Context-Free Grammars

- Many useful languages are not regular
- Context-free grammars are very useful for the definition and processing of programming languages
- A context-free grammar has no restrictions on the right side of its productions, while the left side must be a single variable
- A *language* is context-free if it is generated by a context-free grammar
- Since regular grammars are context-free, the family of regular languages is a proper subset of the family of context-free languages

Leftmost and Rightmost Derivations

In a *leftmost derivation*, the leftmost variable in a sentential form is replaced at each step

In a *rightmost derivation*, the rightmost variable in a sentential form is replaced at each step

Consider the grammar from example 5.5:

V = { S, A, B }, T = { a, b }, and productions

$S \rightarrow aAB$

$A \rightarrow bBb$

$B \rightarrow A \mid \lambda$

The string abb has two distinct derivations:

Leftmost: $S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abbB \Rightarrow abb$

Derivation Trees

- In a derivation tree or parse tree,

  -the root is labeled S

  -internal nodes are labeled with a variable occurring

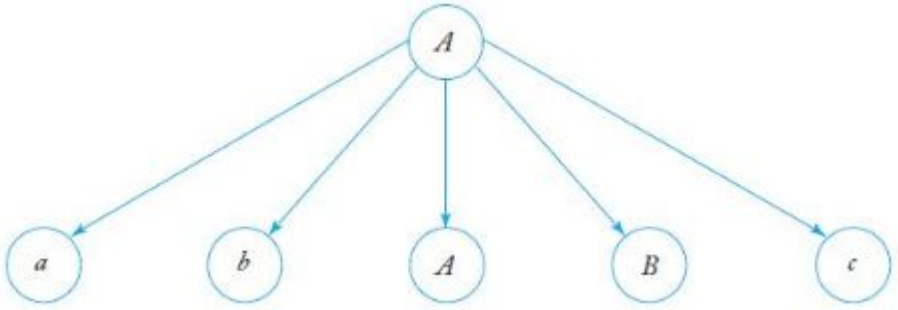
  on the left side of a production

  -the children of a node contain the symbols on the

  corresponding right side of a production

The yield of a derivation tree is the string of terminals produced by a leftmost depth-first traversal of the tree

For ex → abABc, Figure shows the n tree

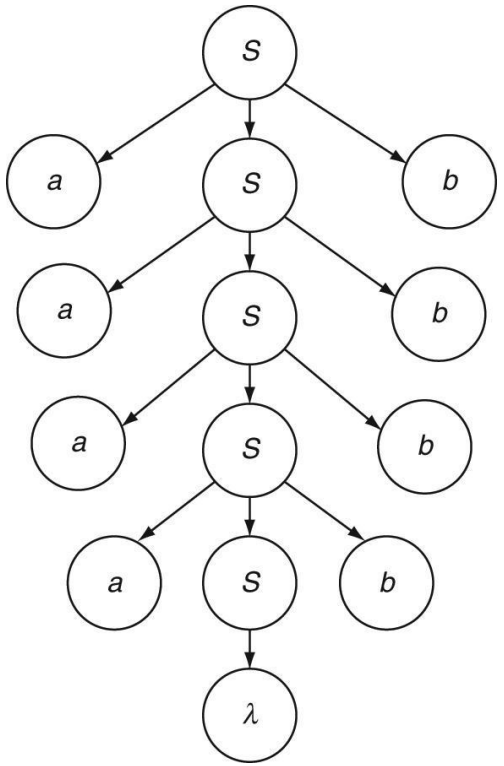


- Simple nesting language $a^n b^n$
- Observation: $a^n b^n = aa^{n-1} b^{n-1} b$

$$S \rightarrow aSb \mid \lambda$$

- Derivation for $a^4 b^4$
- Parse tree:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaa\lambda bbbb = aaaabbbb$$



- Even palindromes ww^R
- $S \rightarrow aSa \mid bSb \mid \lambda$
- Derivation of: *abbabbabba*

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbaSabba \Rightarrow$$

$$abbabSbabba \Rightarrow abbab\lambda babba = abbabbabba$$

- Odd palindromes wcw^R

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

- One more b than a, i.e., $a^n b^{n+1}$

$S \rightarrow aSb \mid b$

- a s followed by twice as many b s, i.e., $a^n b^{2n}$

$S \rightarrow aSbb \mid \lambda$

- a s followed by a lesser number of b s, i.e., $a^n b^m$ n > m

$S \rightarrow aSb \mid aS \mid a$

- Unequal numbers of *a* s and *b* s

$S \rightarrow A \mid B$

$A \rightarrow aAb \mid aA \mid a$

$B \rightarrow aBb \mid Bb \mid b$

- Number of *a* s not equal to twice the number of *b* s

$S \rightarrow A \mid B \mid aD$

$A \rightarrow aA \mid aC$

$B \rightarrow Bb \mid Cb$

$C \rightarrow aaCb \mid \lambda$

$D \rightarrow Db \mid \lambda$

- Language of addition $a^n b^m c^k$ where k = n + m
- $a^n b^m c^k = a^n b^m c^m c^n$

$S \rightarrow aSc \mid A$

$A \rightarrow bAc \mid \lambda$

- Equal numbers of *a* and *b* in any order ($n_a = n_b$)

$$S \rightarrow aSb \mid bSa \mid SS \mid \lambda$$

- Example: *babbabaaaababb*
- Derivation:

$$S \Rightarrow SS \Rightarrow bSaS \Rightarrow b\lambda aS \Rightarrow baSS \Rightarrow babSaS \Rightarrow babbSaaS \Rightarrow babbaSbaaS$$
$$\Rightarrow babba\lambda baaS \Rightarrow babbabaaaSb \Rightarrow babbabaaaaSbb \Rightarrow babbabaaaabSabb$$
$$\Rightarrow babbabaaaab\lambda abb = babbabaaaababb$$

- Count the difference:

(0)*b*(–1)*a*(0)*b*(–1)*b*(–2)*a*(–1)*b*(–2)*a*(–1)*a*(0)*a*(1)*a*(2)*b*(1)*a*(2)*b*(1)*b*(0)

Non-Linear CFG:Arithmetic Expression

Σ = { variable, constant, +, -, *, /, (,) }

$S \rightarrow S + S$

$S \rightarrow S * S$

$S \rightarrow S - S$

$S \rightarrow S / S$

$S \rightarrow (S)$

$S \rightarrow$ variable | constant

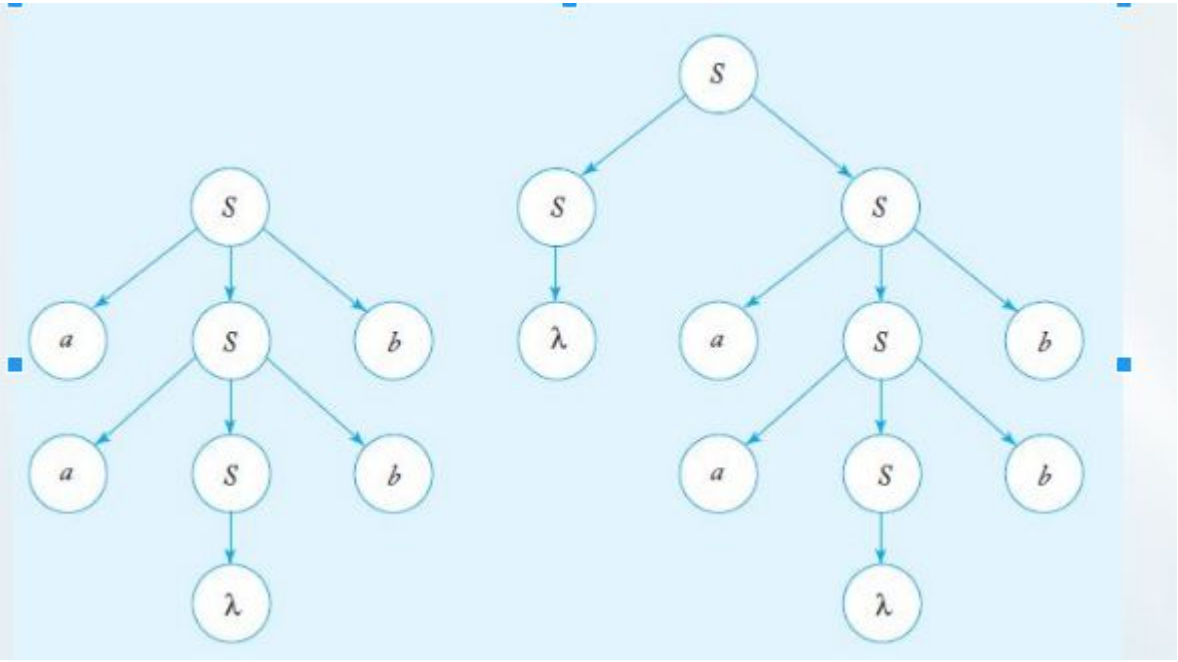
$(x + 2.0) * y / (z - 6.0)$

Derivation:

$$S \Rightarrow S * S \Rightarrow (S) * S \Rightarrow (S + S) * S \Rightarrow (x + S) * S \Rightarrow (x + 2.0) * S \Rightarrow$$
$$(x + 2.0) * S / S \Rightarrow (x + 2.0) * y / S \Rightarrow (x + 2.0) * y / (S) \Rightarrow (x + 2.0) * y / (S - S) \Rightarrow$$
$$(x + 2.0) * y / (z - S) \Rightarrow (x + 2.0) * y / (z - 6.0)$$

Parsing and Ambiguity

- A grammar G is *ambiguous* if there is some string w in L(G) for which more than one derivation tree exists
- The grammar with productions S → aSb | SS | λ is ambiguous, since the string aabb has two derivation trees, as shown below



Ambiguity in Expression Evaluation

- Σ = { variable, constant, +, -, *, /, (,) }

$S \rightarrow S + S$

$S \rightarrow S * S$

$S \rightarrow S - S$

$S \rightarrow S / S$

$S \rightarrow (S)$

$S \rightarrow$ variable | constant

- Example : *a* + *b*/*c* with two leftmost derivations;

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S / S \Rightarrow a + b / S \Rightarrow a + b / c$$

$$S \Rightarrow S / S \Rightarrow S + S / S \Rightarrow a + S / S \Rightarrow a + b / S \Rightarrow a + b / c$$

Eliminating Ambigutiy

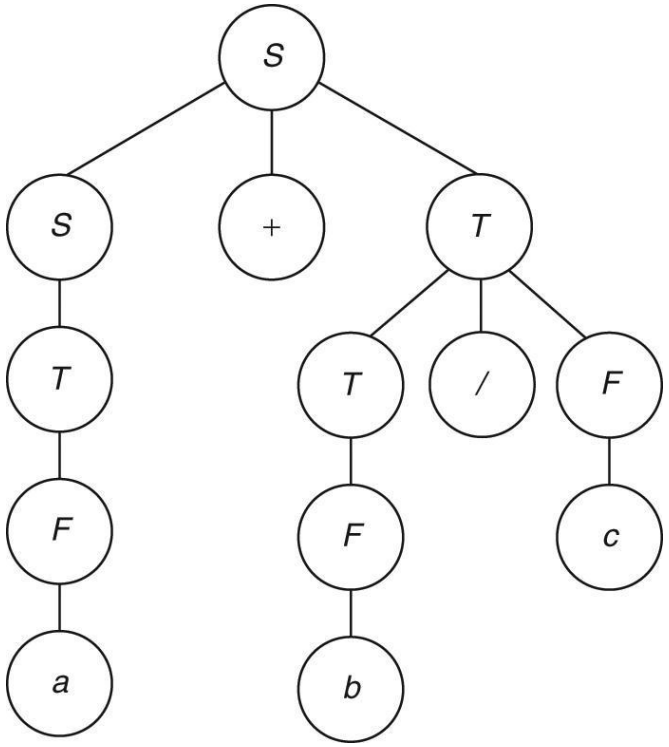
- Modified expression grammar
- Grammar

$$S \rightarrow (S) \mid S + T \mid S - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow \text{variable} \mid \text{constant}$$

- Single leftmost derivation



$$S \Rightarrow S + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow$$

$$a + T / F \Rightarrow a + F / F \Rightarrow a + b / F \Rightarrow a + b / c$$

Ambiguous Languages

- For some languages, it is always possible to find an unambiguous grammar, as shown in the previous example
- However, there are *inherently ambiguous* languages, for which every possible grammar is ambiguous
- Consider the language $\{ a^n b^n b^m \} \cup \{ a^n b^m b^m \}$, which is generated by the grammar

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow S_1 c \mid A$

$A \rightarrow aAb \mid \lambda$

$S_2 \rightarrow aS_2 \mid B$

$B \rightarrow bBc \mid \lambda$

- The grammar above (and every other equivalent

The idea of Chomsky normal form

- Normal forms clean up a grammar
- They make parsing efficient
- Any CFG can be converted to Chomsky Normal Form
- If the language includes the null string, it should be excluded
- Chomsky Normal Form: every production is of one of the two forms:
  - $A \rightarrow a$ Replaces a variable in the sentential form by a terminal symbol
  - $A \rightarrow BC$ Replaces a variable by two variables to grow the sentential form
- Idea: Binary parse tree
- Idea: Each step in derivation either does not change the length of the sentential form or grows it by 1

Cleaning up a Grammar

Grammars in normal forms cannot have

- *Lambda productions* of the form $A \rightarrow \lambda$. Variables having such productions are called *nullable variables*.
- *Unit productions* of the form $A \rightarrow B$ which merely replace one variable by another without making any progress in the derivation.
- *Useless variables* which are of two kinds:
  - *Non-generating* ones which never lead to terminal symbols. If a non-generating variable is introduced into the sentential form, it is a dead-end; the sentential form can never generate any string in the language since there is no way to get rid of the non-generating variable.
  - Productions for *unreachable variables.* An unreachable variable, on the other hand, does not appear on the right-hand side of any production (to make it reachable from the start symbol S) and hence will never appear in a sentential form.

Useless Productions

- A variable is *useful* if it occurs in the derivation of at least one string in the language
- Otherwise, the variable and any productions in which it appears is considered *useless*
- A variable is useless if:
  - No terminal strings can be derived from the variable
  - The variable symbol cannot be reached from S
- In the grammar below, B can never be reached from the start symbol S and is therefore considered useless

$S \rightarrow A$

$A \rightarrow aA \mid \lambda$

$B \rightarrow bA$

Removing Useless Productions

It is always possible to remove useless productions from a context-free grammar:

1. Let V_1 be the set of useful variables, initialized to empty
2. Add a variable A to V_1 if there is a production of the form

   A → terminal symbols or variables in V_1

   (Repeat until nothing else can be added to $V_{1)}$
3. Eliminate any productions containing variables not in V_1
4. Use a dependency graph to identify and eliminate variables that are unreachable from S

Application of the procedure for removing Useless Productions

- Consider the grammar:

  $S \rightarrow aS \mid A \mid C$

  $A \rightarrow a$

  $B \rightarrow aa$

  $C \rightarrow aCb$

- In step 2, variables A, B, and S are added to V_1
- Since C is useless, it is eliminated in step 3, resulting in the grammar with productions

  $S \rightarrow aS \mid A$

  $A \rightarrow a$

  $B \rightarrow aa$

- In step 4, B is identified as unreachable from S, resulting in the grammar with productions

Lambda Productions

- A production with λ on the right side is called a *λ-production*
- A variable A is called *nullable* if there is a sequence of derivations through which A produces λ
- If a grammar generates a language not containing λ, any λ-productions can be removed
- In the grammar below, S_1 is nullable

  $S \rightarrow aS_1b$

  $S_1 \rightarrow aS_1b \mid \lambda$
- Since the language is λ-free, we have the equivalent grammar

  $S \rightarrow aS_1b \mid ab$

Removing lambda productions

It is possible to remove λ-productions from a context-free grammar that does not generate λ:

1. Let V_N be the set of nullable variables, initialized to empty
2. Add a variable A to V_N if there is a production having one of the forms:
   - $A \rightarrow \lambda$
   - $A \rightarrow$ variables already in V_N

   (Repeat until nothing else can be added to $V_{N)}$
3. Eliminate λ-productions
4. Add productions in which nullable symbols are replaced by λ in all possible combinations

Application of the procedure for removing lambda productions

- Consider the grammar :

  $S \rightarrow ABaC$

  $A \rightarrow BC$

  $B \rightarrow b \mid \lambda$

  $C \rightarrow D \mid \lambda$

  $D \rightarrow d$

- In step 2, variables B, C, and A (in that order) are added to V_N
- In step 3, λ-productions are eliminated
- In step 4, productions are added by replacing nullable symbols with in λ all possible combinations, resulting in

Unit Productions

- A production of the form A $\rightarrow$ B (where A and B are variables) is called a *unit-production*
- Unit-productions add unneeded complexity to a grammar and can usually be removed by simple substitution
- The procedure for eliminating unit-productions assumes that all λ-productions have been previously removed

Removing Unit Productions

1. Draw a dependency graph with an edge from A to B corresponding to every A → B production in the grammar
2. Construct a new grammar that includes all the productions from the original grammar, except for the unit-productions
3. Whenever there is a path from A to B in the dependency graph, replace B using the substitution rule.

Application of the procedure for removing Unit productions

- Consider the grammar:

  S → Aa | B

  A → a | bc |B

  B → A | bb

  The dependency graph contains paths from S to A, S to B, B to A, and A to B

- After removing unit-productions and adding the new productions (in red), the resulting grammar is

  S → Aa | a | bc | bb

  A → a | bc | bb

  B → a | bc | bb

Converting CFG to Chomsky normal form

Algorithm

1. *Eliminate lambda productions:* If there is a lambda production $A \rightarrow \lambda$, delete the production after changing all other productions where A occurs on the right-hand side as follows:

   In a production $B \rightarrow xAy$, remove the nullable A by replacing it with:

   $B \rightarrow xy \mid xAy$ If there is some other production that expands A or just

   $B \rightarrow xy$ If there is no other production for A

2. *Eliminate unit productions:* For every unit production $A \rightarrow B$, delete the production by replacing any occurrence of A with B on the right-hand sides of all other productions, that is, replace

   $C \rightarrow xAy$ *by* $C \rightarrow xAy \mid xBy$

   (or just $C \rightarrow xBy$ if there is no other production for A).

3. *Eliminate useless variables:* Simply delete all productions containing unreachable and non-generating variables. A variable is useful if it is a part of the derivation of at least one string in the language of the grammar; useless ones are those do not take part in any derivation as a result of one of two things: (a) they are unreachable from S (i.e., S cannot derive the variable in any sentential form) or they are non-generating (i.e., they cannot get rid of themselves to introduce only terminal symbols).

Cont..

4. *Re-write productions in CNF form* as follows:

Productions which are already of the form $A \rightarrow a$ or $A \rightarrow BC$ are fine;
If a production $A \rightarrow BCD$ has more than two symbols on the right-hand side,
introduce new, intermediate variables and productions of the form:
$A \rightarrow BE$ and $E \rightarrow CD$
If a production $A \rightarrow bC$ has a terminal and a non-terminal,
re-write it in the form:
$A \rightarrow BC$ and $B \rightarrow b$ both of which are in CNF.

Consider a CFG for all palindromes over the alphabet $\{a, b\}$, that is, the union of all even palindromes and all odd palindromes:

$$S \rightarrow aSa \mid bSb \mid A \mid \lambda$$
$$A \rightarrow a \mid b \mid \lambda$$

Let us convert this grammar to CNF:

Step 1: Eliminating lambda productions – S and A both are nullable.

$S \rightarrow aSa \mid aa \mid bSb \mid bb \mid A$	S is nullable
$A \rightarrow a \mid b$	No need for the lambda production

Step 2: Eliminating unit productions – $S \rightarrow A$ is the only unit production.

$$S \rightarrow aSa \mid aa \mid bSb \mid bb \mid a \mid b$$
$$A \rightarrow a \mid b$$

As a result of removing lambda and unit productions, A became an unreachable variable and will be removed in the next step.

Step 3: Eliminating the useless variable A, we get

$$S \rightarrow aSa \mid aa \mid bSb \mid bb \mid a \mid b$$

Step 4: Finally, we need to re-write the first four productions above in CNF by introducing new intermediate variables:

$S \rightarrow XZ$	For aSa
$X \rightarrow a$	
$Z \rightarrow SX$	i.e., Sa
$S \rightarrow XX$	i.e., aa
$S \rightarrow YW$	For bSb
$Y \rightarrow b$	
$W \rightarrow SY$	i.e., Sb
$S \rightarrow YY$	i.e., bb
$S \rightarrow a$	
$S \rightarrow b$	

CYK Algorithm

- J.Cocke
- D.Younger
- T.Kasami
  Independently developed an algorithm to answer whether the given string belongs to the language of the grammar and it is also called as membership algorithm

The CYK Algorithm Basics

The Structure of the rules in a Chomsky Normal

Form grammar

– Uses a “dynamic programming” or “table-filling

algorithm”

Construct a Triangular Table

Each row corresponds to one length of

substrings

– Bottom Row – Strings of length 1

– Second from Bottom Row – Strings of length 2

.

.

– Top Row – string ‘w’

Construct a Triangular Table

$X_{i,i}$ is the set of variables A such that

$A \rightarrow w_i$ is a production of G

• Compare at most n pairs of previously

computed sets:

$(X_{i,i} , X_{i+1,j})$, $(X_{i,i+1} , X_{i+2,j})$ ... $(X_{i,j-1} , X_{j,j})$

Construct a Triangular Table

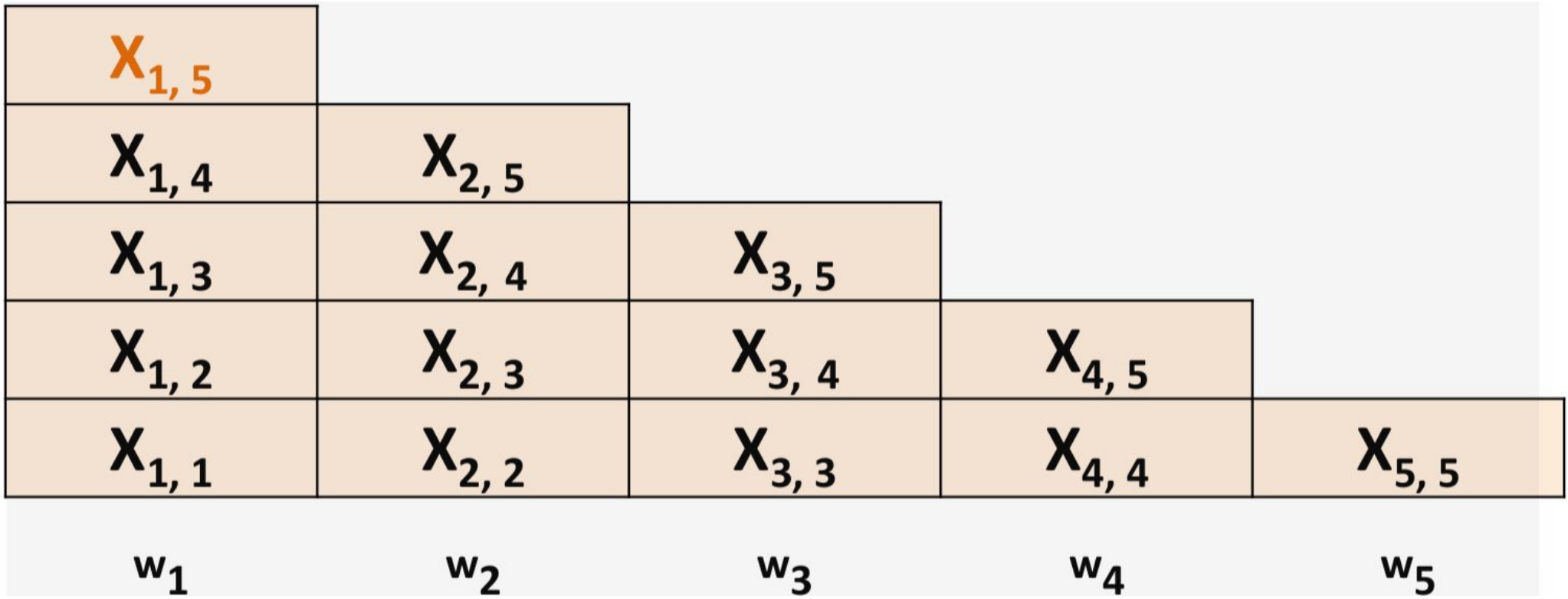
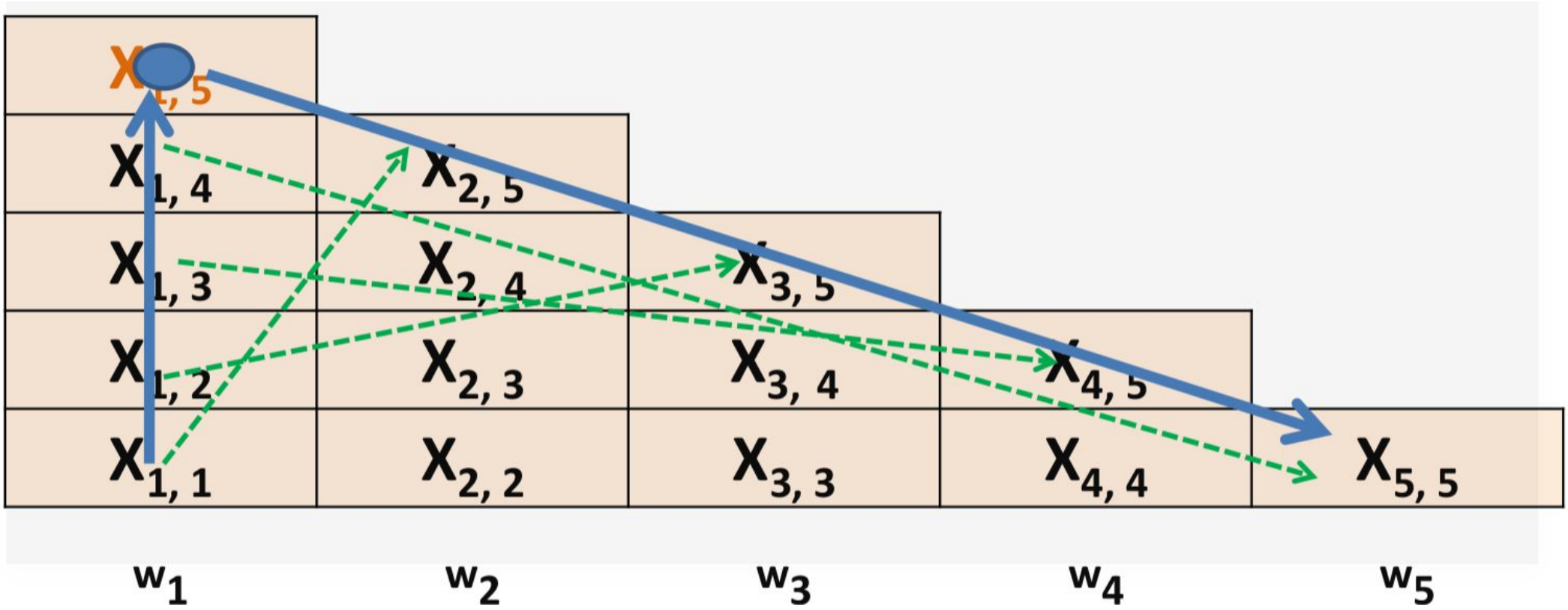


Table for string ‘w’ that has length 5

Construct a Triangular Table



Looking for pairs to compare

Example CYK Algorithm

Show the CYK Algorithm with the following example

CNF grammar G

S→AB | BC

A→BA | a

B→CC | b

C→AB | a

-w is baaba

Question is baaba in L(G)?

S→AB | BC

A→BA | a

B→CC | b

C→AB | a

Filling the bottom row

{B}	{A,C}	{A,C}	{B}	{A,C}

b a a b a

$X_{1,2}=(X_{i,i},X_{i+1,j}) = (X_{1,1},X_{2,2})$

$= \{B\}\{A,C\}=\{BA,BC\}$

Steps:

Looks for production rules to generate BA or BC

There are two: S and A

S→AB | BC

A→BA | a

B→CC | b

C→AB | a

$X_{1,2}=\{S,A\}$

$X_{2,3}=(X_{i,i},X_{i+1,j}) = (X_{2,2},X_{3,3})$

$\{A,C\}\{A,C\} = \{AA,CC,AC,CA\} = Y$

Look for production rules to generate Y

There is one : B

$X_{2,3}=\{B\}$

$X_{1,3} = (X_{i,i}, X_{i+1,j})(X_{i,i+1}, X_{i+2,j})$

$= (X_{1,1}, X_{2,3})(X_{1,2}, X_{3,3})$

$= \{B\}\{B\} \cup \{S,A\}\{A,C\} = \{BB, SA, SC, AA, AC\}$

$= \varnothing$

$X_{2,4} = (X_{2,2}, X_{3,4})(X_{2,3}, X_{4,4})$

$= \{A,C\}\{S,C\} \cup \{B\}\{B\} = \{AS, AC, CS, CC, BB\}$

$= \{B\}$

{S,A,C}				
∅	{S,A,C}			
∅	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}

Is baaba in L(G)?

Yes

We can see the S in the set $X_{1,n}$ where ‘n’ = 5

We can see the table the cell $X_{1,5}$ = (S, A, C) then if S Є $X_{1,5}$ then baaba Є L(G)