Pharos University in Alexandria
Faculty of Computer Science & Artificial Intelligence
Course Title: Theory of Computation
Code: CS 307

Theory of Computation

Lecturer: Sherine Shawky

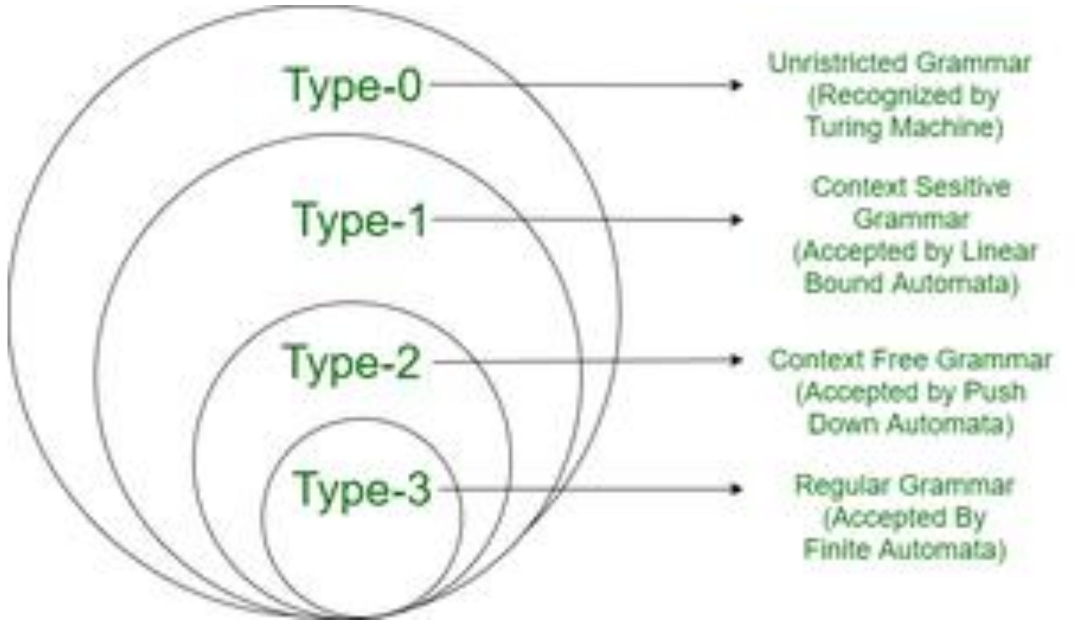
Text Books

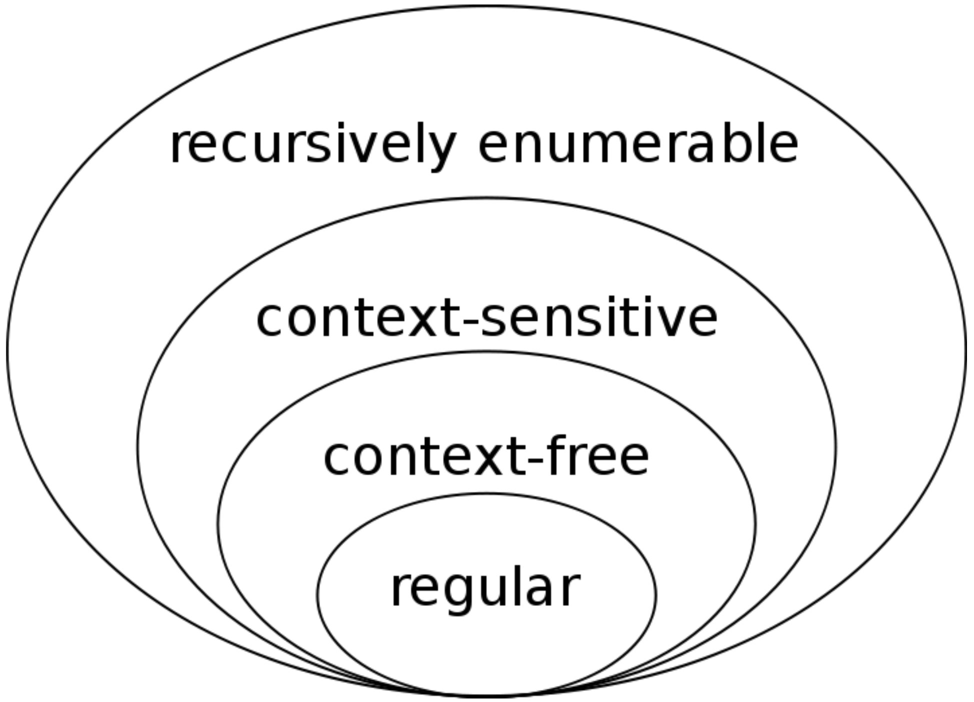
1. Introduction to formal languages and automata, Peter Linz, 6th edition, 2017.

Week 9

Context Free Language

Languages


Type-0
Unrristricted Grammar (Recognized by Turing Machine)
Type-1
Context Sesitive Grammar (Accepted by Linear Bound Automata)
Type-2
Context Free Grammar (Accepted by Push Down Automata)
Type-3
Regular Grammar (Accepted By Finite Automata)


recursively enumerable
context-sensitive
context-free
regular

Context Free Language (CFL)

- A context-free language is a language generated by a context free grammar (CFG).
- They are more general (and include) regular languages.
- All regular languages are context-free languages, but not all context-free languages are regular
- The same CFL might be generated by multiple context-free grammars.
- The set of all CFLs is identical to the set of languages that are accepted by pushdown automata (PDA).
- An inputed language is accepted by a computational model if it runs through the model and ends in an accepting final state.

Context Free Language

- Here is an example of a language that is not regular but *is* context-free:

  $\{a^{n}b^{n} | n \geq 0\}$

  This is the language of all strings that have an equal number of a's and b's.

- In this notation, a^4b^4 can be expanded out to aaaabbbb, where there are four a's and then four b's. (So this isn't exponentiation, though the notation is similar).

Closure Properties

- Context-free languages have the following closure properties.
- A set is closed under an operation if doing the operation on a given set always produces a member of the same set.
- This means that if one of these closed operations is applied to a context-free language the result will also be a context-free language
- **Union**:
  - Context-free languages are closed under the union operation.
  - This means that
    - if L and P are both context-free languages, then
    - L∪P is also a context-free language.

Closure Properties

- **Concatenation**:
- If L and P*L* and *P* are both context-free languages, then LP*LP* is also context free.
- The concatenation of a string is defined as follows:

S1S2=vw:v∈S1 and w∈S2*S*1*S*2=*vw*:*v*∈*S*1 and *w*∈*S*2.

Closure Properties

- **Kleene Star**:
- If LL is a context-free language, then L∗$L*$ is also context free.
- The Kleene star can repeat the string or symbol it is attached to any number of times (including zero times).
- The Kleene star basically performs a recursive concatenation of a string with itself.
- For example,

{a,b}∗={ϵ,a,b,ab,aab,aaab,abb⋯}$\{a,b\}*=\{\epsilon,a,b,ab,aab,aaab,abb\cdots\}$

and so on.

RL and CFL

- Regular Language And Context-Free Language are two important concepts in formal language theory.
- Both are classes of formal languages that are used to describe sets of strings that can be generated by a set of rules or symbols.
- A Regular Language is a language that can be generated by a regular expression or a finite-state machine.
- These languages are characterized by their simple grammatical rules, which can be expressed using only basic operations such as concatenation, alternation, and Kleene closure.
- On the other hand, a context-free language is a language that can be generated by context-free grammar.
- These languages are characterized by their more complex grammatical rules, which allow for the creation of nested structures and the use of recursion.
- more expressive and powerful than regular languages and are used in a wider range of applications in computer science and natural language processing.

Grammars

- In formal language theory, a language is defined as a set of strings of symbols that may be constrained by specific rules.
- Similarly, the written English language is made up of groups of letters (words) separated by spaces.
- A valid (accepted) sentence in the language must follow particular rules, the grammar.

Context Free Grammar

- Classification of Context Free Grammar is done on the basis of the number of parse trees.
- Only one parse tree->Unambiguous.
- More than one parse tree->Ambiguous.
- Productions are in the form –
- A->B;
- A∈N i.e A is a non-terminal.
- B∈V*(Any string).

Context Free Grammar

- Example
- S –> AB
- A –> a
- B –> b

Regular Grammar

- Productions are in the form –

- V –> VT / T (left-linear grammar)
- (or)
- V –> TV /T (right-linear grammar)
- Example –

- 1. S –> ab.
- 2. S -> aS | bS | ∊

Difference Between CFG and RG:

Parameter	**Context Free Grammar**	**Regular Grammar**
Type	Type-2	Type-3
Recognizer	Push-down automata. (more powerful computational model)	Finite State Automata
Rules	Productions are of the form: A->B; A∈N(Non-Terminal) B∈V*(Any string)	Productions are of the form: V –> VT / T (left-linear grammar) (or) V –> TV /T (right-linear grammar)
Restriction	Less than Regular Grammar	More than any other grammar
Right-hand Side	The right-hand side of production has no restrictions.	The right-hand side of production should be either left linear or right linear.

Difference Between CFG and RG:

Parameter	**Context Free Grammar**	**Regular Grammar**
level of expressiveness	More expressive (allow creation of more complex structures)	Less expressive (generate a limited set of simple structures)
Grammatical rules	More flexible	Less flexible
Set Property	Super Set of Regular Grammar	Subset of Context Free Grammar
Intersection	Intersection of two CFL need not be a CFL	Intersection of two RG is a RG.
Complement	They are not closed under complement	Closed under complement
Range	The range of languages that come under CFG is wide.	The range of languages that come under RG is less than CFG.
Examples	S –> AB;A –> a;B –> b	S -> aS | bS | ∊

Grammars

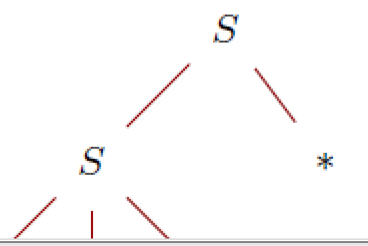
- A grammar G <N, Σ ,P, S> consists of the following components:
  - A finite set N of non terminal symbols or variables.
  - A finite set Σ of terminal symbols that are disjoint from N.
  - A finite set P of production rules of the form
- (Σ U N)* N (Σ U N)* -> (Σ U N)* where * is the Kleene star operator and U denotes the set union.
- Each production rule maps from one string of symbols to another where the left hand side contains at least one non terminal symbol.
- A distinguished start symbol S ∈ N.

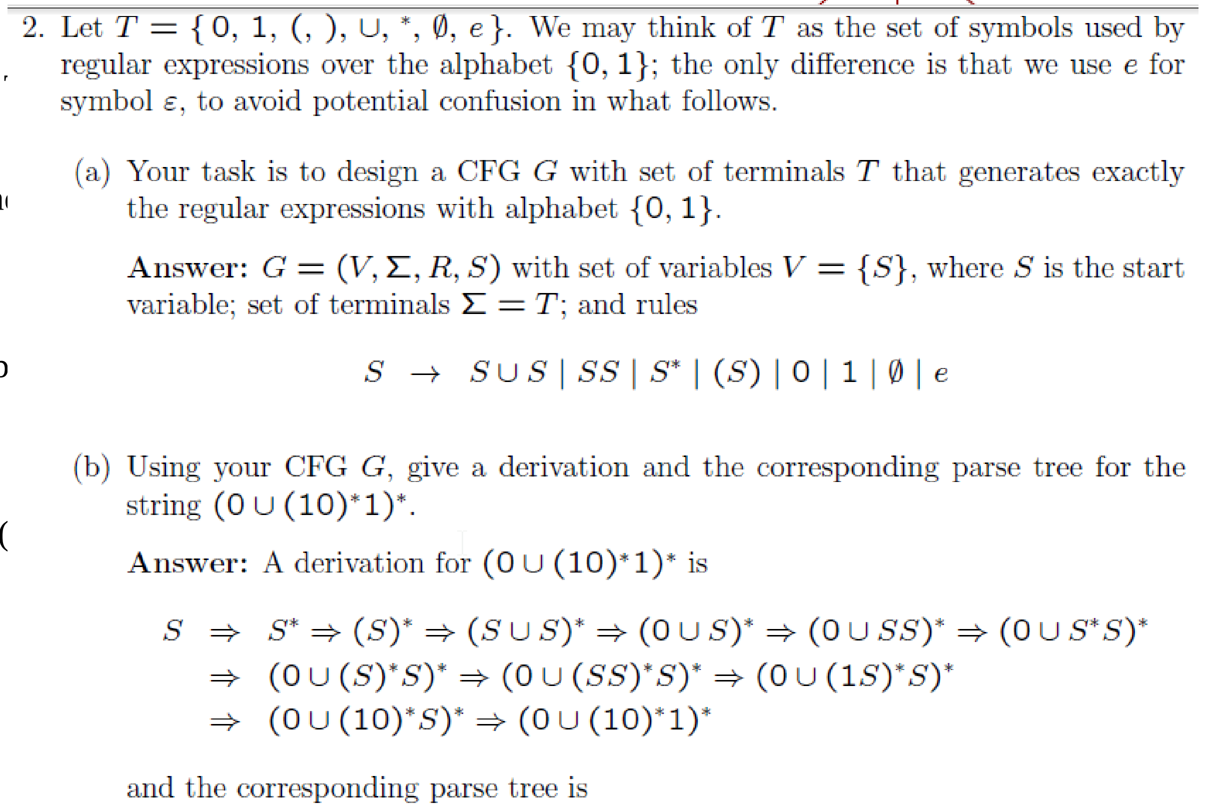
Grammars

- A language is said to be a regular language if it is generated by a RG.
- A grammar is said to be regular if it's either right-linear or left-linear.
- Specifically, a grammar G <N, Σ ,P, S> is said to be right-linear if each of its production rules is either of the form A -> xB or of the form A -> x, where A and B are non terminal symbols in N and x is a string of terminal symbols in Σ*.
- Similarly, it is left-linear if each of its production rules is either of the form A -> Bx or of the form A -> x, where A and B are non terminal symbols in N and x is a string of terminal symbols in Σ*.
- A language is said to be context-free if it is generated by a CFG. A grammar G <N, Σ, P, S> is context-free if the production rules are of the form N -> (N U Σ)*.
- Unlike RGs, the right hand side of the production rules in CFGs are unrestricted and can be any combination of terminals and non terminals.
- Regular languages are subsets of context free languages.

Context Free Language

- 2. Let T = { 0, 1, (,), ∪, ∗, ∅, e }. We may think of T as the set of symbols used by
- regular expressions over the alphabet {0, 1}; the only difference is that we use e for
- symbol ", to avoid potential confusion in what follows.
- (a) Your task is to design a CFG G with set of terminals '
- the regular expressions with alphabet {0, 1}.
- Answer: G = (V,,R, S) with set of variables V = {S}, wh
- variable; set of terminals = T ; and rules
- S → S ∪ S | SS | S∗ | (S) | 0 | 1 | ∅ | e
- (b) Using your CFG G, give a derivation and the corresp
- string (0 ∪ (10)∗1)∗.
- Answer: A derivation for (0 ∪ (10)∗1)∗ is
- S ⇒ S∗ ⇒ (S)∗ ⇒ (S ∪ S)∗ ⇒ (0 ∪ S)∗ ⇒ (0 ∪ SS)∗ ⇒ (
- ⇒ (0 ∪ (S)∗S)∗ ⇒ (0 ∪ (SS)∗S)∗ ⇒ (0 ∪ (1S)∗S)∗
- ⇒ (0 ∪ (10)∗S)∗ ⇒ (0 ∪ (10)∗1)∗
- and the corresponding parse tree is
- 3





2. Let $T = \{0, 1, (,), \cup, {}^*, \emptyset, e\}$. We may think of T as the set of symbols used by regular expressions over the alphabet $\{0, 1\}$; the only difference is that we use e for symbol ε, to avoid potential confusion in what follows.

(a) Your task is to design a CFG G with set of terminals T that generates exactly the regular expressions with alphabet $\{0, 1\}$.

Answer: $G = (V, \Sigma, R, S)$ with set of variables $V = \{S\}$, where S is the start variable; set of terminals $\Sigma = T$; and rules

$$S \rightarrow S \cup S \mid SS \mid S^* \mid (S) \mid 0 \mid 1 \mid \emptyset \mid e$$

(b) Using your CFG G, give a derivation and the corresponding parse tree for the string $(0 \cup (10)^*1)^*$.

Answer: A derivation for $(0 \cup (10)^*1)^*$ is

$$\begin{aligned} S &\Rightarrow S^* \Rightarrow (S)^* \Rightarrow (S \cup S)^* \Rightarrow (0 \cup S)^* \Rightarrow (0 \cup SS)^* \Rightarrow (0 \cup S^*S)^* \\ &\Rightarrow (0 \cup (S)^*S)^* \Rightarrow (0 \cup (SS)^*S)^* \Rightarrow (0 \cup (1S)^*S)^* \\ &\Rightarrow (0 \cup (10)^*S)^* \Rightarrow (0 \cup (10)^*1)^* \end{aligned}$$

and the corresponding parse tree is

Context Free Grammar

- A formal grammar called context free grammar (CFG) is used to produce every conceivable string in a given formal language.
- Four tuples are used to define the context free grammar G: G = (V, T, P, S)

Here,

  - G refers to a grammar that consists of sets of various production rules. We use it to generate a language's strings.
  - T refers to the terminal symbol's final set. Lower case letters are used to denote it.
  - V refers to the nonterminal symbol's final set. Capital letters are used to denote it.
  - P refers to a set of production rules that can be used to replace the nonterminal symbols (on the production's left side) in a string along with other terminals (present on the production's right side).
  - S refers to the start symbol that is used to derive the string.

- The start symbol is used in CFG to derive the string.
- This string can be derived by replacing a nonterminal repeatedly by the production's right-hand side, until and unless the terminal symbols replace all the nonterminals.

Classification of CFGs

- CFG is classified on the basis of the following two properties:
- **1) The Number of Generated Strings**
  - CFG is non-recursive if it produces a finite number of strings, or the grammar becomes a non-recursive grammar.
  - The grammar is complete if CFG can produce an endless amount of strings, repeating grammar.
  - The parser creates a derivation tree or a parse tree out of the source code during compilation by using the language’s grammar. There must be no ambiguity in the grammar. Parsing must not employ unclear grammar.
- **2) The Number of Derivation Trees**
  - If there is just one derivation tree, the CFG is clear/unambiguous.
  - If there are multiple derivation trees, the CFG is unclear/ambiguous.

Types of CFG

- **Examples for Recursive Grammars**

1) S->SxS

   S->y

   The set of strings (language) generated by the grammar given above would be: {y, yxy, yxyxy,…}, which is infinite.

2) S-> Xx

   X->Ay|z

   The generated language with the grammar given above is: {zx, zyx, zyyx …}, which is infinite.

- **Note:** The recursive CFG that does not consist of any useless rules would necessarily produce an infinite language.

Types of CFG

- **Example for Non-Recursive Grammars**

  S->Xx
  X->y|z
  By the above grammar, the language generated would be: {yx, zx}, which is finite.
- **Types of Recursive Grammars**
- A further division of a recursive CFG can be made based on the type of recursion in the grammar:
  - Recursive Left Grammar (that has left recursion)
  - Appropriate recursive grammar (that has the right recursion)
  - Recursive grammar in general (having general recursion)
- **Note:** A linear grammar is a CFG that produces sentences with not more than one non-terminal on the right side.

Context Free Grammar

- **Example:**
- Construct a CFG for the language L = $a^n b^{2n}$ where n>=1.

Context Free Grammar

- **Example 1:**
- Construct a CFG for the language L = $a^n b^{2n}$ where n>=1.
- **Solution:**
- The string that can be generated for a given language is {abb, aabbbb, aaabbbbbb....}.
- The grammar could be:
- $S \rightarrow aSbb \mid abb$
- Now if we want to derive a string "aabbbb", we can start with start symbols.
- $S \rightarrow aSbb$
- $S \rightarrow aabbbb$

Context Free Grammar

- **Example 2:**
- Construct a CFG for a language L = {wcwR | where w € (a, b)*}.

Context Free Grammar

- **Example 2:**
- Construct a CFG for a language L = {wcwR | where w € (a, b)*}.
- **Solution:**
- The string that can be generated for a given language is {aacaa, bcb, abcba, bacab, abbcbba, ...}
- The grammar could be:
- S → aSa   rule 1
- S → bSb   rule 2
- S → c   rule 3
- Now if we want to derive a string "abbcbba", we can start with start symbols.
- S → aSa
- S → abSba   from rule 2
- S → abbSbba   from rule 2
- S → abbcbba   from rule 3

Context Free Language

- **Example 3:**
- Construct a CFG for the regular expression (0+1)*

Context Free Language

- **Example 3:**
- Construct a CFG for the regular expression (0+1)*
- **Solution:**
- The CFG can be given by,
- Production rule (P):
- S → 0S | 1S
- S → ε
- The rules are in the combination of 0's and 1's with the start symbol. Since (0+1)* indicates {ε, 0, 1, 01, 10, 00, 11, ...}. In this set, ε is a string, so in the rule, we can set the rule S → ε.

Leftmost and Rightmost Derivations

- In a grammar that is not linear, a derivation may involve sentential
- forms with more than one variable. In such cases, we have a choice in
- the order in which variables are replaced. Take, for example, the
- grammar $G = (\{A, B, S\}, \{a, b\}, S, P)$ with productions
- 1. $S \rightarrow AB$.
- 2. $A \rightarrow aaA$.
- 3. $A \rightarrow \lambda$.
- 4. $B \rightarrow Bb$.
- 5 $B \rightarrow \lambda$.
- This grammar generates the language $L(G) = \{a2nbm : n \geq 0, m \geq 0\}$.
- Carry out a few derivations to convince yourself of this.

Leftmost and Rightmost Derivations

- Consider now the two derivations
- $S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$

- $S \Rightarrow AB \Rightarrow ABb \Rightarrow aaABb \Rightarrow aaAb \Rightarrow aab.$

Leftmost and Rightmost Derivations

- In order to show which production is applied, we have numbered the
- productions and written the appropriate number on the ⇒ symbol.
- From this we see that the two derivations not only yield the same
- sentence but also use exactly the same productions. The difference is
- entirely in the order in which the productions are applied. To remove
- such irrelevant factors, we often require that the variables be replaced in
- a specific order.

Leftmost and Rightmost Derivations

- DEFINITION 5.2
- A derivation is said to be leftmost if in each step the leftmost variable
- in the sentential form is replaced. If in each step the rightmost variable
- is replaced, we call the derivation rightmost.

Leftmost and Rightmost Derivations

- Consider the grammar with productions
- $S \to aAB$,
- $A \to bBb$,
- $B \to A|\lambda$.
- Then
- $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB \Rightarrow abbbbB \Rightarrow abbbb$
- is a leftmost derivation of the string *abbbb*. A rightmost derivation of
- the same string is
- $S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb \Rightarrow abbBbb \Rightarrow abbbb$.

Parsing Tree

- Parse : It means to resolve (a sentence) into its component parts and describe their syntactic roles or simply it is an act of parsing a string or a text.
- Tree: A tree may be a widely used abstract data type that simulates a hierarchical tree structure, with a root value and sub-trees of youngsters with a parent node, represented as a group of linked nodes.
- Rules to Draw a Parse Tree
  - All leaf nodes need to be terminals.
  - All interior nodes need to be non-terminals.
  - In-order traversal gives the original input string.

Parsing Tree

- Example 2: Let us take another example of Grammar (Production Rules).

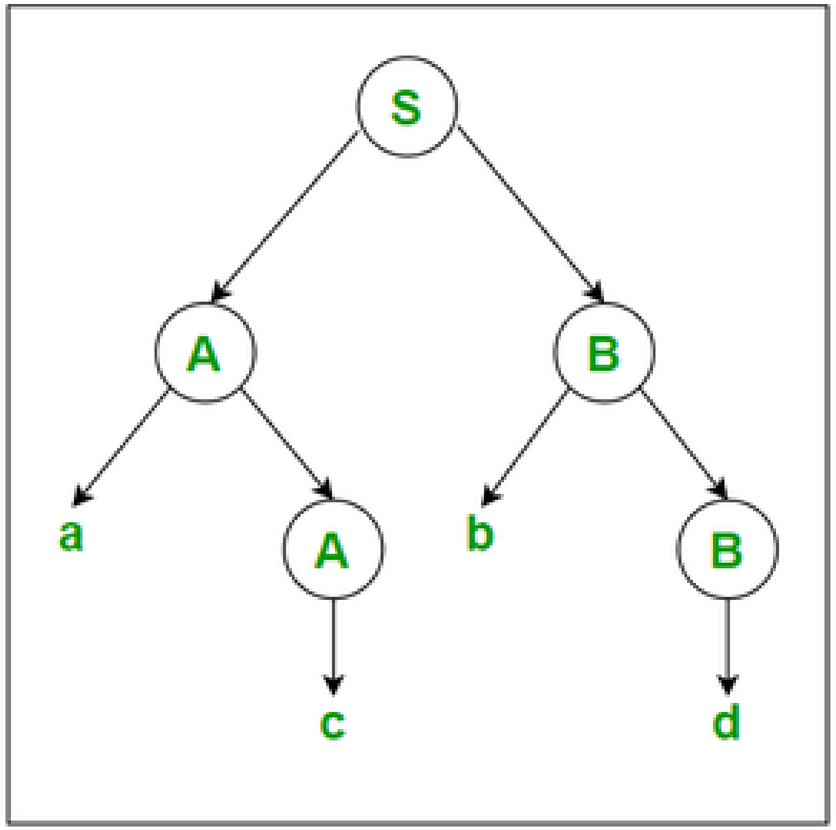
  S -> AB

  A -> c/aA

  B -> d/bB

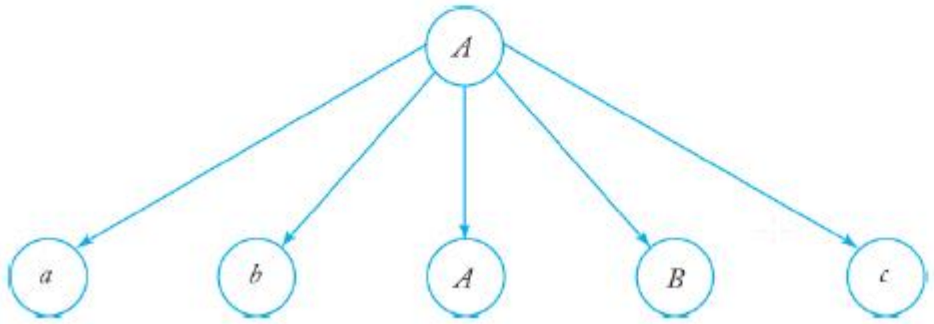
- The input string is “acbd”,

then the Parse Tree is as follows:



Parsing Tree

- Derivation Trees
- A second way of showing derivations, independent of the order in which
- productions are used, is by a derivation or parse tree. A derivation tree is
- an ordered tree in which nodes are labeled with the left sides of
- productions and in which the children of a node represent its
- corresponding right sides. For examp
- derivation tree representing the produ
- A → abABc.



Parsing Tree

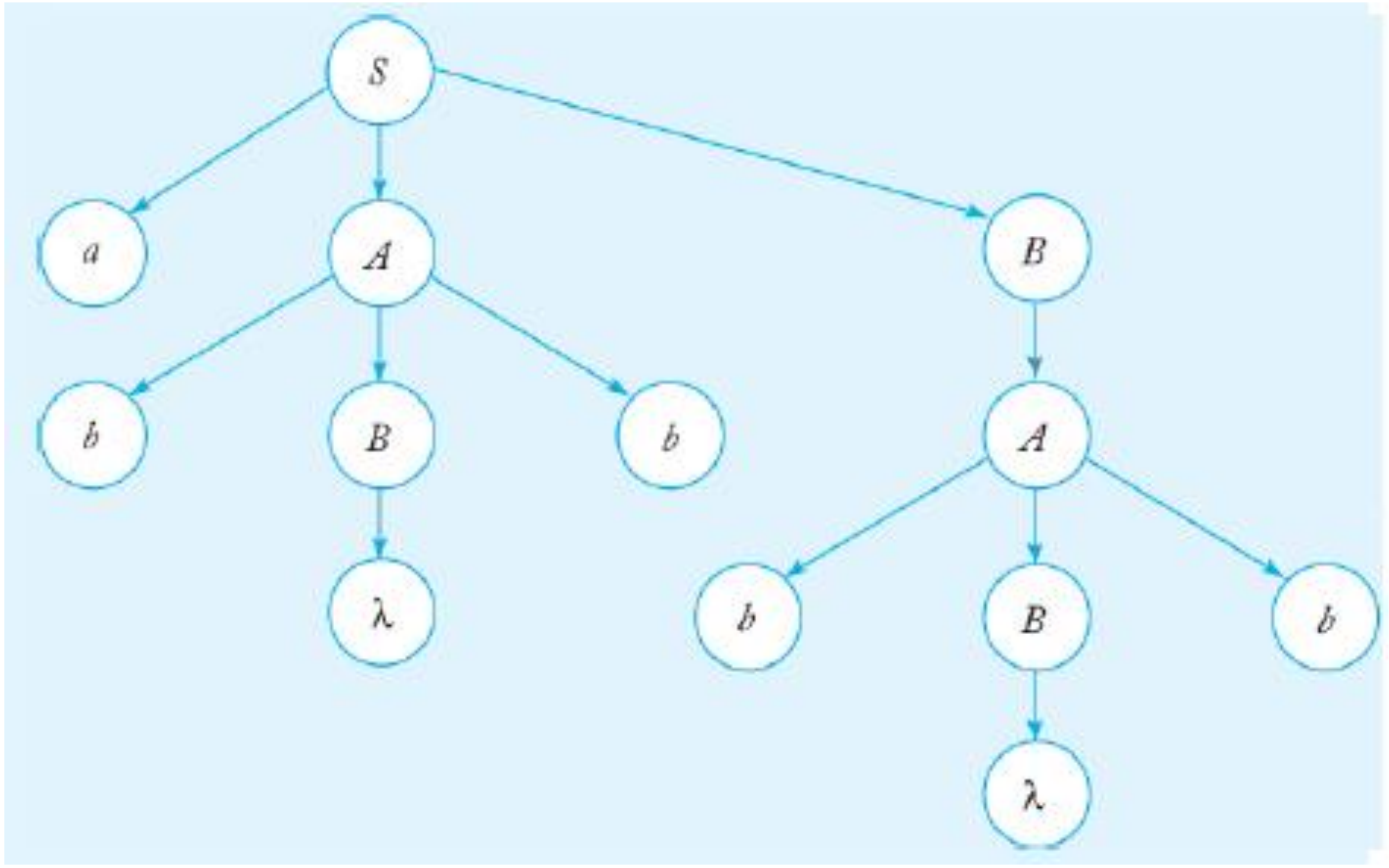
- DEFINITION 5.3
- Let $G = (V, T, S, P)$ be a context-free grammar. An ordered tree is a
- derivation tree for G if and only if it has the following properties.
- 1. The root is labeled S.
- 2. Every leaf has a label from $T \cup \{\lambda\}$.
- 3. Every interior vertex (a vertex that is not a leaf) has a label from V.
- 4. If a vertex has label $A \in V$, and its children are labeled (from left to
- right) $a1$,$a2$, ..., an, then P must contain a production of the form $A \longrightarrow$
- $a1a2 \cdots an$.

Parsing Tree

- 5. A leaf labeled λ has no siblings, that is, a vertex with a child labeled λ
- can have no other children.
- A tree that has properties 3, 4, and 5, but in which 1 does not
- necessarily hold and in which property 2 is replaced by 2a. Every leaf
- has a label from $V \cup T \cup \{\lambda\}$, is said to be a partial derivation tree.
- The string of symbols obtained by reading the leaves of the tree from
- left to right, omitting any λ's encountered, is said to be the yield of the
- tree. The descriptive term *left to right* can be given a precise meaning. The
- yield is the string of terminals in the order they are encountered when the
- tree is traversed in a depth-first manner, always taking the leftmost
- unexplored branch.

Parsing Tree

- The tree in the Figure is a derivation tree of the string *abBbB*,



Parsing and Ambiguity

- We have so far concentrated on the generative aspects of grammars.
- Given a grammar G, we studied the set of strings that can be derived using G.
- In cases of practical applications, we are also concerned with the analytical side of the grammar: Given a string w of terminals, we want to know whether or not w is in $L(G)$.
- If so, we may want to find a derivation of w.
- An algorithm that can tell us whether w is in $L(G)$ is a membership algorithm. The term parsing describes finding a sequence of productions by which a $w \in L(G)$ is derived.

Context Free Grammar

- Consider the grammar
- $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
- and the string $w = aabb$. Round one gives us
- 1. $S \Rightarrow SS$,
- 2. $S \Rightarrow aSb$,
- 3. $S \Rightarrow bSa$,
- 4. $S \Rightarrow \lambda$.
- The last two of these can be removed from further consideration for
- obvious reasons. Round two then yields sentential forms
- $S \Rightarrow SS \Rightarrow SSS$,
- $S \Rightarrow SS \Rightarrow aSbS$,
- $S \Rightarrow SS \Rightarrow bSaS$,
- $S \Rightarrow SS \Rightarrow S$,

Context Free Grammar

- which are obtained by replacing the leftmost S in sentential form 1 with
- all applicable substitutes. Similarly, from sentential form 2 we get the
- additional sentential forms
- $S \Rightarrow aSb \Rightarrow aSSb$,
- $S \Rightarrow aSb \Rightarrow aaSbb$,
- $S \Rightarrow aSb \Rightarrow abSab$,
- $S \Rightarrow aSb \Rightarrow ab$.
- Again, several of these can be removed from contention. On the next
- round, we find the actual target string from the sequence
- 1. $S \Rightarrow SS$,
- 2. $S \Rightarrow aSb$,
- 3. $S \Rightarrow bSa$,
- 4. $S \Rightarrow \lambda$.
- $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.
- Therefore, *aabb* is in the language generated by the grammar under
- consideration.

Context Free Grammar

- DEFINITION 5.4
- A context-free grammar $G = (V, T, S, P)$ is said to be a simple grammar or s-grammar if all its productions are of the form

$A \rightarrow ax$, where $A \in V$, $a \in T$, $x \in V^*$, and any pair (A, a) occurs at most once in P.

Context Free Grammar

- The grammar
- $S \to aS \mid bSS \mid c$

is an s-grammar. The grammar

- $S \to aS \mid bSS \mid aSS \mid c$

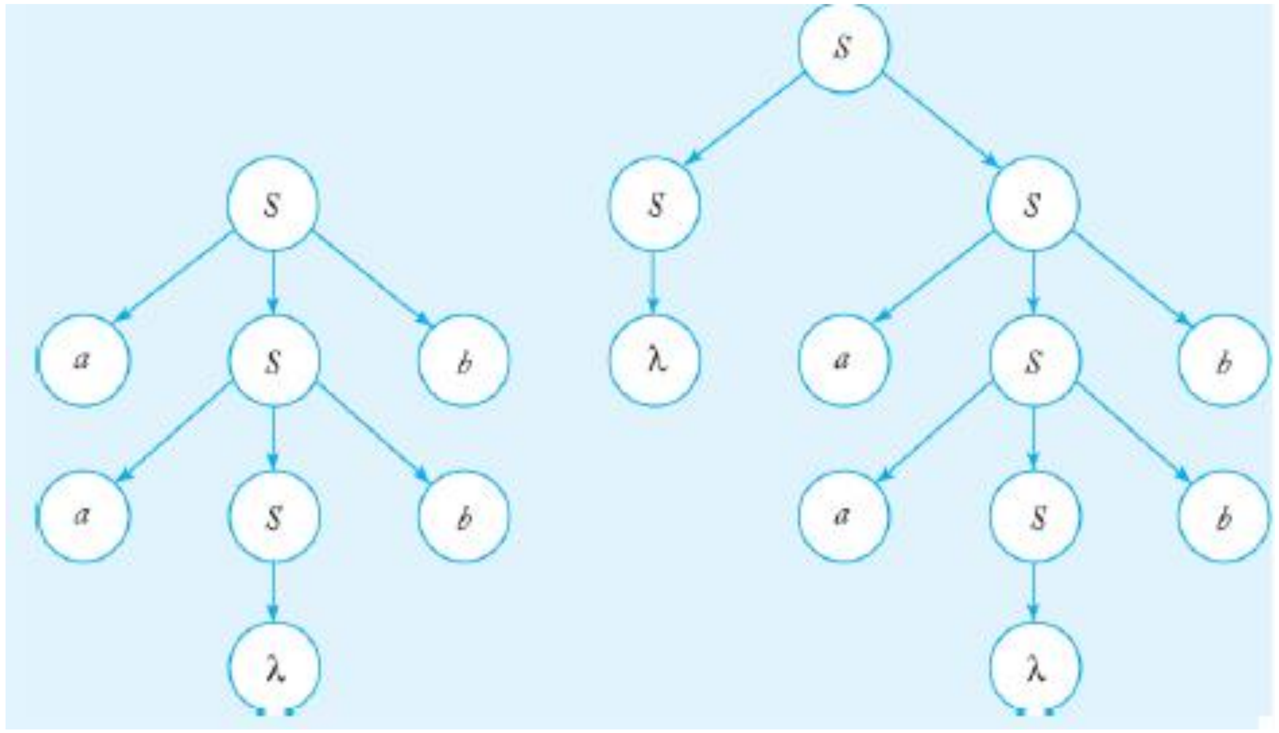
is not an s-grammar because the pair (S, a) occurs in the two productions $S \to aS$ and $S \to aSS$.

Context Free Grammar

- DEFINITION 5.5
- A context-free grammar G is said to be ambiguous if there exists some w ∈ L (G) that has at least two distinct derivation trees.
- Alternatively, ambiguity implies the existence of two or more leftmost or rightmost derivations.

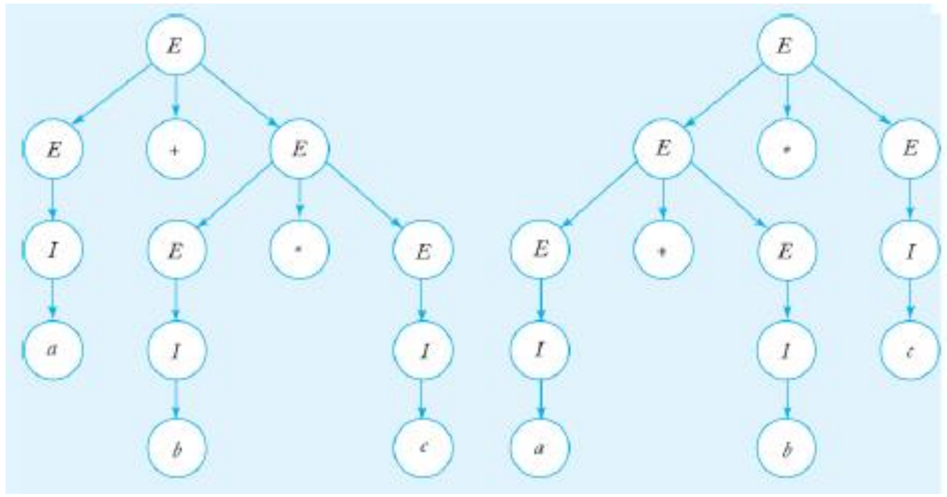
Context Free Grammar

- The grammar in with productions $S \rightarrow aSb \mid SS \mid \lambda$, is ambiguous.
- The sentence *aabb* has the two derivation trees shown in Figure.



Context Free Grammar

- Consider the grammar $G = (V, T, E, P)$ with
- $V = \{E, I\}$,
- $T = \{a, b, c, +, *, (,)\}$,
- and productions
- $E \longrightarrow I$,
- $E \longrightarrow E + E$,
- $E \longrightarrow E*E$,
- $E \longrightarrow (E)$,
- $I \longrightarrow a|b|c$.
- The strings $(a + b) *c$ and $a*b + c$ are in $L(G)$. It is easy to see that this
- grammar generates a restricted subset of arithmetic expressions for Clike
- programming languages. The grammar is ambiguous. For instance,
- the string $a + b*c$ has two different derivation trees, as shown in Figure
- 5.5.



Simplification

- THEOREM 6.1
- Let $G = (V, T, S, P)$ be a context-free grammar. Suppose that P contains a
- production of the form
- $A \longrightarrow x1Bx2$.
- Assume that A and B are different variables and that
- $B \longrightarrow y1 \mid y2 \mid \cdots \mid yn$
- is the set of all productions in P that have B as the left side. Let $\hat{G} = (V, T, S,$
- $\hat{}P$) be the grammar in which
- $\hat{}P$ is constructed by deleting
- $A \longrightarrow x1Bx2$ (6.1)
- from P, and adding to it
- $A \longrightarrow x1y1x2 \mid x1y2x2 \mid \cdots \mid x1ynx2$.
- Then
- $L(\hat{G}) = L(G)$.

Simplification

- 1. Removing Useless Productions
- 2. Unit Removal
- 3. Null Removal

Removing Useless Productions

- A variable is useless either because it cannot be reached from the start symbol or because it cannot derive a terminal string.
- A procedure for removing useless variables and productions is based on recognizing these two situations.

Removing Useless Productions

- **Example 1**
  - G:

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

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

Removing Useless Productions

- G_T: $S \longrightarrow BS \mid B$
  $A \longrightarrow aA \mid aF$
  $B \longrightarrow b$
  $E \longrightarrow aA \mid BSA$
  $F \longrightarrow bB \mid b$
- $S \in$ **REACH**, since it is the start symbol
  - $B \in$ REACH, since $S \longrightarrow SB$, 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

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.

Simplification

- **EXAMPLE 6.1**
- Consider $G = (\{A, B\}, \{a, b, c\}, A, P)$ with productions
- $A \longrightarrow a|aaA|abBc,$
- $B \longrightarrow abbA|b.$

Simplification

- **Solution**
- Using the suggested substitution for the variable *B*, we get the grammar $\hat{G}$ with productions
- $A \longrightarrow a|aaA|ababbAc|abbc,$
- $B \longrightarrow abbA|b.$
- The new grammar $\hat{G}$ is equivalent to *G*. The string *aaabbc* has the derivation
- $A \Rightarrow aaA \Rightarrow aaabBc \Rightarrow aaabbc$
- in *G*, and the corresponding derivation
- $A \Rightarrow aaA \Rightarrow aaabbc$
- in $\hat{G}$.
- Notice that, in this case, the variable *B* and its associated productions are still in the grammar even though they can no longer play a part in any derivation. We will next show how such unnecessary productions can be removed from a grammar.

Context Free Language

- Eliminate useless symbols and productions from $G = (V, T, S, P)$,
- where $V = \{S, A, B, C\}$ and $T = \{a, b\}$, with P consisting of
- $S \longrightarrow aS \,|A|\, C,$
- $A \longrightarrow a,$
- $B \longrightarrow aa,$
- $C \longrightarrow aCb.$
- First, we identify the set of variables that can lead to a terminal
- string. Because $A \longrightarrow a$ and $B \longrightarrow aa$, the variables A and B belong to
- this set. So does S, because $S \Rightarrow A \Rightarrow a$. However, this argument
- cannot be made for C, thus identifying it as useless. Removing C and
- its corresponding productions, we are led to the grammar $G1$ with
- variables $V1 = \{S, A, B\}$, terminals $T = \{a\}$, and productions

Context Free Language

- $S \longrightarrow aS|A,$
- $A \longrightarrow a,$
- $B \longrightarrow aa.$
- Next we want to eliminate the variables that cannot be reached
- from the start variable. For this, we can draw a dependency graph
- for the variables. Dependency graphs are a way of visualizing
- complex relationships and are found in many applications. For
- context-free grammars, a dependency graph has its vertices labeled
- with variables, with an edge between vertices C and D if and only if
- there is a production of the form
- $C \longrightarrow xDy.$

Context Free Language

- THEOREM 6.2
- Let $G = (V, T, S, P)$ be a context-free grammar. Then there exists an
- equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ that does not contain
- any useless variables or productions.

Removing λ-Productions

- DEFINITION 6.2
- Any production of a context-free grammar of the form $A \rightarrow \lambda$
- is called a λ-production. Any variable A for which the derivation
- $A \overset{*}{\Rightarrow} \lambda$ (6.3)
- is possible is called nullable.
- A grammar may generate a language not containing λ, yet have some
- λ-productions or nullable variables. In such cases, the λ-productions
- can be removed.

Context Free Language

- Example
- Consider the grammar
- $S \rightarrow aS1b$,
- $S1 \rightarrow aS1b|\lambda$, with start variable S. This grammar generates
- the λ-free language $\{anbn : n \geq 1\}$. The λ-production $S1 \rightarrow \lambda$
- can be removed after adding new productions obtained by
- substituting λ for $S1$ where it occurs on the right. Doing this
- we get the grammar $S \rightarrow aS1b|ab$, $S1 \rightarrow aS1b|ab$.
- We can easily show that this new grammar generates the same
- language as the original one.
- In more general situations, substitutions for λ-productions can be
- made in a similar, although more complicated, manner.

Context Free Language

- **Example**
- Find a context-free grammar without λ-productions equivalent to the grammar defined by
    - $S \longrightarrow ABaC,$
    - $A \longrightarrow BC,$
    - $B \longrightarrow b|\lambda,$
    - $C \longrightarrow D|\lambda,$
    - $D \longrightarrow d.$

Context Free Language

- **Solution**
- From the first step of the construction in Theorem 6.3, we find that
- the nullable variables are A, B, C. Then, following the second step of
- the construction, we get
- $S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a,$
- $A \rightarrow B \mid C \mid BC,$
- $B \rightarrow b,$
- $C \rightarrow D,$
- $D \rightarrow d.$

Removing Unit-Productions

- As we have seen in Theorem 5.2, productions in which both sides are a
- single variable are at times undesirable.
- DEFINITION 6.3
- Any production of a context-free grammar of the form $A \rightarrow B$,
- where $A, B \in V$, is called a unit-production.
- To remove unit-productions, we use the substitution rule discussed in
- Theorem 6.1. As the construction in the next theorem shows, this can
- be done if we proceed with some care.
- THEOREM 6.4
- Let $G = (V, T, S, P)$ be any context-free grammar without λ-
- productions. Then there exists a context-free grammar
- $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ that does not have any unit-productions
- and that is equivalent to G.

Removing Unit-Productions

- Example

  Remove all unit-productions from

  - $S \rightarrow Aa|B,$
  - $B \rightarrow A|bb,$
  - $A \rightarrow a\ |bc|\ B.$

Removing Unit-Productions

Example

Remove all unit-productions from

- $S \longrightarrow Aa|B,$
- $B \longrightarrow A|bb,$
- $A \longrightarrow a\ |bc|\ B.$

Solution

- Hence, we add to the original non-unit productions
- $S \longrightarrow Aa,$
- $A \longrightarrow a|bc,$
- $B \longrightarrow bb,$

Removing Unit-Productions

- the new rules
- $S \longrightarrow a\ |bc|\ bb,$
- $A \longrightarrow bb,$
- $B \longrightarrow a|bc,$
- to obtain the equivalent grammar
- $S \longrightarrow a\ |bc|\ bb|Aa,$
- $A \longrightarrow a\ |bb|\ bc,$
- $B \longrightarrow a\ |bb|\ bc.$
- Note that the removal of the unit-productions has made B and the associated productions useless.