language.

Alphabet

Symbol

A physical entity that we shall not formally define; we shall rely on intuition.

Alphabet

A finite, non-empty set of symbols

- We often use the symbol Σ (sigma) to denote an alphabet
- Examples of alphabet
 - Binary: $\Sigma = \{0,1\}$
 - All lower case letters: $\Sigma = \{a,b,c,..z\}$
 - Alphanumeric: $\Sigma = \{a-z, A-Z, 0-9\}$
 - DNA molecule letters: $\Sigma = \{a,c,g,t\}$ (guanine, adenine, thymine, and cytosine)
 - C character set
 - KPL token set.

C character set

Types	Character Set
Lowercase Letters	a –z
Uppercase Letters	A - Z
Digits	0-9
Special Characters	~! # \$% ^ & *()_ + \ ' - = { } [] : " ; < > ? , . /
White Spaces	Tab Or New line Or Space

Token set of KPL

- Identifiers, numbers, character constants
- Keywords

PROGRAM, CONST, TYPE, VAR, PROCEDURE, FUNCTION, BEGIN, END, ARRAY, OF, INTEGER, CHAR, CALL, IF, ELSE, WHILE, DO, FOR, TO

- Operators

:= (assign), + (addition), - (subtraction), * (multiplication), / (division), = (comparison of equality), != (comparison of difference), > (comparison of greatness), < (comparison of lessness), >= (comparison of greatness or equality), <= (comparison of lessness or equality)

Separators:

::,,(,),,,,(..),,.

String (sentence)

- *A string is finite sequence of symbols chosen from some alphabet*
- Empty string is ε
- Examples of string:
 - 1000010101111
 - A C program is a string of tokens
 - A human DNA pattern

```
GGTGTGGGGACAGGGGTGTGGGGACAGGGGTCTGGGGACAGGGGTGTGGG
GACAGGGGTCTGGGGACAGGGGTGTGGGGATAGGGGTGTGGGGACAGGG
GTGTGGGGACAGGGGTGTGGGGACAGGGGTCTGGGGACAGCAGCGCAAAG
AGCCCCGCCCTGCAGCCTCCAGCTCTCCTGGTCTAATGTGGAAAGTGGCC
CAGGTGAGGGCTTTGCTCTCCTGGAGACATTTGCCCCAGCTGTGAGCAG
GGACAGGTCTGGCCACCGGGCCCTGGTTAAGACTCTAATGACCCGCTGG
TCCTGAGGAAGAGGTGCTGACGACCAAGGAGATCTTCCACAGACCCAGC
ACCAGGGAAATGGTCCGAAATTGCAGCCTCAGCCCCAGCCATCTGCCG
ACCCCCCAGCCAGGCCCTAATGGGCCAGGCGGCAGGGGTTGAGAGGTA
GGGGAGATGGGCTCTGAGACTATAAAGCCAGCGGGGGCCAGCAGCCCTC
```

Languages

A language over alphabet Σ is a set of strings over Σ

Examples of languages:

- The set of all words over $\{a, b\}$,
- The set $\{ a^n \mid n \text{ is a prime number} \}$,
- Programming language C: the set of syntactically correct programs in C

Chomsky's Hierarchy

- Type-0 languages (recursive enumerable)
instances of a problem.
- Type-1 languages (context-sensitive)
natural languages, DNA languages
- Type-2 languages (context-free)
programming language, natural languages
- Type-3 languages (regular)
tokens of programming languages

A grammar to generate real numbers

$\langle \text{real number} \rangle ::= \langle \text{sign} \rangle \langle \text{natural number} \rangle \mid$
 $\quad \langle \text{sign} \rangle \langle \text{natural number} \rangle '.' \langle \text{digit sequence} \rangle \mid$
 $\langle \text{sign} \rangle '.' \langle \text{digit} \rangle \langle \text{digit sequence} \rangle \mid$
 $\langle \text{sign} \rangle \langle \text{real number} \rangle 'e' \langle \text{natural number} \rangle$
 $\langle \text{sign} \rangle ::= ' ' \mid '+' \mid '-'$
 $\langle \text{natural number} \rangle ::= '0' \mid \langle \text{nonzero digit} \rangle \langle \text{digit sequence} \rangle$
 $\langle \text{nonzero digit} \rangle ::= '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9'$
 $\langle \text{digit sequence} \rangle ::= ' ' \mid \langle \text{digit} \rangle \langle \text{digit sequence} \rangle$
 $\langle \text{digit} \rangle ::= '0' \mid '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9'$

How a string of a context free language can be generate ?

A **context free grammar** can be used to generate strings in the corresponding language as follows:

let X = the start symbol s

while there is some nonterminal Y in X do

apply any one production rule using Y , e.g.

$Y \rightarrow w$

Context Free Grammars (CFG)

A context free grammar G has:

- A set of terminal symbols, Σ
- A set of nonterminal symbols, Δ
- A start symbol, S , which is a member of Δ
- A set P of production rules of the form $A \rightarrow w$, where A is a nonterminal and w is a string of terminal and nonterminal symbols or ϵ .

Parse Tree

$S \rightarrow NP VP$

$NP \rightarrow D N$

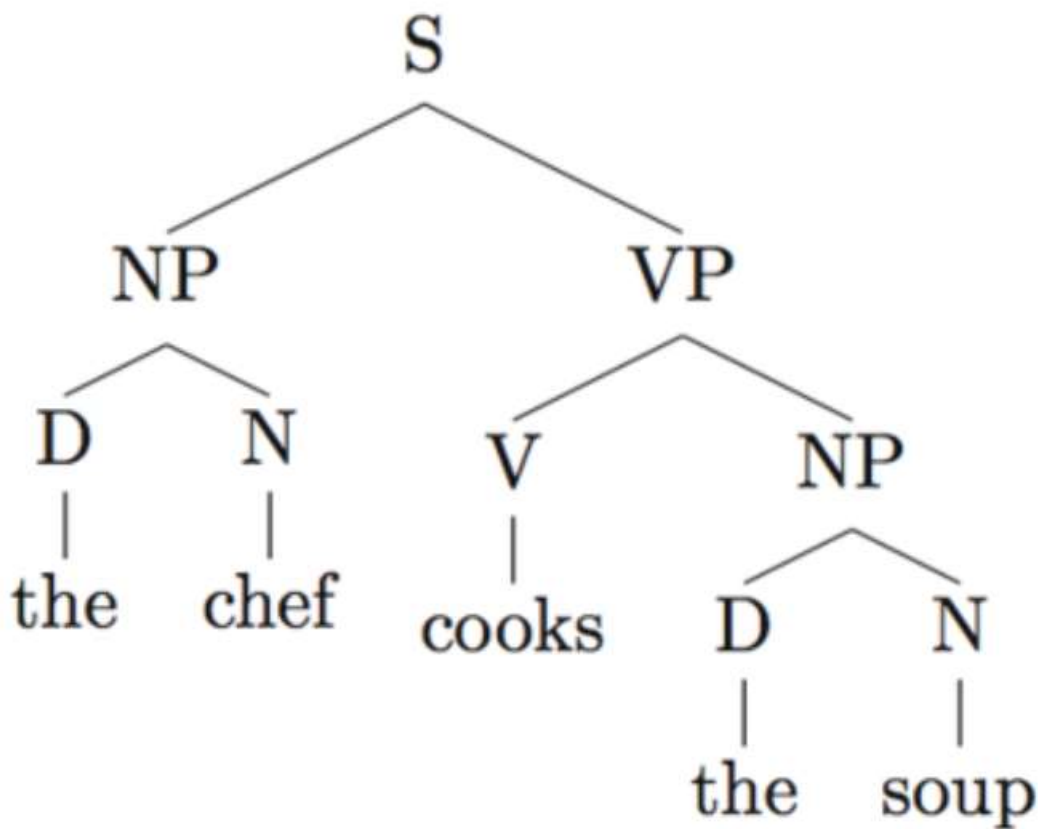
$VP \rightarrow V NP$

$D \rightarrow \text{the}$

$N \rightarrow \text{chef}$

$N \rightarrow \text{soup}$

$V \rightarrow \text{cooks}$



Context Free Grammar Examples

- Grammar of nested parentheses

$G = (\Sigma, \Delta, P, S)$ where

$$\Delta = \{S\}$$

$$\Sigma = \{ (,) \}$$

$$P = \{ S \rightarrow (S), S \rightarrow SS, S \rightarrow \epsilon \}$$

Context Free Grammar Examples

- The grammar of decimal numbers

$$S \rightarrow +A \mid -A \mid A$$
$$A \rightarrow B.B \mid B$$
$$B \rightarrow BC \mid C$$
$$C \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$$

Derivations

- When X consists only of terminal symbols, it is a string of the language denoted by the grammar.
- Each iteration of the loop is a derivation step.
- If an iteration has several nonterminals to choose from at some point, the rules of derivation would allow any of these to be applied.
- Example : $S \Rightarrow -A \Rightarrow -B.B \Rightarrow -B.C \Rightarrow -C.C \Rightarrow -1.C \Rightarrow -1.5$

Leftmost and Rightmost Derivations

- In practice, parsing algorithms tend to always choose the leftmost nonterminal, or the rightmost nonterminal, resulting in strings that are **leftmost derivations** or **rightmost derivations**
- Example:

Leftmost derivation:

$$S \Rightarrow -A \Rightarrow -B.B \Rightarrow -C.B \Rightarrow -1.B \Rightarrow -1.C \Rightarrow -1.5$$

Rightmost derivation:

$$S \Rightarrow -A \Rightarrow -B.B \Rightarrow -B.C \Rightarrow -B.5 \Rightarrow -C.5 \Rightarrow -1.5$$

Ambiguity

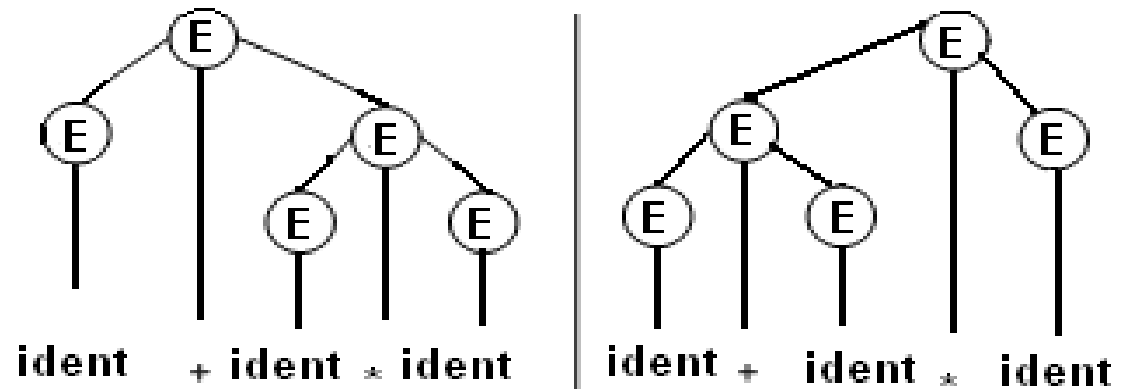
Grammar

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow \text{ident}$



allows two different derivations for strings
such as $\text{ident} + \text{ident} * \text{ident}$ (e.g. $x + y * z$)



The grammar is ambiguous

Disambiguation

$E \rightarrow E + T$

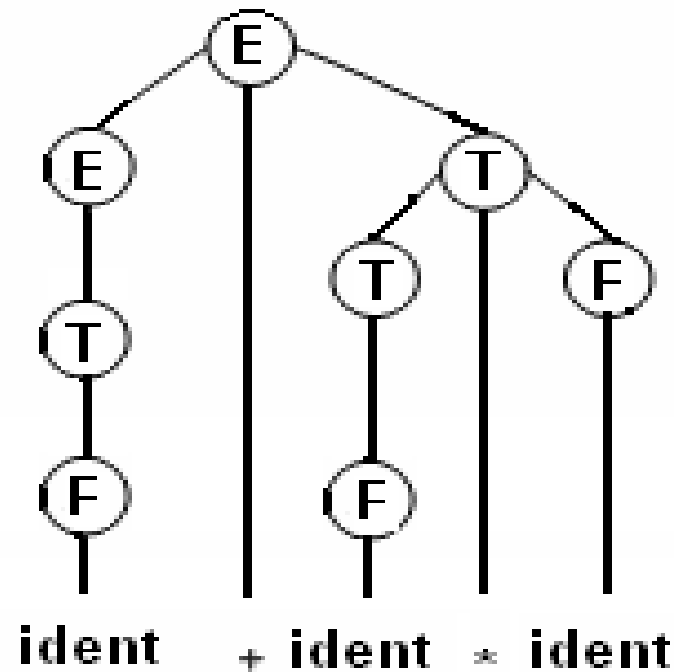
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow \text{ident}$



(by adding some nonterminals and production rules to force operator precedence)

Recursion

- A production is recursive if $X \Rightarrow^* \omega_1 X \omega_2$ Can be used to represent repetitions and nested structures

Direct recursion $X \Rightarrow \omega_1 X \omega_2$

Left recursion $X \rightarrow b \mid \textcolor{red}{X}a. X \Rightarrow Xa \Rightarrow Xaa \Rightarrow Xaaa \Rightarrow baaaaa \dots$

Right recursion $X \rightarrow b \mid a \textcolor{red}{X}. X \Rightarrow aX \Rightarrow aaX \Rightarrow aaaX \Rightarrow \dots aaaaaa b$

Central recursion $X \rightarrow b \mid (\textcolor{red}{X}). X \Rightarrow (X) \Rightarrow ((X)) \Rightarrow (((X))) \Rightarrow (((\dots (b) \dots)))$

Indirect recursion $X \Rightarrow^* \omega_1 X \omega_2$

Example

Removing Left Recursion

Let the left-recursive productions in which A occurs as lhs be

$$A \rightarrow A\alpha_1$$

.....

$$A \rightarrow A\alpha_r$$

and the remaining productions in which A occurs as lhs be

$$A \rightarrow \beta_1$$

.....

$$A \rightarrow \beta_s$$

Removing Left Recursion

Let K_A denote a symbol which does not already occur in the grammar.

Replace the above productions by:

$$A \rightarrow \beta_1 K_A \mid \dots \mid \beta_s K_A$$

$$K_A \rightarrow \varepsilon \mid \alpha_1 K_A \mid \dots \mid \alpha_r K_A$$

Clearly the grammar G' produced is equivalent to G .