Chapter 4

Context Free Languages and Context Free Grammars

Grammar (Review)

- A grammar is a 4-tuple $G = (NT, \mathcal{T}, S, P)$, where
  - ***NT***: a finite set of *non-terminal* symbols
  - $\mathcal{T}$: a finite set of *terminal* (alphabet) symbols
  - ***S***: is the starting symbol $\boldsymbol{S} \in \boldsymbol{NT}$
  - ***P***: a finite set of production rules: $x \rightarrow y$
    - $x \in \boldsymbol{NT}$ $\quad$ $y \in (\boldsymbol{NT} \cup \boldsymbol{T})^*$
- Derivation of $w \in T^*$:
  - $S \Rightarrow w_1 \cdots \Rightarrow w_n \Rightarrow w$ (written as $S \Rightarrow^* w$)
- The language generated by the grammar is
  - $L(G) = \{w \in T^* : S \Rightarrow^* w\}$

Context-Free Grammars (CFG)

- **Context-Free Grammars** (CFG) are language-description mechanisms ***used to generate the strings*** of a language.

- A grammar is said to be **context-free** if every rule has a **single non-terminal** on the **left-hand side**.
  - $A \to \alpha$
    - $A \in \boldsymbol{NT}$: single non-terminal on the left hand side
    - $\alpha \in (\boldsymbol{NT} \cup \boldsymbol{T})^*$: string of terminals and non-terminals
- A language L is called **context-free language** (**CFL**) if and only if there is a context-free grammar G (**CFG**) that generates it (i.e., $L = L(G)$).

Context-Free Grammars…. **Examples**

Example

- $G = (\{S\}, \{a, b\}, S, \{S \rightarrow aSa \mid bSb \mid \varepsilon\})$
  - *For aabbaa,* typical derivation might be:
    - $S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$
  - Grammar generates language
    - $L(G)=\{ww^R : w \in \{a, b\}^*\}$

Context-Free Grammars….

- What is the **language** of this grammar?
  - $G = (\{S\}, \{a, b\}, S, \{S \rightarrow aSb, S \rightarrow ab\})$
    - $L(G) = \{a^n b^n \mid n \geq 1\}$

- What is the **language** of this grammar?
  - $G = (\{S\}, \{a, b\}, S, \{S \rightarrow aSb, S \rightarrow \varepsilon\})$
    - $L(G) = \{a^n b^n \mid n \geq 0\}$

Context-Free Grammars….

- What is the **language** of this grammar?
  - $G = (\{S, A\}, \{a, b\}, S, \{\ S \rightarrow aSa \mid aAa$
    $A \rightarrow bA \mid b\ \})$
    - $L(G) = \{a^n b^m a^n \mid n \geq 1, m \geq 1\}$

- What is the **language** of this grammar?
  - $G_1 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB$
    $A \rightarrow aA \mid a$
    $B \rightarrow bB \mid \varepsilon\ \})$
  - $G_2 = (\{S, B\}, \{a, b\}, S, \{S \rightarrow aS \mid aB$
    $B \rightarrow bB \mid \varepsilon\ \})$
    - $L(G) = \{a^+ b^*\} = \{a^n b^m \mid n \geq 1, m \geq 0\}$

Context-Free Grammars….

- What is the **grammar** of this language?
  - $L = \{a^* b a^* b a^*\}$
    - $G_1 = (\{S, A\}, \{a, b\}, S, \{\, S \rightarrow AbAbA$

      $A \rightarrow aA \mid \varepsilon \,\})$

    - $G_2 = (\{S, A, C\}, \{a, b\}, S, \{\, S \rightarrow aS \mid bA$

      $A \rightarrow aA \mid bC \,\}$

      $C \rightarrow aC \mid \varepsilon$

Context-Free Grammars….

- What is the **grammar** of this language?
  - A language over $\{a, b\}$ with at least 2 b's
    - $G_1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow AbAbA$

      $A \rightarrow aA \mid bA \mid \varepsilon \}$

    - $G_2 = (\{S, A, C\}, \{a, b\}, S, \{S \rightarrow aS \mid bA$

      $A \rightarrow aA \mid bC$

      $C \rightarrow aC \mid bC \mid \varepsilon \}$

Context-Free Grammars….

- What is the **grammar** of this language?
  - A language over $\{a, b\}$ of even-length strings.
    - $G = (\{S, O\}, \{a, b\}, S, \{\, S \rightarrow aO \mid bO \mid \varepsilon$
      $O \rightarrow aS \mid bS \,\}$

- What is the **grammar** of this language?
  - A language over $\{a, b\}$ of an even no. of *b*’s
    - $G = (\{S, B, C\}, \{a, b\}, S, \{\ \ S \rightarrow aS \mid bA \mid \varepsilon$
      $A \rightarrow aA \mid bS \quad \}$

Context-Free Grammars….Exercise

- What is the **grammar** of this language?
  - A language over {*a*, *b*, *c*} that do not contain the substring *abc.*

Context-Free Grammars….

- Write a CFG for the following Languages

$L_1 = \{wcw^R \mid w \in \{a, b\}^*\}$

$$S \rightarrow aSa \mid bSb \mid c$$

$L_2 = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\}$

$$S \rightarrow XY$$
$$X \rightarrow aXb \mid ab$$
$$Y \rightarrow cYd \mid cd$$

- $G = (\{S\}, \{a, b\}, S, \{S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon\})$
  - is this grammar context-free?
    - **Yes**, there is a ***single variable*** on the left hand side.

Context-Free Grammars….

- Are **regular** languages **context-free**? Why?
  - Yes.
  - But, ***not all*** context-free grammars are regular.

- Regular languages are a proper subset of the class of context-free languages.
  - Regular grammars are a proper subset of context-free grammars.

- Non-regular languages can be generated by context-free grammars,
  - so the term context-free generally includes non-regular languages and regular languages.

Context-Free Grammars….

- **Constructing (right linear) CFG from DFA**

1. Each state of the DFA will be represented by a non-terminal
2. The initial state will correspond to the start non-terminal
3. For each transition $\delta(q_i, a) = q_j$, add a rule

   $q_i \rightarrow aq_j$
4. For each accepting state q_f , add a rule

   $q_f \rightarrow \varepsilon$

Leftmost & Rightmost Derivations

- Given the grammar:
  - $S \rightarrow aAB, \qquad A \rightarrow bBb, \qquad B \rightarrow A \mid \varepsilon$
  - String *abbbb* can be derived in different ways:
  - **Left-most derivation**:
    - always replace the leftmost *NT* in each sentential form
    - $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Rightarrow abbbb$
  - **Right-most derivation**:
    - always replace the rightmost *NT* in each sentential form
    - $S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb$

Derivations….

Definition:

$\overset{+}{\Rightarrow}$ derives in one step

$\overset{*}{\underset{G}{\Rightarrow}}$ derives in $\geq$ one step

$\Rightarrow$ indicates that the derivation utilizes the rules of grammar *G.*

Derivations….

<u>Example</u>

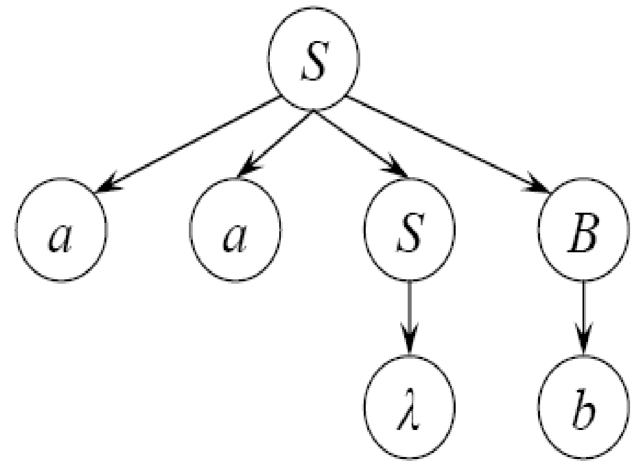
- $S \rightarrow S + E \mid E$
- $E \rightarrow$ number | (S)

- Left-most derivation
  - $S \Rightarrow S+E \Rightarrow E+E \Rightarrow (S)+E \Rightarrow (S+E)+E \Rightarrow (S+E+E)+E \Rightarrow (E+E+E)+E$
    $\Rightarrow (1+E+E)+E \Rightarrow (1+2+E)+E \Rightarrow (1+2+(S))+E \Rightarrow (1+2+(S+E))+E$
    $\Rightarrow (1+2+(E+E))+E$
    $\Rightarrow (1+2+(3+E))+E \Rightarrow (1+2+(3+4))+E \Rightarrow (1+2+(3+4))+5$

- Right-most derivation
  - $S \Rightarrow S+E \Rightarrow E+5 \Rightarrow (S)+5 \Rightarrow (S+E)+5 \Rightarrow (S+(S))+5 \Rightarrow (S+(S+E))+5$
    $\Rightarrow (S+(S+4))+5 \Rightarrow (S+(E+4))+5 \Rightarrow (S+(3+4))+5 \Rightarrow (S+E+(3+4))+5$
    $\Rightarrow (S+2+(3+4))+5 \Rightarrow (E+2+(3+4))+5 \Rightarrow (1+2+(3+4))+5$

Derivation (Parsing) Trees

- Given the grammar
  - $S \rightarrow aaSB \mid \varepsilon,\qquad B \rightarrow bB \mid b$
  - A leftmost derivation
    - $S \Rightarrow aaSB \Rightarrow aaB \Rightarrow aab$
  - A rightmost derivation
    - $S \Rightarrow aaSB \Rightarrow aaSb \Rightarrow aab$



- Both derivations correspond to the parse (or derivation) tree above.

Derivations….

- In a **parse tree**, the nodes labeled with NTs correspond to the left side of a production rule and the children of that node correspond to the right side of the rule, e.g., $S \rightarrow aaSB$.
- The tree structure shows the rule that is applied to each NT (for a specific derivation), without showing the order of rule application.
- Each **internal node** of the tree corresponds to a ***NT***, and the **leaves** of the derivation tree represent the **string of terminals**.
- Note the tree applies to a **specific derivation** and may **not include all rules**, e.g., $B \rightarrow bB$ is not shown above.

Derivations….

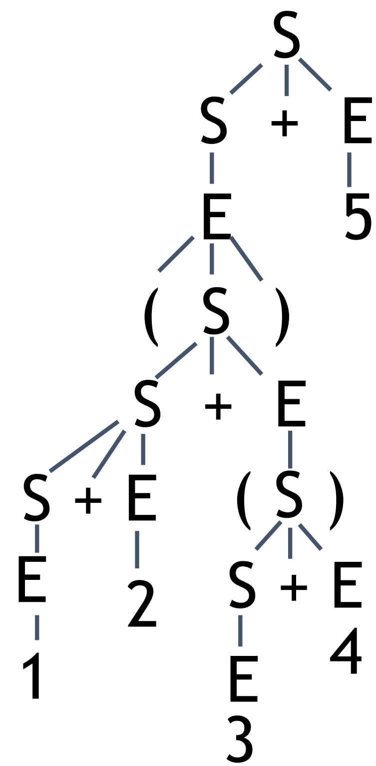
- **Definition**: Let $G = (NT, T, S, P)$ be a context-free grammar. An ordered tree is a **derivation tree** for G if and only if it has the following properties:
  1. The root is labeled S.
  2. Every leaf has a label from $T \cup \{\varepsilon\}$.
  3. Every non-leaf (interior) vertex has a label from NT.
  4. If a vertex has label $A \in NT$, and its children are labeled (left to right) $a_1, a_2, \ldots, a_n$, then P contain a production of the form $A \to a_1 a_2 \ldots a_n$.
  5. A leaf labeled ε has no siblings; that is, a vertex with a child labeled ε can have no other children.
- **Partial derivation tree:** a tree having properties 3–5, but for which property 1 may not hold, and for which property 2 is replaced with:
  - **2a**. every leaf has a label from $NT \cup T \cup \{\varepsilon\}$.

Derivations….

Derivation ⟷ Parse Tree - Example

$S \rightarrow S + E \mid E$

$E \rightarrow \text{number} \mid (S)$



Parse Tree

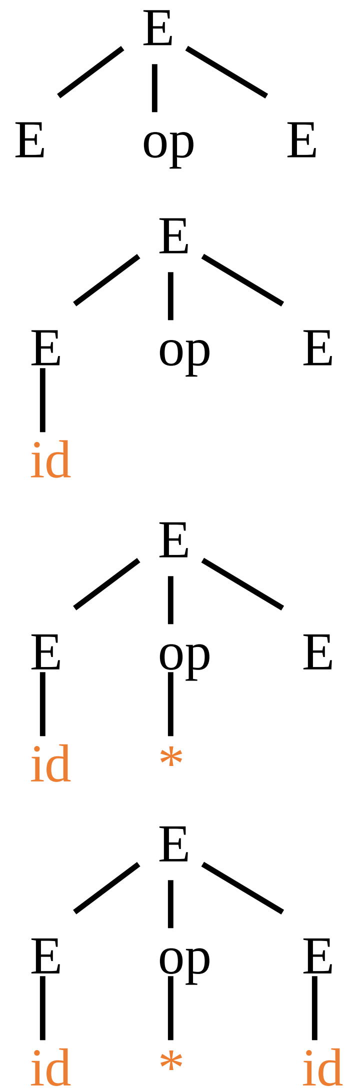
Derivation

$S \Rightarrow S + E \Rightarrow$ E+E $\Rightarrow$ (S)+E $\Rightarrow$ (S+E)+E $\Rightarrow$(S+E+E)+E $\Rightarrow$(E+E+E)+E $\Rightarrow$(1+E+E)+E $\Rightarrow$(1+2+E)+E $\Rightarrow$… $\Rightarrow$(1+2+(3+4))+E $\Rightarrow$(1+2+(3+4))+5

Derivations....

$E \rightarrow E\ \ op\ \ E \mid (\ E\ ) \mid -E \mid id$

$op \rightarrow + \mid - \mid * \mid / \mid \uparrow$



$E \Rightarrow E\ op\ E$

$\Rightarrow id\ op\ E$

$\Rightarrow id * E$

$\Rightarrow id * id$

Derivations….

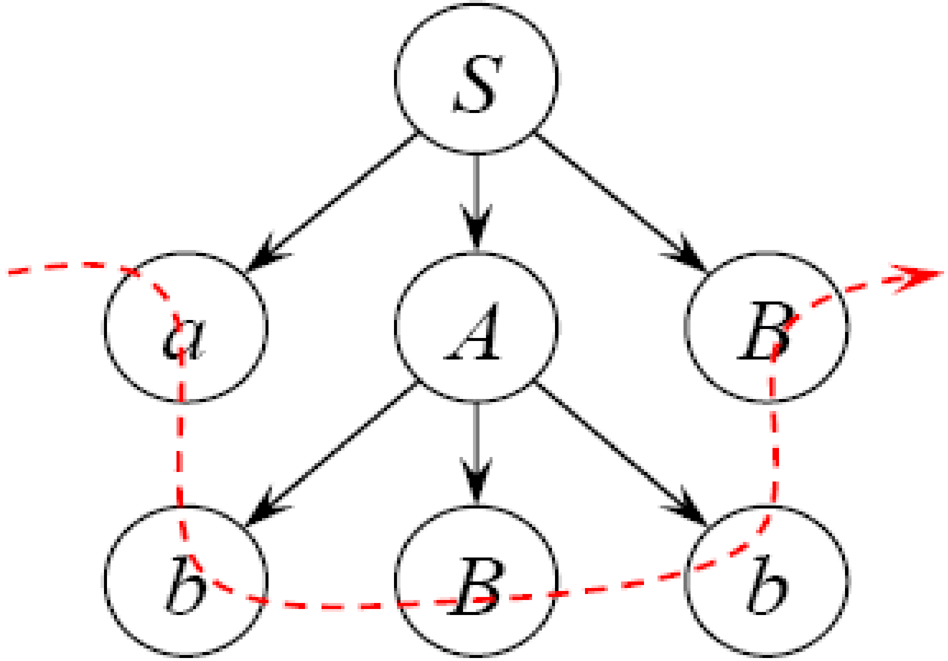
- A **derivation tree** for grammar *G* **yields** a **sentence** of the language *L*(*G*).

- A **partial** derivation tree **yields** a **sentential form.**

- The **yield** of a parse tree is the string of leaf symbols obtained by reading the tree **left-to-right**.
  - The order they are encountered when the tree is traversed in a depth-first manner, always taking the leftmost unexplored branch.

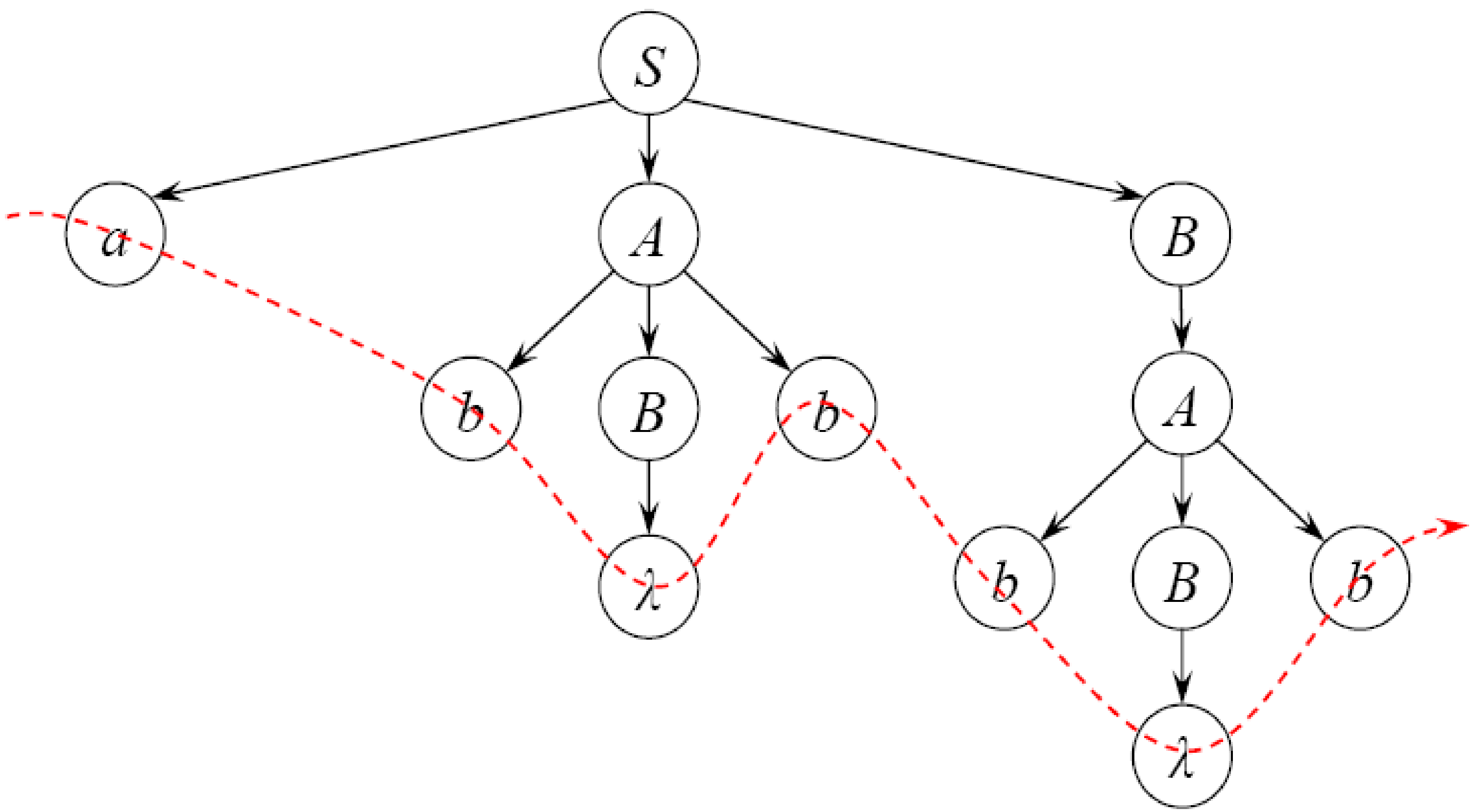
Derivations….

- Given the grammar:
  - $S \rightarrow aAB$ $A \rightarrow bBb$ $B \rightarrow A \mid \varepsilon$,
  - The yield of the **partial derivation** tree is the **sentential form** *abBbB.*



Derivations….

- Given the grammar"
  - $S \rightarrow aAB$ $A \rightarrow bBb$ $B \rightarrow A \mid \varepsilon$,
  - The yield of the **derivation tree** is the **sentence** *abbbb* $\in L(G)$.



Derivations….

- **Theorem** (Relationship between **Sentential Forms** & **Derivation Trees**)
- Let $G = (NT, T, S, P)$ be a CFG. Then,
  - for every $w \in L(G)$, there exists a derivation tree of G whose yield is w.
  - the yield, w, of any derivation tree of G is such that $w \in L(G)$.
  - if t_G is any partial derivation tree for G whose root is labeled S, then the yield of t_G is a **sentential** form of G.

- As a side note, any $w \in L(G)$ has a **leftmost** and a **rightmost** derivation.

Parsing and Ambiguity

- In practical applications, it is usually not enough to decide whether a string belongs to a language.

- It is also important to know how to derive the string from the language.

- **Parsing** uncovers the **syntactical structure** of a string, which is represented by **a parse tree**.

- The syntactical structure is important for **assigning semantics** to the string -- for example, if it is a computer program.

Parsing and Ambiguity…

Application of parsing (Compilers):

- Let G be a context-free grammar for the C language. Let the string $\boldsymbol{w}$ be a C program.

- One thing a compiler does - in particular, the part of the compiler called the "parser" - is determine whether $\boldsymbol{w}$ is a syntactically correct C program.

- It also constructs a **parse tree** for the program that is used in code generation.

- There are many sophisticated and efficient algorithms for parsing. You may study them in more advanced classes (for example, on compilers).

Parsing and Ambiguity…

- <u>**S-grammars**</u>

- **Definition**: A context-free grammar $G = (NT, T, S, P)$ is said to be a simple grammar or **s-grammar** if all of its productions are of the form:
  - $A \rightarrow ax$ where $A \in NT$, $a \in T$, $x \in NT^*$, and any pair (A, a) occurs at most once in P.

- Examples:
  - $S \rightarrow aS \mid bSS \mid c$ is an s-grammar.
  - $S \rightarrow aS \mid bSS \mid aSB \mid c$ is not an s-grammar.
  - because pair (S, a) occurs in two productions:
    - $S \rightarrow aS$, and $S \rightarrow aSB$

Parsing and Ambiguity…

Example:

- Given s-grammar G with production:
  - $S \rightarrow aS \mid bSS \mid c$
- Show the derivation of the string $w = abcc$.
- Since G is an s-grammar,
  - the only way to get the a up front is via rule,
    - $S \rightarrow aS$
  - the only way to get the b is via rule,
    - $S \rightarrow bSS$
  - the only way to get each c is via rule,
    - $S \rightarrow c$
  - Thus, we have parsed the string in 4 steps as,
    - $S \Rightarrow aS \Rightarrow abSS \Rightarrow abcS \Rightarrow abcc$

Parsing and Ambiguity…

- Find an s-grammar for
  - *aaa*b | b*

- We need to start with an ***a*** or have only the ***b***.
  - $S \rightarrow aA$
  - $S \rightarrow b$

  - $A \rightarrow aB$

  - $B \rightarrow aB$
  - $B \rightarrow b$

Parsing and Ambiguity…

- A **terminal string** may be generated by a **number** of **different derivations**.

- Let *G* be a CFG. A string *w* is in *L*(*G*), iff there is
  - a **leftmost derivation** of *w* from *S*. ($S \overset{*}{\underset{lm}{\Rightarrow}} w$)

- Question:
  - Is there a **unique leftmost derivation** of **every** sentence (string) in the language of a grammar?
    - NO

Parsing and Ambiguity…

- Consider the expression grammar:

$$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid id$$

- **Two** different **leftmost** derivations of id + id * id

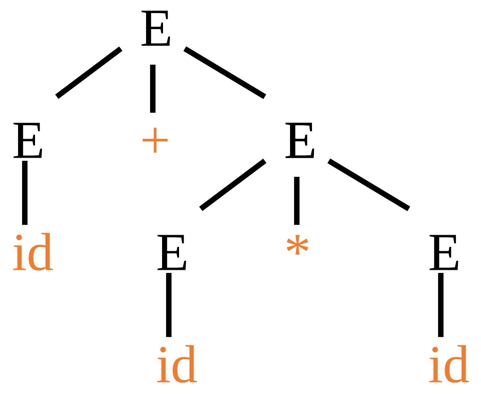
$\underline{E} \Rightarrow \underline{E} + E$

$\Rightarrow id + \underline{E}$

$\Rightarrow id + \underline{E} * E$

$\Rightarrow id + id * \underline{E}$

$\Rightarrow id + id * id$



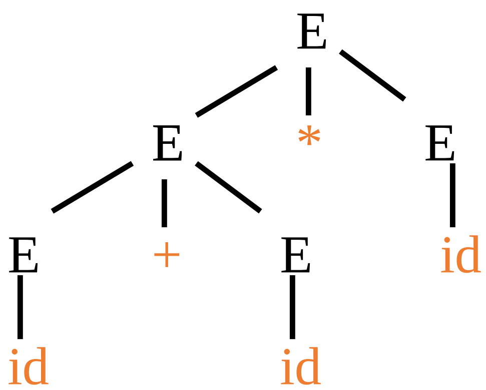
Parsing and Ambiguity…

- Consider the expression grammar:

$$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid id$$

- **Two** different **leftmost** derivations of id + id * id

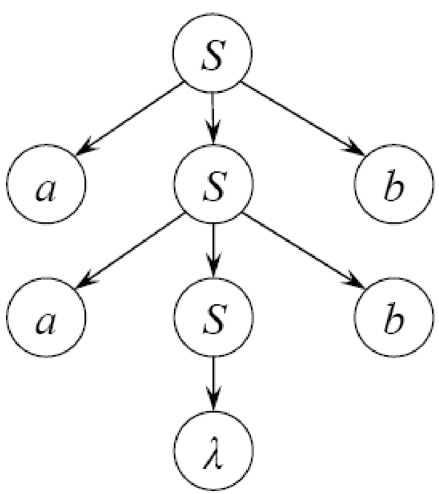
$$
\begin{aligned}
\underline{E} &\Rightarrow \underline{E} * E \\
&\Rightarrow \underline{E} + E * E \\
&\Rightarrow id + \underline{E} * E \\
&\Rightarrow id + id * \underline{E} \\
&\Rightarrow id + id * id
\end{aligned}
$$

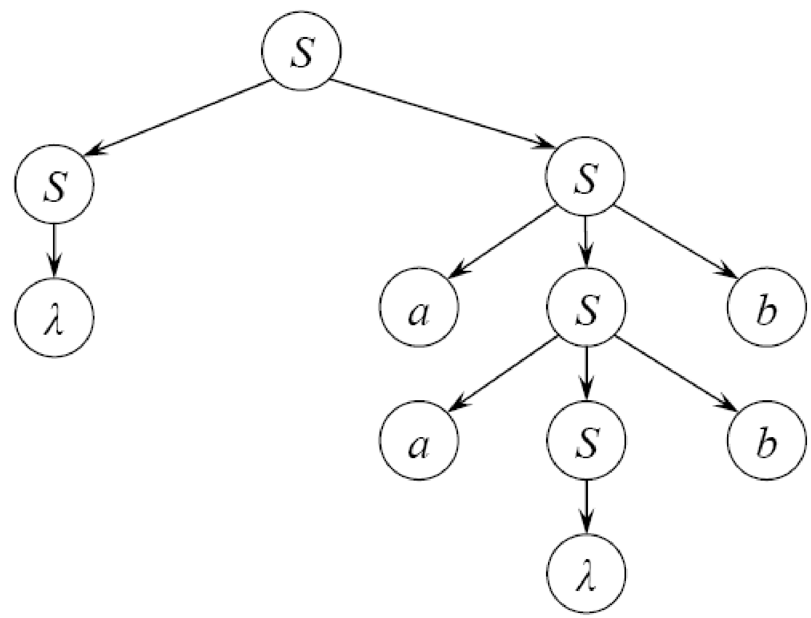


Parsing and Ambiguity…

Definition:

- a grammar *G* is **ambiguous** if there is a string with at least **two parse trees**,
  - two or more leftmost or rightmost derivations.
- Example: CFG *G* with productions $S \rightarrow aSb \mid SS \mid \varepsilon$ is ambiguous because there are **two parse tree** for *w* = *aabb* as shown below.





Parsing and Ambiguity…

- A CFG is **ambiguous** if there is a string $w \in L(G)$ that can be derived by **two distinct leftmost derivations.**

- A grammar *G* is **ambiguous** if there exists a sentence in *G* with more than **one** derivation (parsing) tree.

- A grammar that is not ambiguous is called **unambiguous.**

- If *G* is ambiguous then $L(G)$ is **not** necessarily ambiguous.

- A language *L* is **inherently ambiguous** if there is no unambiguous grammar that generates it.

Parsing and Ambiguity…

- Let G be $S \rightarrow aS \mid Sa \mid a.$
- G is ambiguous since the string *aa* has 2 distinct leftmost derivation.
  - $S \Rightarrow aS \Rightarrow aa$
  - $S \Rightarrow Sa \Rightarrow aa$
- $L(G) = a^{+}$
- This language is also generated by the unambiguous grammar $S \rightarrow aS \mid a.$
- $L(G)$ is **not** ambiguous. **Why**?

Parsing and Ambiguity…

- Let G be $S \rightarrow bS \mid Sb \mid a$ $L(G) = b^{*}ab^{*}$
- G is **ambiguous** since the string *bab* has 2 distinct **leftmost derivation.**
  - $S \Rightarrow bS \Rightarrow bSb \Rightarrow bab$
  - $S \Rightarrow Sb \Rightarrow bSb \Rightarrow bab$

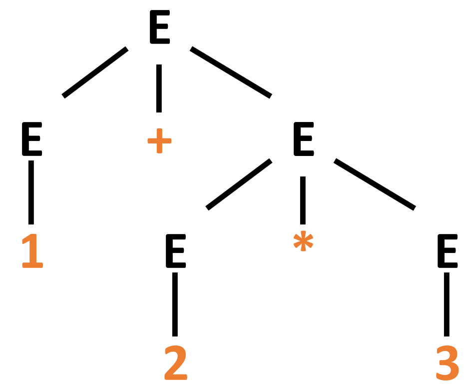
- The ability to generate the *b*'s in either order must be eliminated to obtain an unambiguous grammar.
- This language is also generated by the unambiguous grammars.

G_1: $S \rightarrow bS \mid aA$
$A \rightarrow bA \mid e$

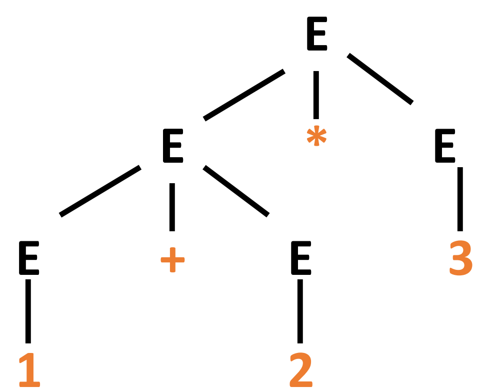
G_2: $S \rightarrow bS \mid A$
$A \rightarrow Ab \mid a$

Parsing and Ambiguity…

- **Eliminating Ambiguity**
  - Consider the following grammar,
    - $E \rightarrow E + E \mid E * E \mid num$
  - Consider the sentence $1 + 2 * 3$
  - Leftmost Derivation 1
    - $E \Rightarrow E + E$
      $\Rightarrow 1 + E$
      $\Rightarrow 1 + E * E$
      $\Rightarrow 1 + 2 * E$
      $\Rightarrow 1 + 2 * 3$



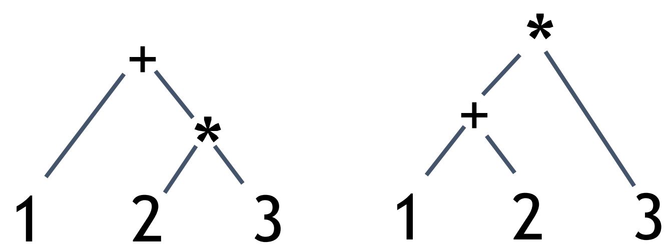
  - Leftmost Derivation 2
    - $E \Rightarrow E * E$
      $\Rightarrow E + E * E$
      $\Rightarrow 1 + E * E$
      $\Rightarrow 1 + 2 * E$
      $\Rightarrow 1 + 2 * 3$



Parsing and Ambiguity…

- Different parse trees correspond to different evaluations (meaning).

LMD-1 LMD-2



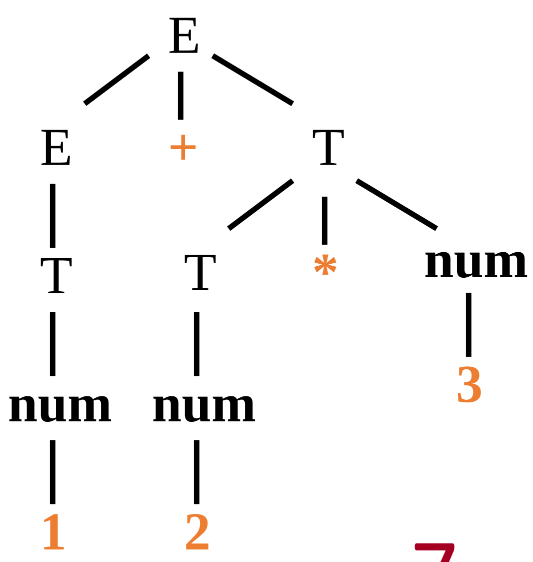
= 7 = 9

Parsing and Ambiguity…

- **Ambiguous**: $E \rightarrow E + E \;\mid E * E \mid \textit{number.}$
- Both + and * have the same precedence.
- To remove ambiguity, you have to give + and * different precedence.
- Let us say * has higher precedence than +.
  - $E \rightarrow E + T \mid T$
  - $T \rightarrow T * num \mid num$

Parsing and Ambiguity…

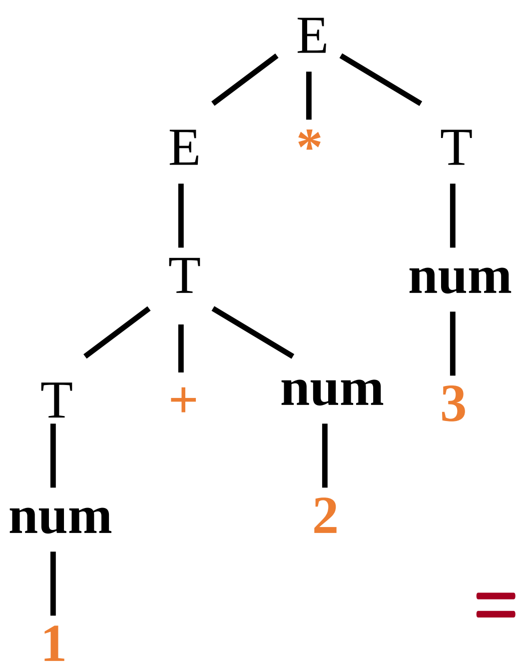
- **Eliminating Ambiguity**
    - $E \rightarrow E + T \mid T$
    - $T \rightarrow T * num \mid num$
- Consider the sentence 1 + 2 * 3.
- Leftmost Derivation (only one LMD).
    - E $\Rightarrow$ E + T
      $\Rightarrow$ T + T
      $\Rightarrow$ num + T
      $\Rightarrow$ 1 + T
      $\Rightarrow$ 1 + T * num
      $\Rightarrow$ 1 + num * num
      $\Rightarrow$ 1 + 2 * num
      $\Rightarrow$ 1 + 2 * 3



= 7

Parsing and Ambiguity…

- Let us say + has higher precedence than *.
  - $E \rightarrow E * T \mid T$
  - $T \rightarrow T + num \mid num$
  - Consider the sentence 1 + 2 * 3.
  - Leftmost Derivation (only one LMD).
    - E $\Rightarrow$ E * T
      $\Rightarrow$ T * T
      $\Rightarrow$ T + num * T
      $\Rightarrow$ num + num * T
      $\Rightarrow$ 1 + num * T
      $\Rightarrow$ 1 + 2 * T
      $\Rightarrow$ 1 + 2 * num
      $\Rightarrow$ 1 + 2 * 3



= 9

Parsing and Ambiguity…

- A derivation tree…
  - Corresponds to exactly one leftmost derivation.
  - Corresponds to exactly one rightmost derivation.

- CFG **ambiguous** $\Leftrightarrow$ any of following equivalent statements:
  - $\exists$ string *w* with multiple derivation trees.
  - $\exists$ string *w* with multiple leftmost derivations.
  - $\exists$ string *w* with multiple rightmost derivations.

- **Note**: Defining *grammar*, not language, ambiguity.

CFGs & Programming Languages

- Programming languages are **context-free**, but **not regular.**

- Programming languages have the following features that require infinite "**stack memory**".
  - matching parentheses in algebraic expressions.
  - nested if .. then .. else statements.
  - nested loops.
  - block structure.

CFGs & Programming Languages…

<unsigned constant> → *<unsigned number>*
<constant> → *<unsigned number>* | *<sign> <unsigned number>*

<unsigned number> → *<unsigned integer>* | *<unsigned real>*
<unsigned integer> → *<digit> <unsigned integer>* | *<digit>*
<unsigned real> → *<unsigned integer>* **.** *<unsigned integer>* |
<unsigned integer> **.** *<unsigned integer>* **E** *<exp>* |
<unsigned integer> **E** *<exp>*

<exp> → *<unsigned integer>* | *<sign> <unsigned integer>*

<digit> → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
<letter> → **a** | **b** | **c** | … | **y** | **z** | **A** | **B** | **C** | … | **Y** | **Z**
<sign> → **+** | **-**

<identifier> → *<letter> <identifier tail>*
<identifier tail> → *<letter> <identifier tail>* | *<digit> <identifier tail>*

CFGs & Programming Languages…

<expression> → *<simple expression>*

<simple expression> → *<term>* | *<sign term>* |
<simple expression> *<adding operator>* *<term>*

<term> → *<factor>* |
<term> *<multiplying operator>* *<factor>*

<factor> → *<variable>* | *<unsigned constant>* | **(***<expression>***)**

<adding operator> → **+** | **-**

<multiplying operator> → ***** | **/** | **div** | **mod**

Simplification of CFGs and Normal Forms

Methods for Transforming Grammars
A Useful Substitution Rule

Theorem

- This intuitive theorem allows us to **simplify** grammars.
- Let $G = (NT, T, S, P)$ be a context-free grammar. Suppose that P contains a production rule of the form,
  - $A \rightarrow xBz$
  - Assume that A and B are different NT and that,
  - $B \rightarrow y_1 \mid y_2 \mid \ldots \mid y_n$ is the set of all productions in P which have B as the left side.
- Let $G' = (NT, T, S, P')$ be the grammar in which P' is constructed from P by replacing rule,
  - $A \rightarrow xBz$ with $A \rightarrow x\,y_1z \mid xy_2z \mid \ldots \mid xy_nz$
- **Then $L(G') = L(G)$.**

Methods for Transforming Grammars...
A Useful Substitution Rule

- Let G be
  - $S \rightarrow a \mid aaS \mid abBc$
  - $B \rightarrow abbS \mid b$
- Applying theorem 6.1 (on Textbook) results in
  - $S \rightarrow a \mid aaS \mid ababbSc \mid abbc$
  - $B \rightarrow abbS \mid b$
- The rules $B \rightarrow abbS \mid b$, which are still part of the grammar, no longer serve any purpose.
  - Both of these **useless** rules may be deleted without effectively changing the grammar.

Methods for Transforming Grammars…
Removing Useless Productions

- A **non-terminal** *A* is **useful** (it occurs in at least one derivation) if:
  - it is **reachable**: occurs in a sentential form $S \Rightarrow^* \alpha A \beta$.
  - it is **live**: generates a terminal string $A \Rightarrow^* w \in T^*$.

- A non-terminal *A* is **useless** if:
  - *A* does ***not occur*** in any sentential form.
    - It **cannot be reached** from start symbol.

  OR
  - *A* does not generate any ***string of terminals***.
    - It **cannot derive** a terminal string.

- A **terminal** is useful if it occurs in a sentence $w \in L(G)$.

- Any production involving a useless symbol is a **useless production**.

Methods for Transforming Grammars…
Removing Useless Productions

- To eliminate useless symbols:
  - First: Find the set TERM that contains **all non-terminals** that derive a terminal string.
    - $A \Rightarrow^* w$, where $w \in T^*$
  - Non-terminals NOT in TERM are **useless**, they cannot contribute to generate strings in $L(G)$.

  - Second: Find the set REACH that contains all non-terminals $A \in$ TERM that are reachable from S.
    - $S \Rightarrow^* \alpha A \beta$

Methods for Transforming Grammars…
Removing Useless Productions

- **Example 1**
  - G: $S \rightarrow AC \mid BS \mid B$
    $A \rightarrow aA \mid aF$
    $B \rightarrow CF \mid b$ $\leftarrow$
    $C \rightarrow cC \mid D$
    $D \rightarrow aD \mid BD \mid C$
    $E \rightarrow aA \mid BSA$
    $F \rightarrow bB \mid b$ $\leftarrow$
  - $L(G)$ is b^+
  - $B, F \in$ TERM, since both generate terminals
  - $S \in$ TERM, since $S \rightarrow B$ and hence $S \Rightarrow^* b$
  - $A \in$ TERM, since $A \rightarrow aF$ and hence $A \Rightarrow^* ab$
  - $E \in$ TERM, since $E \rightarrow aA$ and hence $E \Rightarrow^* aab$

Methods for Transforming Grammars…
Removing Useless Productions

- *C* and *D* do not belong to **TERM**, so all rules containing *C* and *D* are **removed.**
- The new grammar is:
  - G_T:
$$\begin{aligned} S &\to BS \mid B \\ A &\to aA \mid aF \\ B &\to b \\ E &\to aA \mid BSA \\ F &\to bB \mid b \end{aligned}$$
- All non-terminals in $\boldsymbol{G_T}$ derive terminal strings.
- Now, we must remove the non-terminals that do not occur in sentential forms of the grammar.
- A set **REACH** is built that contains all non-terminals $\in$ TERM derivable from *S.*

Methods for Transforming Grammars…

Removing Useless Productions

- G_T: $S \rightarrow BS \mid B$
  $A \rightarrow aA \mid aF$
  $B \rightarrow b$
  $E \rightarrow aA \mid BSA$
  $F \rightarrow bB \mid b$

- $S \in$ **REACH**, since it is the start symbol.
  - $B \in$ REACH, since $S \rightarrow BS$, and hence B is derivable from S.
  - A, E, and F can not be derived from S or B, so all rules containing A, E and F are removed.

Methods for Transforming Grammars…

Removing Useless Productions

- The new grammar is,
  - G_U: $S \rightarrow BS \mid B$
    $B \rightarrow b$
  - $L(G_U) = b^+$
- The set of terminals of G_U is $\{b\}$, a is removed since it does not occur in any string in the language of G_U.

- The order is important:
  - Applying **Second Step** (REACH) before **First Step** (TERM) may not remove all useless symbols.

Methods for Transforming Grammars…

Removing Useless Productions: Exercise

- **Remove all useless productions**.
  - $S \rightarrow AB \mid CD \mid ADF \mid CF \mid EA$
  - $A \rightarrow abA \mid ab$
  - $B \rightarrow bB \mid aD \mid BF \mid aF$
  - $C \rightarrow cB \mid EC \mid Ab$
  - $D \rightarrow bB \mid FFB$
  - $E \rightarrow bC \mid AB$
  - $F \rightarrow abbF \mid baF \mid bD \mid BB$
  - $G \rightarrow EbE \mid CE \mid ba$

- **Remove all useless productions.**
  - Let $G = (\{S, A, B, C\}, \{a, b\}, S, \{S \rightarrow aS \mid A \mid C, A \rightarrow a, B \rightarrow aa, C \rightarrow aCb\})$ be a CFG.
  - Final grammar is
    - $G' = (\{S\}, \{a\}, S, \{S \rightarrow aS \mid a\})$

Methods for Transforming Grammars…

Removing ε-Productions

- Let G be $S \rightarrow SaB \mid aB \qquad B \rightarrow bB \mid \varepsilon$
- A non-terminal symbol that can **derive** the *null string* (ε) is called **nullable**.
- For example, in G above, B is nullable since $B \rightarrow \varepsilon$
- A grammar *without* **nullable** non-terminals is called **non-contracting.**
- G, above, is not non-contracting, since it has one nullable non-terminal, which is B.

Methods for Transforming Grammars…
Removing ε-Productions

- **How to find nullable non-terminals?**

  - Mark all non-terminals A for which there exists a production of the form $A \rightarrow \varepsilon$.

  - Repeat
    - Mark non-terminal X for which there exists $X \rightarrow \beta$ and all symbols in β have been marked as nullable.

  - Until no new non-terminal is marked.

Methods for Transforming Grammars…
Removing ε-Productions

- The set of nullable non-terminals of the grammar.
  - $S \rightarrow ACA$
    $A \rightarrow aAa \mid B \mid C$
    $B \rightarrow bB \mid b$
    $C \rightarrow cC \mid \varepsilon$
  - is $\{S, A, C\}$
  - C is nullable
    - since $C \rightarrow \boldsymbol{\varepsilon}$ and hence $C \Rightarrow^* \varepsilon$.
  - A is nullable
    - since $A \rightarrow C$, and C is nullable.
  - S is nullable
    - since $S \rightarrow ACA$, and A and C are nullable.

Methods for Transforming Grammars...
Removing ε-Productions-Exercise

- Find nullable non-terminals.

$$S \rightarrow aS \mid SS \mid bA$$
$$A \rightarrow BB$$
$$B \rightarrow CC \mid ab \mid aAbC$$
$$C \rightarrow \varepsilon$$

Methods for Transforming Grammars…
Removing ε-Productions

- If $\varepsilon \notin L(G)$, we can eliminate all productions $A \rightarrow \varepsilon$.

- For every B referring to A:

$$\begin{aligned} B &\rightarrow \alpha\, A\, \beta \mid \ldots \\ A &\rightarrow \varepsilon \mid \ldots \end{aligned} \quad \Rightarrow \quad \begin{aligned} B &\rightarrow \alpha\, \beta \mid \alpha\, A\, \beta \mid \ldots \\ A &\rightarrow \ldots \end{aligned}$$

- **For example**, if $B \rightarrow \varepsilon$ and $A \rightarrow BABa$.
- Then after eliminating the rule $B \rightarrow \varepsilon$, new rules for A will be added,
  - $A \rightarrow BABa$
  - $A \rightarrow ABa$
  - $A \rightarrow BAa$
  - $A \rightarrow Aa$

Methods for Transforming Grammars...

Removing ε-Productions

- Let G be
  - $S \rightarrow SaB \mid aB$
    $B \rightarrow bB \mid \varepsilon$
- After removing ε-productions, the new grammar will be,
  - $S \rightarrow SaB \mid Sa \mid aB \mid a$
    $B \rightarrow bB \mid b$
- The removal of ε-productions *increases the number of rules* but *reduces the length of derivations*.

Methods for Transforming Grammars…
Removing ε-Productions

- Let G
  $S \rightarrow ACA$
  $A \rightarrow aAa \mid B \mid C$
  $B \rightarrow bB \mid b$
  $C \rightarrow cC \mid \varepsilon$

- The equivalent essentially **non-contracting** grammar G_L is
  - G_L: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \varepsilon$
    $A \rightarrow aAa \mid aa \mid B \mid C$
    $B \rightarrow bB \mid b$
    $C \rightarrow cC \mid c$

- Since $S \Rightarrow^* \varepsilon$ in G, the rule $S \rightarrow \varepsilon$ is allowed in G_L, but all other ε-productions are replaced.

- A grammar satisfying these conditions is called ***essentially non-contracting*** (only start symbol is nullable).

Methods for Transforming Grammars...

Removing ε-Productions

- Let G be,
  - $S \rightarrow aS \mid SS \mid bA$
  - $A \rightarrow BB$
  - $B \rightarrow ab \mid aAbC \mid aAb \mid CC$
  - $C \rightarrow \varepsilon$

- We eliminate $C \rightarrow \varepsilon$ by replacing:
  - $B \rightarrow CC$ into $B \rightarrow CC$, $B \rightarrow C$, and $B \rightarrow \varepsilon$
  - $B \rightarrow aAbC$ into $B \rightarrow aAbC$ and $B \rightarrow aAb$

- Since $C \rightarrow \varepsilon$ is only C production
  - only $B \rightarrow \varepsilon$ and $B \rightarrow aAb$ retained.

- The new grammar:
  - $S \rightarrow aS \mid SS \mid bA$
  - $A \rightarrow BB$
  - $B \rightarrow \varepsilon \mid ab \mid aAb$

Methods for Transforming Grammars…

Removing ε-Productions

- The new grammar:
  - $S \rightarrow aS \mid SS \mid bA$
  - $A \rightarrow BB$
  - $B \rightarrow \varepsilon \mid ab \mid aAb$
- We eliminate $B \rightarrow \varepsilon$ by replacing,
  - $A \rightarrow BB$ into $A \rightarrow BB$, $A \rightarrow B$, and $A \rightarrow \varepsilon$
- Since there are other B productions, these are all retained.
- The new grammar:
  - $S \rightarrow aS \mid SS \mid bA$
  - $A \rightarrow BB \mid B \mid \varepsilon$
  - $B \rightarrow ab \mid aAb$

Methods for Transforming Grammars...

Removing ε-Productions

- The new grammar:
  - $S \rightarrow aS \mid SS \mid bA$
  - $A \rightarrow BB \mid B \mid \varepsilon$
  - $B \rightarrow ab \mid aAb$

- Finally we eliminate $A \rightarrow \varepsilon$ by replacing
  - $B \rightarrow aAb$ into $B \rightarrow aAb$, $B \rightarrow ab$
  - $S \rightarrow bA$ into $S \rightarrow bA \mid b$

- The final CFG is:
  - $S \rightarrow aS \mid SS \mid bA \mid b$
  - $A \rightarrow BB \mid B$
  - $B \rightarrow ab \mid aAb$

Methods for Transforming Grammars...

Removing of Unit Rules

- Rules having this form $A \rightarrow B$ are called **unit rules.**

- Consider the rules,
  - $A \rightarrow aA \mid a \mid B$
  - $B \rightarrow bB \mid b \mid C$

- The unit rule $A \rightarrow B$ indicates that any string derivable from B is also derivable from A.

- The **removal of unit** rules *increases the number of rules* but *reduces the length of derivations*.

Methods for Transforming Grammars…
Removing of Unit Rules

- To eliminate the unit rule, add *A* rules that directly generate the same strings as *B.*
  - Add a rule $A \rightarrow u$ for each $B \rightarrow u$ and deleting $A \rightarrow B$ from the grammar.

$$\begin{array}{l} A \rightarrow B \\ B \rightarrow \alpha \mid \ldots \end{array} \quad \Longrightarrow \quad \begin{array}{l} A \rightarrow \alpha \mid \ldots \\ B \rightarrow \alpha \mid \ldots \end{array}$$

Methods for Transforming Grammars...

Removing of Unit Rules

- Consider the rules,
  - $A \rightarrow aA \mid a \mid B$
  - $B \rightarrow bB \mid b \mid C$

- The new rules after eliminating the unit rule $A \rightarrow B$.
  - $A \rightarrow aA \mid a \mid bB \mid b \mid C$
  - $B \rightarrow bB \mid b \mid C$

- We add new rules to A by replacing B in A with all its *RHS* rules.

Methods for Transforming Grammars...

Removing of Unit Rules

- G_L: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \varepsilon$

  $A \rightarrow aAa \mid aa \mid B \mid C$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$

- The new equivalent grammar (without unit rules)
- G_C: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid$

  $aAa \mid aa \mid bB \mid b \mid cC \mid c \mid \varepsilon$

  $A \rightarrow aAa \mid aa \mid bB \mid b \mid cC \mid c$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$

Methods for Transforming Grammars…

Removing of Unit Rules

- Remove unit rules:
  - S → T | S + T
  - T → F | F * T
  - F → a | (S)

  - S → T | S + T
  - T → a | (S) | F * T
  - F → a | (S)

  - S → a | (S) | F * T | S + T
  - T → a | (S) | F * T
  - F → a | (S)

Chomsky Normal Form (CNF)

- The Chomsky normal form **places restrictions** on the *length* and the *composition* of the **right-hand side** of a rule.

Definition

- A CFG is in **Chomsky normal form** if each production rule has one of the following forms:
  - $A \rightarrow a$
  - $A \rightarrow BC$
  - $S \rightarrow \varepsilon$
  - where $B, C \in$ NT

Chomsky Normal Form…
Converting CFG to CNF

Algorithm Step 1

- Make sure that the following are satisfied:
  - No ε-productions (other than $S \to \varepsilon$)
  - No chain rules
  - No useless symbols

Chomsky Normal Form…
Converting CFG to CNF

Algorithm Step 2

- Eliminate terminals from RHS of productions.
  - For each production $A \rightarrow X_1 X_2 \ldots X_m$
    - where $X_i \in NT \cup \mathcal{T}$
  - If $m > 1$, replace each **terminal** $a \in$ RHS of A
    - Add (if needed) $C_a \rightarrow a$ for each $a \in \mathcal{T}$, where each C_a is new non-terminal.
    - In production A, replace terminal a with corresponding C_a.

Chomsky Normal Form…
Converting CFG to CNF

Algorithm Step 3

- Eliminate productions with long RHS:
  - For each production:
    - $A \rightarrow B_1B_2 \ldots B_m$, $m > 2$, where $B_i \in NT$
  - replace with productions
    - $A \rightarrow B_1D_1$
    - $D_1 \rightarrow B_2D_2$
    - …
    - $D_{m\text{-}2} \rightarrow B_{m\text{-}1}B_m$
  - where $D_1 \ldots D_{m\text{-}2}$ are new non-terminals.

Chomsky Normal Form…
Converting CFG to CNF: **Examples**

1. Original grammar (no chain rules, useless symbols, or ε-productions):
   $S \rightarrow X\ a\ Y \mid Y\ b$
   $X \rightarrow Y\ X\ a\ Y \mid a$
   $Y \rightarrow S\ S \mid a\ X \mid b$

2. Grammar after eliminating terminals from RHSs:
   $S \rightarrow X\ A\ Y \mid Y\ B$ $\quad A \rightarrow a$
   $X \rightarrow Y\ X\ A\ Y \mid a$ $\quad B \rightarrow b$
   $Y \rightarrow S\ S \mid A\ X \mid b$

3. Grammar after eliminating long RHSs:
   $S \rightarrow X\ T \mid Y\ B$ $\quad T \rightarrow A\ Y$ $\quad A \rightarrow a$
   $X \rightarrow Y\ F \mid a$ $\quad F \rightarrow X\ G$ $\quad B \rightarrow b$
   $Y \rightarrow S\ S \mid A\ X \mid b$ $\quad G \rightarrow A\ Y$

Note: Could simplify by combining redundant variables *T* and *G.*

Chomsky Normal Form…
Converting CFG to CNF

1. Original grammar (no chain rules, useless symbols, or ε-productions):

   $S \to aXYZ \mid a$ $\quad X \to aX \mid a$

   $Y \to bcY \mid bc$ $\quad Z \to cZ \mid c$

2. Grammar after eliminating terminals from RHSs:

   $S \to AXYZ \mid a$ $\quad A \to a$

   $X \to AX \mid a$ $\quad B \to b$

   $Y \to BCY \mid BC$ $\quad C \to c$

   $Z \to CZ \mid c$

3. Grammar after eliminating long RHSs:

   $S \to AF \mid a$ $\quad A \to a$ $\quad F \to XG$

   $X \to AX \mid a$ $\quad B \to b$ $\quad G \to YZ$

   $Y \to BH \mid BC$ $\quad C \to c$ $\quad H \to CY$

   $Z \to CZ \mid c$

Greibach Normal Form (GNF)

- A context-free grammar is in **Greibach Normal Form** if every production is of the form $A \rightarrow aX$.
  - where $A \in \text{NT}$, $X \in \text{NT}^*$, and $a \in \Sigma$.

- Examples:
  - $G_1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aSA \mid a, A \rightarrow aA \mid b\})$
    - **GNF**
  - $G_2 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow AS \mid AAS, A \rightarrow SA \mid aa\})$
    - **not GNF**
- This grammar $S \rightarrow AB \quad A \rightarrow aA \mid bB \mid b \quad B \rightarrow b$
  - is **not in GNF**
- This grammar

$$S \rightarrow aAB \mid bBB \mid bB$$
$$A \rightarrow aA \mid bB \mid b$$
$$B \rightarrow b$$

  - is **in GNF**

Reading (Self-Study)

- Conversion from CFG to GNF.