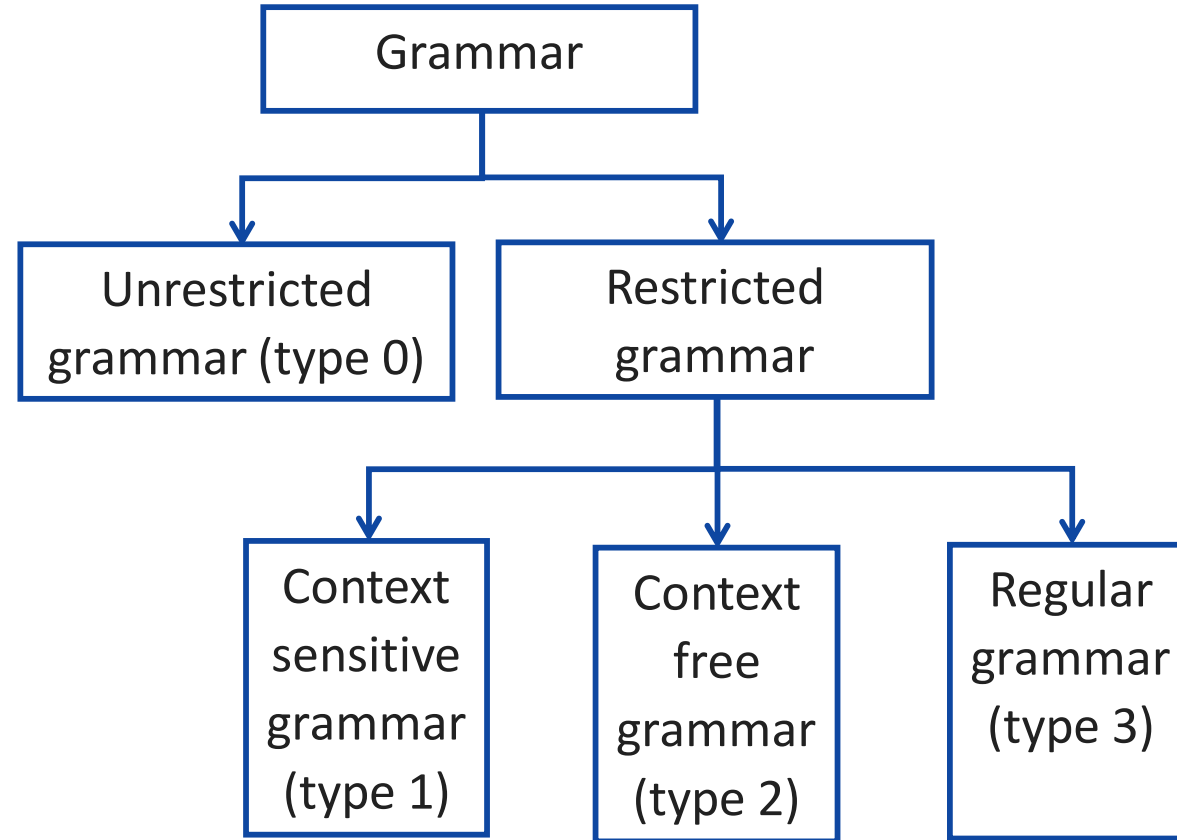


Unit – 4

Context Free Grammar

Chomsky Hierarchy

Chomsky hierarchy (Classification of grammar)



Type 0 grammar (Phrase Structure Grammar)

- ▶ Their productions are of the form:

$$\alpha \rightarrow \beta$$

- ▶ where both α and β can be strings of terminal and nonterminal symbols.

- ▶ Example: $S \rightarrow ACaB$

$$Bc \rightarrow acB$$

$$CB \rightarrow DB$$

$$aD \rightarrow Db$$

Type 1 grammar (Context Sensitive Grammar)

- ▶ Their productions are of the form:

$$\alpha A \beta \rightarrow \alpha \pi \beta$$

- ▶ where **A** is non terminal and **α, β, π** are strings of terminals and non terminals.
- ▶ The strings α and β may be empty, but π must be non-empty.
- ▶ Here, a string π can be replaced by 'A' (or vice versa) only when it is enclosed by the strings α and β in a sentential form.
- ▶ Example: $AB \rightarrow AbBc$
 $A \rightarrow bcA$
 $B \rightarrow b$

Type 2 grammar (Context Free Grammar)

- ▶ Their productions are of the form:

$$A \rightarrow \alpha$$

- ▶ Where **A** is non terminal and **α** is string of terminals and non terminals.

- ▶ Example: $S \rightarrow Xa$

$$X \rightarrow a$$

$$X \rightarrow aX$$

$$X \rightarrow abc$$

Type 3 grammar (Linear or Regular grammar)

- ▶ Their productions are of the form:

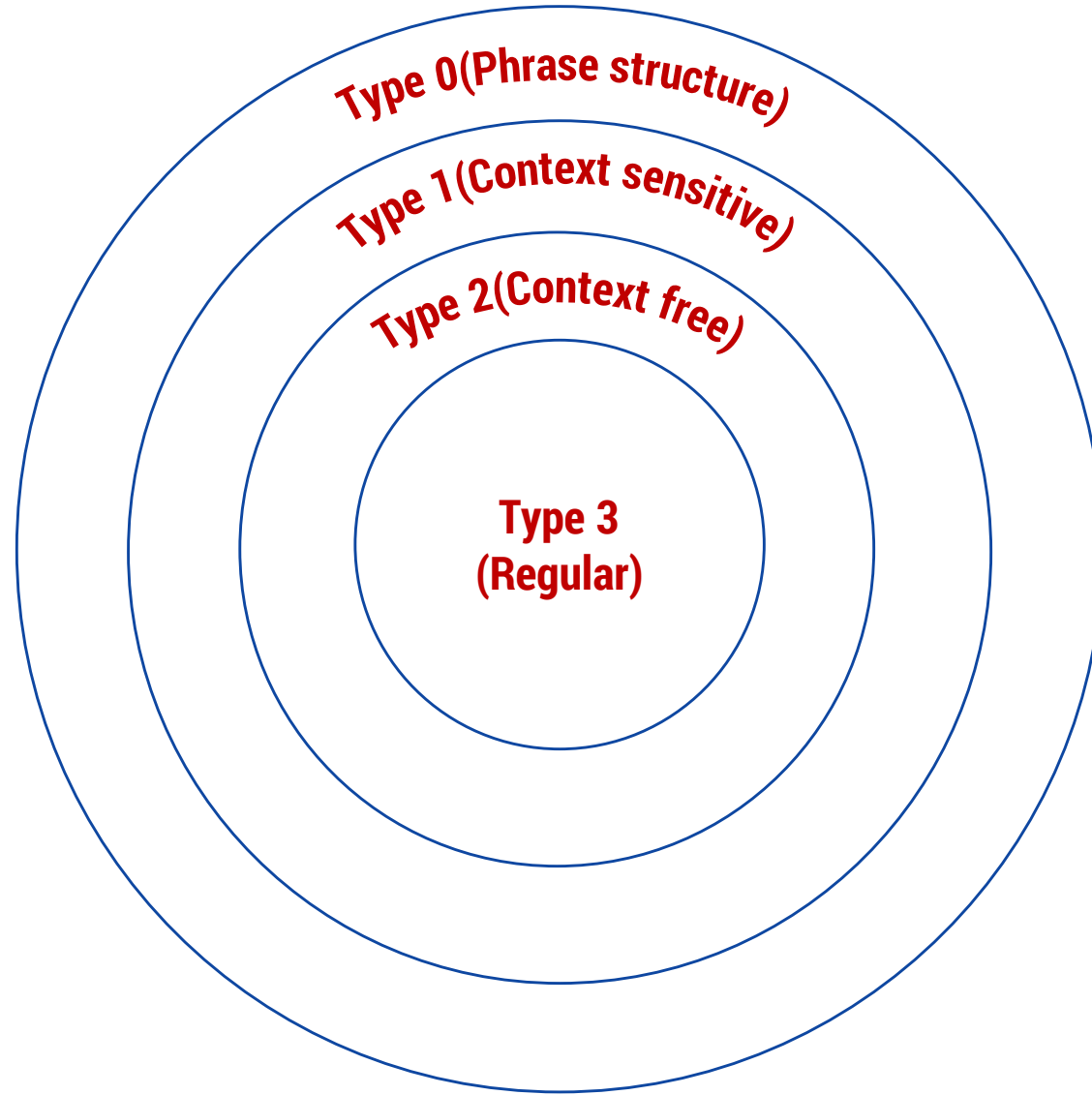
$$A \rightarrow tB \mid t \quad \text{or} \quad A \rightarrow Bt \mid t$$

- ▶ Where A, B are non terminals and t is terminal.

- ▶ Example: $X \rightarrow a \mid aY$

$$Y \rightarrow b$$

Hierarchy of grammar



Context Free Grammar

Context Free Grammar

- ▶ A context free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$ where,
 - V is finite set of **non terminals**,
 - Σ is disjoint finite set of **terminals**,
 - S is an element of V and it's a **start symbol**,
 - P is a finite **set of productions** of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.
- ▶ **Application of CFG:**
 1. CFG are extensively used to specify the **syntax of programming language**.
 2. CFG is **used to develop a parser**.

Context Free Language

- ▶ Let $G = (V, \Sigma, S, P)$ be a CFG. The language generated by G is

$$L(G) : \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$$

- ▶ A language L is a **context free Language** (CFL) if there is a CFG G so that $L = L(G)$.

Context free grammar

- ▶ A context free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$ where,
 - V is finite set of non terminals,
 - Σ is disjoint finite set of terminals,
 - S is an element of V and it's a start symbol,
 - P is a finite set formulas of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
-

- ▶ **Nonterminal symbol:**

- ↳ The name of syntax category of a language, e.g., noun, verb, etc.
- ↳ The It is written as a **single capital letter**, or as a **name enclosed between < ... >**, e.g., A or $\langle \text{Noun} \rangle$

$\langle \text{Noun Phrase} \rangle \rightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle$

$\langle \text{Article} \rangle \rightarrow a \mid an \mid the$

$\langle \text{Noun} \rangle \rightarrow boy \mid apple$

Context free grammar

- ▶ A context free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$ where,
 - V is finite set of non terminals,
 - Σ is disjoint finite set of terminals,
 - S is an element of V and it's a start symbol,
 - P is a finite set formulas of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
-

- ▶ **Terminal symbol:**

- ↳ A symbol in the alphabet.
- ↳ It is denoted by lower case letter and punctuation marks used in language.

$\langle \text{Noun Phrase} \rangle \rightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle$

$\langle \text{Article} \rangle \rightarrow a \mid an \mid the$

$\langle \text{Noun} \rangle \rightarrow boy \mid apple$

Context free grammar

- ▶ A context free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$ where,
 - V is finite set of non terminals,
 - Σ is disjoint finite set of terminals,
 - S is an element of V and it's a start symbol,
 - P is a finite set formulas of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
-

- ▶ **Start symbol:**

- ↪ First nonterminal symbol of the grammar is called start symbol.

<Noun Phrase> \rightarrow <Article><Noun>

<Article> \rightarrow a | an | the

<Noun> \rightarrow boy | apple

Context free grammar

- ▶ A context free grammar (CFG) is a 4-tuple $G = (V, \Sigma, S, P)$ where,
 - V is finite set of non terminals,
 - Σ is disjoint finite set of terminals,
 - S is an element of V and it's a start symbol,
 - P is a finite set formulas of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
-
- ▶ **Production:**
 - ↳ A production, also called a rewriting rule, is a rule of grammar. It has the form of
A nonterminal symbol \rightarrow String of terminal and nonterminal symbols

<Noun Phrase> \rightarrow <Article><Noun>

<Article> \rightarrow a | an | the

<Noun> \rightarrow boy | apple

Example: Grammar

Write terminals, non terminals, start symbol, and productions for following grammar.

$$E \rightarrow E \ 0 \ E \mid (E) \mid -E \mid \text{id}$$
$$0 \rightarrow + \mid - \mid * \mid / \mid \uparrow$$

Terminals: **id + - * / \uparrow ()**

Non terminals: **E, 0**

Start symbol: **E**

Productions: $E \rightarrow E \ 0 \ E \mid (E) \mid -E \mid \text{id}$
 $0 \rightarrow + \mid - \mid * \mid / \mid \uparrow$

CFG Examples

- ▶ Write CFG for ab^*

$$S \rightarrow aX$$
$$X \rightarrow \wedge \mid bX$$

- ▶ Write CFG for a^*b^*

$$S \rightarrow XY$$
$$X \rightarrow aX \mid \wedge$$
$$Y \rightarrow bY \mid \wedge$$

- ▶ Write CFG for $(a+b)^*$

$$S \rightarrow aS \mid bS \mid \wedge$$

- ▶ Write CFG for $a(a+b)^*$

$$S \rightarrow aX$$
$$X \rightarrow aX \mid bX \mid \wedge$$

CFG Examples

- ▶ Write CFG for either a or b

$$S \rightarrow a \mid b$$

- ▶ Write CFG for a^+

$$S \rightarrow aS \mid a$$

- ▶ Write CFG for a^*

$$S \rightarrow aS \mid \wedge$$

- ▶ Write CFG for $(ab)^*$

$$S \rightarrow abS \mid \wedge$$

- ▶ Write CFG for any string of a and b

$$S \rightarrow aS \mid bS \mid a \mid b$$

CFG Examples

- ▶ Write CFG for $a^* \mid b^*$

$$S \rightarrow A \mid B$$
$$A \rightarrow \wedge \mid aA$$
$$B \rightarrow \wedge \mid bB$$

- ▶ Write CFG for $(011+1)^*(01)^*$

$$S \rightarrow AB$$
$$A \rightarrow 011A \mid 1A \mid \wedge$$
$$B \rightarrow 01B \mid \wedge$$

- ▶ Write CFG for balanced parenthesis

$$S \rightarrow [] \mid \{\} \mid [s] \mid \{s\} \mid \wedge$$

CFG Examples

- ▶ Write CFG which contains at least three times 1.

$$S \rightarrow A1A1A1A$$
$$A \rightarrow 0A \mid 1A \mid \wedge$$

- ▶ Write CFG that must start and end with same symbol.

$$S \rightarrow 0A0 \mid 1A1$$
$$A \rightarrow 0A \mid 1A \mid \wedge$$

- ▶ The language of even & odd length palindrome string over {a,b}

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \wedge$$

- ▶ No. of a and no. of b are same

$$S \rightarrow aSb \mid bSa \mid \wedge$$

- ▶ The language of {a, b} ends in a

$$S \rightarrow aS \mid bS \mid a$$

CFG Examples

- ▶ Write CFG for regular expression $(a+b)^*a(a+b)^*a(a+b)^*$

$S \rightarrow XaXaX$

$X \rightarrow aX | bX | \wedge$

- ▶ Write CFG for number of 0's and 1's are same ($n_0(x)=n_1(x)$)

$S \rightarrow 0S1 \mid 1S0 \mid \wedge$

- ▶ Write CFG for $L = \{a^i b^j c^k \mid i=j \text{ or } j=k\}$

For $i=j$

$S \rightarrow AB$

$A \rightarrow aAb \mid ab$

$B \rightarrow cB \mid c$

for $j=k$

$S \rightarrow CD$

$C \rightarrow aC \mid a$

$D \rightarrow bDc \mid bc$

CFG Examples

- ▶ Write CFG for $L = \{ a^i b^j c^k \mid j > i+k \}$

$S \rightarrow ABC$

$A \rightarrow aAb \mid \wedge$

$B \rightarrow bB \mid b$

$C \rightarrow bCc \mid \wedge$

- ▶ Write CFG for $L = \{ 0^i 1^j 0^k \mid j > i+k \}$

$S \rightarrow ABC$

$A \rightarrow 0A1 \mid \wedge$

$B \rightarrow 1B \mid 1$

$C \rightarrow 1C0 \mid \wedge$

Recursive Definitions

Recursive Definitions

1. Recursive Definition of $\{a,b\}^*$

CFG: $S \rightarrow aS \mid bS \mid \Lambda$

$\Lambda \in L$

For any $S \in L$, $aS \in L$

For any $S \in L$, $bS \in L$

No other strings are in L

2. Recursive Definition of Palindrome

CFG: $S \rightarrow aSa \mid bSb \mid a \mid b \mid \Lambda$

$\Lambda, a, b \in L$

For any $S \in L$, $aSa \in L$ and $bSb \in L$

No other string are in L

3. Recursive Definition of the language $\{a^n b^n \mid n \geq 0\}$

CFG: $S \rightarrow aSb \mid \Lambda$

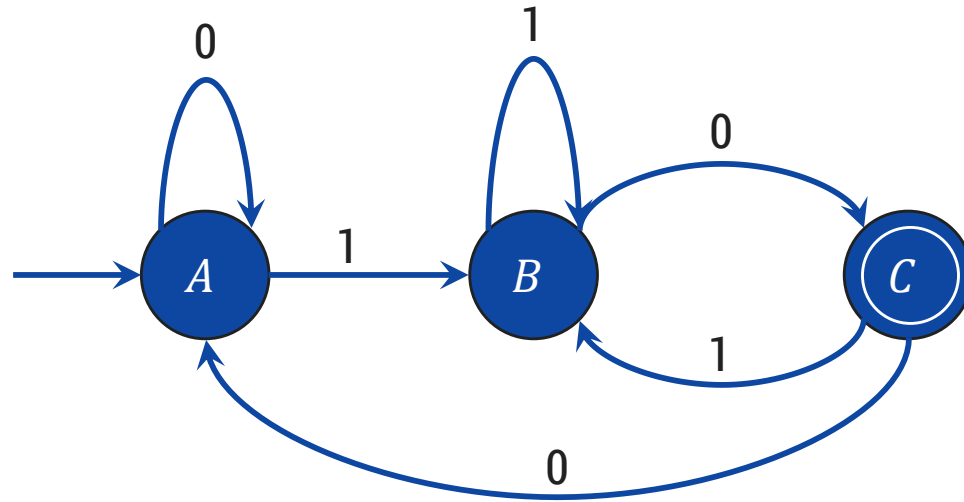
$\Lambda \in L$

For every $S \in L$, $aSb \in L$

No other strings are in L

FA to Regular Grammar

FA to Regular Grammar



$A \rightarrow 0A$

$A \rightarrow 1B$

$B \rightarrow 0C$

$B \rightarrow 1B$

$C \rightarrow 0A$

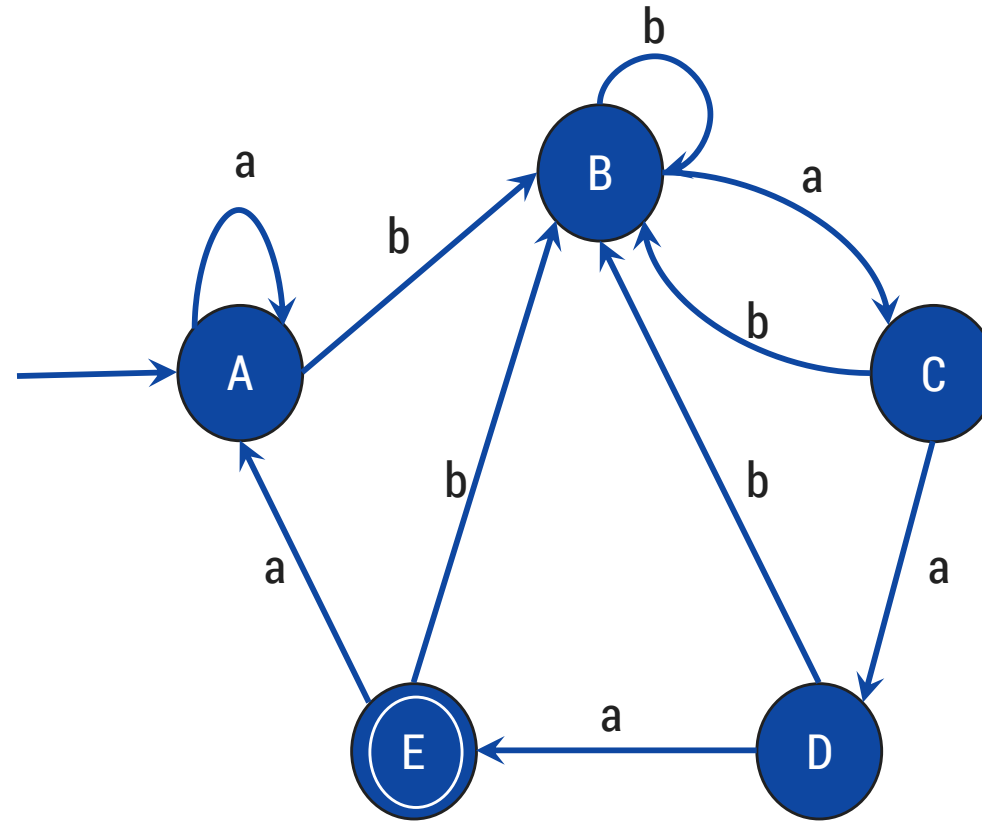
$C \rightarrow 1B$

$B \rightarrow 0$

At last, all the incoming transitions to the accepting states are designated by the production

Source State \rightarrow input symbol

Exercise: FA to Regular Grammar



Derivation

Derivation

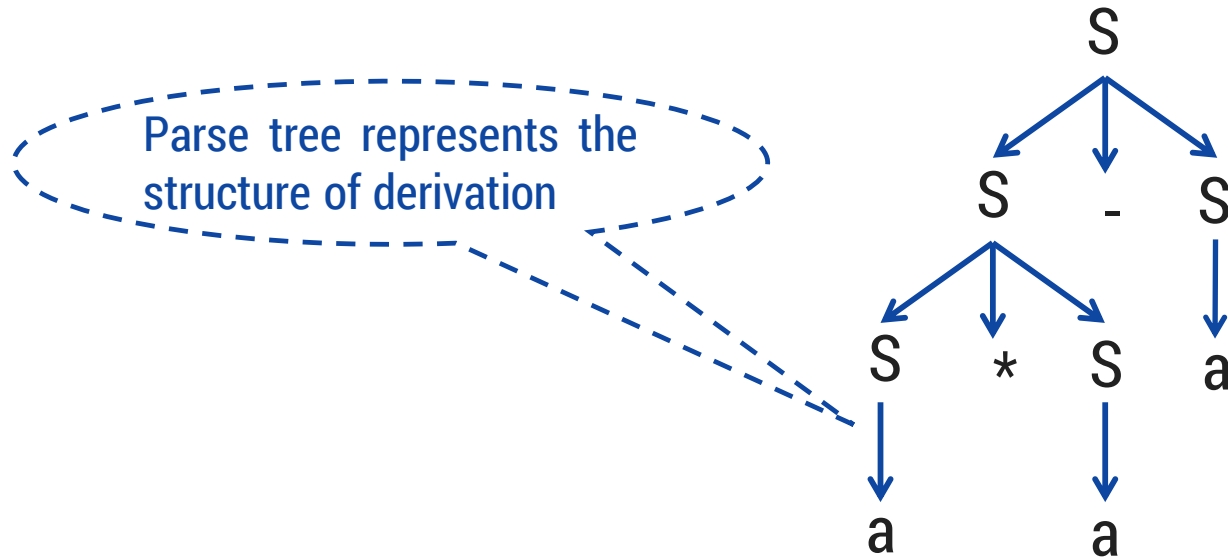
- ▶ Derivation is used to find whether the string belongs to a given grammar or not.
- ▶ Types of derivations are:
 1. Leftmost derivation
 2. Rightmost derivation

Leftmost derivation

- ▶ A derivation of a string W in a grammar G is a left most derivation if at every step the **left most non terminal** is replaced.
- ▶ Grammar: $S \rightarrow S+S \mid S-S \mid S*S \mid S/S \mid a$ Output string: $a*a-a$

S
 $\rightarrow \underline{S}-S$
 $\rightarrow \underline{S}*S-S$
 $\rightarrow a*\underline{S}-S$
 $\rightarrow a*a-\underline{S}$
 $\rightarrow a*a-a$

Leftmost Derivation



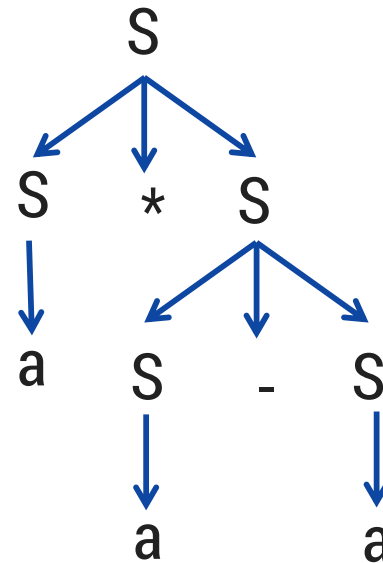
Parse tree

Rightmost derivation

- ▶ A derivation of a string W in a grammar G is a right most derivation if at every step the right most non terminal is replaced.
- ▶ It is all called canonical derivation.
- ▶ Grammar: $S \rightarrow S+S \mid S-S \mid S*S \mid S/S \mid a$ Output string: $a*a-a$

S
 $\rightarrow S*\underline{S}$
 $\rightarrow S*S-\underline{S}$
 $\rightarrow S*S-\underline{a}$
 $\rightarrow \underline{S}^*a-a$
 $\rightarrow a*a-a$

Rightmost Derivation



Parse Tree

Exercise: Derivation

1. Perform leftmost derivation and draw parse tree.

$S \rightarrow A1B$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 0B \mid 1B \mid \epsilon$

Output string: 1001

2. Perform leftmost derivation and draw parse tree.

$S \rightarrow 0S1 \mid 01$ **Output string: 000111**

3. Perform rightmost derivation and draw parse tree.

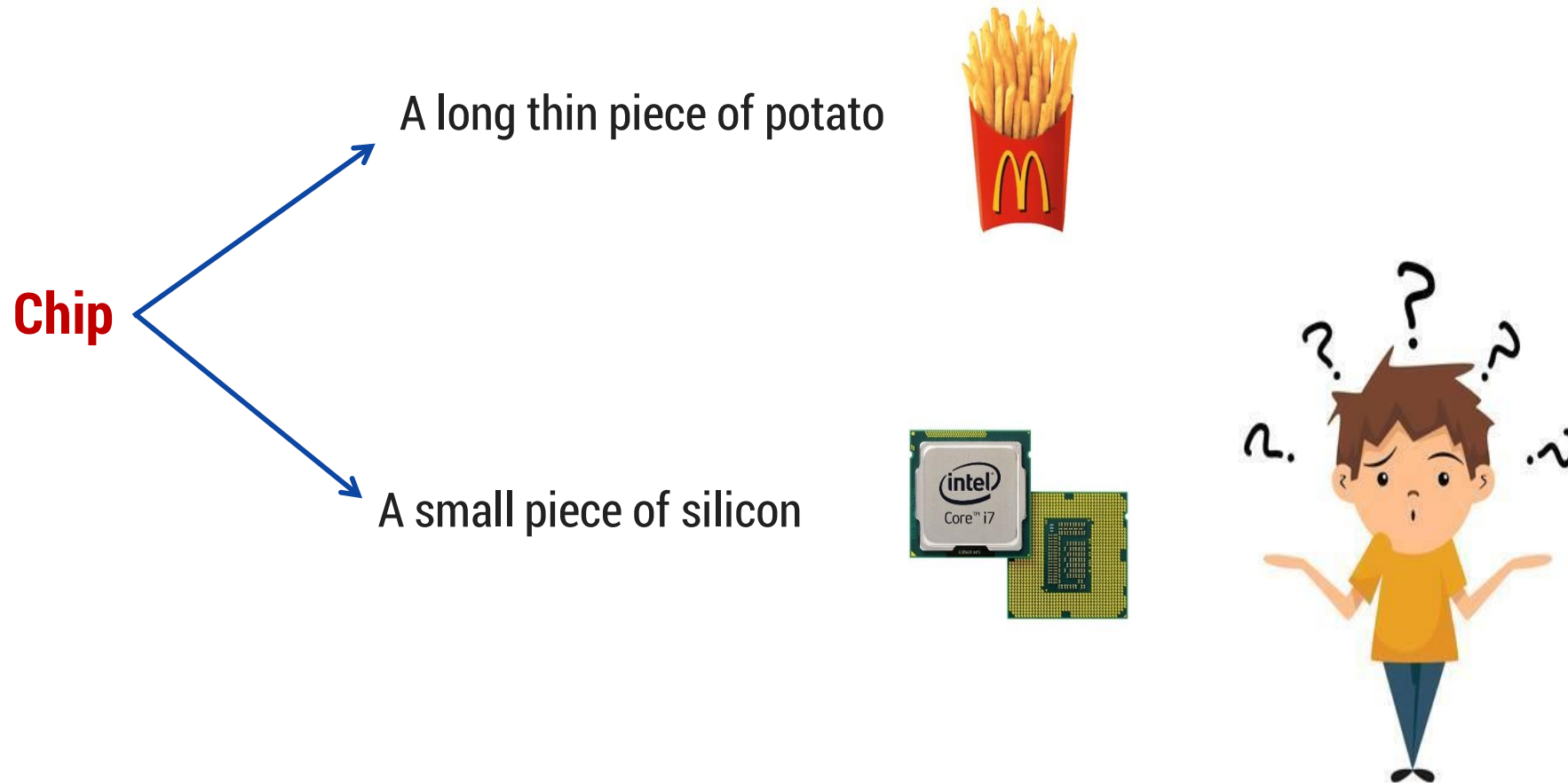
$E \rightarrow E+E \mid E * E \mid \text{id} \mid (E) \mid -E$

Output string: id + id * id

Ambiguous grammar

Ambiguity

- Ambiguity, is a word, phrase, or statement which contains **more than one meaning**.



Ambiguity

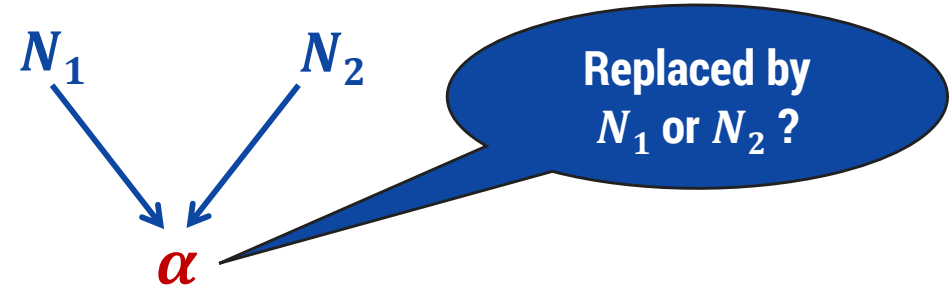
- ▶ In formal language grammar, ambiguity would arise if identical string can occur on the RHS of two or more productions.

- ▶ Grammar:

$$N1 \rightarrow \alpha$$

$$N2 \rightarrow \alpha$$

- ▶ α can be derived from either $N1$ or $N2$



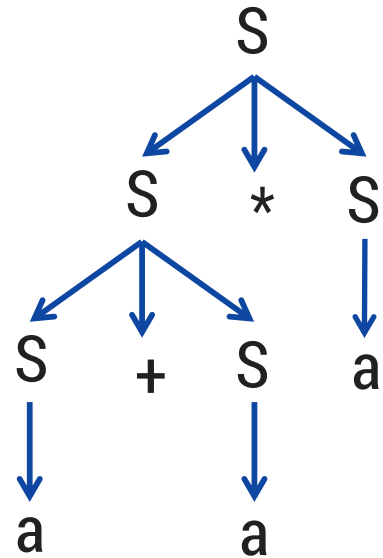
Ambiguous grammar

- ▶ Ambiguous grammar is one that produces more than one leftmost or more than one rightmost derivation for the same sentence.

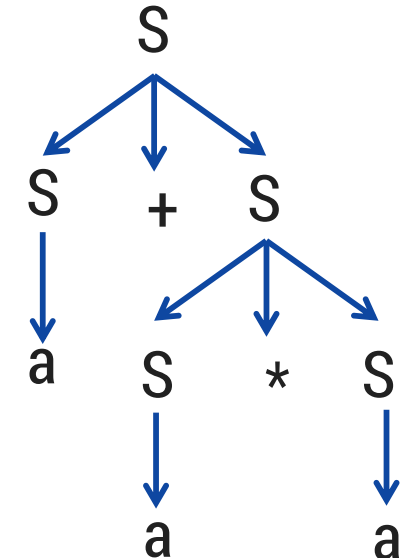
▶ Grammar: $S \rightarrow S+S \mid S*S \mid (S) \mid a$

Output string: $a+a*a$

S
 $\rightarrow \underline{S}*S$
 $\rightarrow \underline{S}+S*S$
 $\rightarrow a+\underline{S}*S$
 $\rightarrow a+a*\underline{S}$
 $\rightarrow a+a*a$



S
 $\rightarrow \underline{S}+S$
 $\rightarrow a+\underline{S}$
 $\rightarrow a+\underline{S}*S$
 $\rightarrow a+a*\underline{S}$
 $\rightarrow a+a*a$



- ▶ Here, Two leftmost derivation for string $a+a*a$ is possible hence, above grammar is ambiguous.

Exercise: Ambiguous Grammar

Check Ambiguity in following grammars:

1. $S \rightarrow aS \mid Sa \mid \epsilon$ (output string: aaaa)
2. $S \rightarrow aSbS \mid bSaS \mid \epsilon$ (output string: abab)
3. $S \rightarrow SS^+ \mid SS^* \mid a$ (output string: aa+a*)
4. $\langle \text{exp} \rangle \rightarrow \langle \text{exp} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle$
 $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \langle \text{letter} \rangle \mid \langle \text{letter} \rangle$
 $\langle \text{letter} \rangle \rightarrow a|b|c|\dots|z$ (output string: a+b*c)
5. Prove that the CFG with productions: $S \rightarrow a \mid Sa \mid bSS \mid SSb \mid SbS$ is ambiguous (Hint: consider output string yourself)

Simplified forms & Normal forms

Nullable Variable

- ▶ A Nullable variable in a CFG, $G = (V, \Sigma, S, P)$ is defined as follows:
 1. Any variable A for which P contains $A \rightarrow \epsilon$ is nullable.
 2. If P contains the production $A \rightarrow B_1 B_2 \dots B_n$ and B_1, B_2, \dots, B_n are nullable variables, then A is nullable.
 3. No other variables in V are nullable.

Eliminate \wedge production

$S \rightarrow a \ X \mid Yb$
 $X \rightarrow \wedge \mid S$
 $Y \rightarrow bY \mid b$

Nullable variable = {X}

$S \rightarrow aX \mid Yb \mid a^{\wedge}$
 $X \rightarrow \wedge \mid S$
 $Y \rightarrow bY \mid b$

Replacing X by \wedge in all productions containing X on RHS and rewriting the production again

$S \rightarrow aX \mid Yb \mid a$
 $X \rightarrow S$
 $Y \rightarrow bY \mid b$

Removing \wedge productions

Exercise: Eliminate \wedge production

$S \rightarrow AC$

$A \rightarrow aAb | \wedge$

$C \rightarrow aC | a$

After elimination of \wedge production:

$S \rightarrow AC | C$

$A \rightarrow aAb | ab$

$C \rightarrow aC | a$

$S \rightarrow XaX | bX | Y$

$X \rightarrow XaX | XbX | \wedge$

$Y \rightarrow ab$

After elimination of \wedge production:

$S \rightarrow XaX | bX | Y | aX | Xa | a | b$

$X \rightarrow XaX | XbX | aX | Xa | a | Xb | bX | b$

$Y \rightarrow ab$

A-derivable

- ▶ A variable is called A-derivable ,
 1. If $A \rightarrow B$ is a production, B is A-derivable.
 2. If C is A-derivable, $C \rightarrow B$ is a production, and $B \neq A$, then B is A-derivable.
 3. No other variables are A-derivable.
- ▶ Examples:

$S \rightarrow A$
 $S \rightarrow B$
S-derivable={A,B}

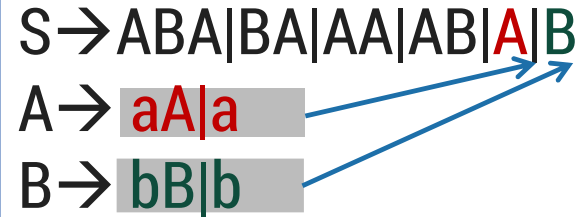
$S \rightarrow A$
 $A \rightarrow B$
S-derivable={A,B}

Unit Production

- ▶ A production of the form $A \rightarrow B$ is termed as unit production. Where A & B are nonterminals.

Elimination of Unit production

$S \rightarrow ABA|BA|AA|AB|A|B$
 $A \rightarrow aA|a$
 $B \rightarrow bB|b$



Unit Productions
are $S \rightarrow A$ and $S \rightarrow B$

$S \rightarrow ABA|BA|AA|AB|aA|a|bB|b$
 $A \rightarrow aA|a$
 $B \rightarrow bB|b$

Removing unit
productions

Left recursion & Left factoring

Left recursion

- ▶ A grammar is said to be **left recursive** if it has a non terminal A such that there is a derivation $A \rightarrow A\alpha$ for some string α .

Algorithm to eliminate left recursion

1. Arrange the non terminals in some order A_1, \dots, A_n
2. For $i := 1$ to n **do begin**
 - for $j := 1$ to $i - 1$ **do begin**
 - replace each production of the form $A_i \rightarrow A_j\gamma$
by the productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \dots \mid \delta_k\gamma$,
where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ are all the current A_j
productions;
 - end**
 - eliminate the immediate left recursion among the A_i - productions
- end**

Left recursion elimination

$$A \rightarrow A\alpha \mid \beta \quad \longrightarrow \quad \begin{array}{l} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \mid \epsilon \end{array}$$

Examples: Left recursion elimination

$E \rightarrow E+T \mid T$

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow T*F \mid F$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$X \rightarrow X\%Y \mid Z$

$X \rightarrow ZX'$

$X' \rightarrow \%YX' \mid \varepsilon$

Exercise: Left recursion

1. $A \rightarrow Abd \mid Aa \mid a$
 $B \rightarrow Be \mid b$
2. $A \rightarrow AB \mid AC \mid a \mid b$
3. $S \rightarrow A \mid B$
 $A \rightarrow ABC \mid Acd \mid a \mid aa$
 $B \rightarrow Bee \mid b$
4. $\text{Exp} \rightarrow \text{Exp} + \text{term} \mid \text{Exp} - \text{term} \mid \text{term}$

Left factoring

Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing.

Algorithm to left factor a grammar

Input: Grammar G

Output: An equivalent left factored grammar.

Method:

For each non terminal A find the longest prefix α common to two or more of its alternatives. If $\alpha \neq \epsilon$, i.e., there is a non trivial common prefix, replace all the A productions $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$ where γ represents all alternatives that do not begin with α by

$$\begin{aligned} A &\rightarrow \alpha A' \mid \gamma \\ A' &\rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \end{aligned}$$

Here A' is new non terminal. Repeatedly apply this transformation until no two alternatives for a non-terminal have a common prefix.

Left factoring elimination

$$A \rightarrow \alpha \beta \mid \alpha \delta \longrightarrow A \rightarrow \alpha A' \\ A' \rightarrow \beta \mid \delta$$

Example: Left factoring elimination

$S \rightarrow aAB \mid aCD$

$S \rightarrow aS'$

$S' \rightarrow AB \mid CD$

$A \rightarrow xByA \mid xByAzA \mid a$

$A \rightarrow xByAA' \mid a$

$A' \rightarrow \epsilon \mid zA$

$A \rightarrow aAB \mid aA \mid a$

$A \rightarrow aA'$

$A' \rightarrow AB \mid A \mid \epsilon$

$A' \rightarrow AA'' \mid \epsilon$

$A'' \rightarrow B \mid \epsilon$

Exercise

1. $S \rightarrow iEtS \mid iEtSeS \mid a$
2. $A \rightarrow ad \mid a \mid ab \mid abc \mid x$

Regular expressions vs CFGs

Regular Expressions	Context-free grammar
Lexical rules are quite simple in case of Regular Expressions.	Lexical rules are difficult in case of Context free grammar.
Notations in regular expressions are easy to understand.	Notations in Context free grammar are quite complex.
A set of string is defined in case of Regular Expressions.	In Context free grammar the language is defined by the collection of productions.
It is easy to construct efficient recognizer from Regular Expressions.	By using the context free grammar, it is very difficult to construct the recognizer.
There is proper procedure for lexical and syntactical analysis in case of Regular Expressions.	There is no specific guideline for lexical and syntactic analysis in case of Context free grammar.
Regular Expressions are most useful for describing the structure of lexical construct such as identifiers, constant etc.	Context free grammars are most useful in describing the nested chain structure or syntactic structure such as balanced parenthesis, if else etc. and these can't be define by Regular Expression.

Thank You