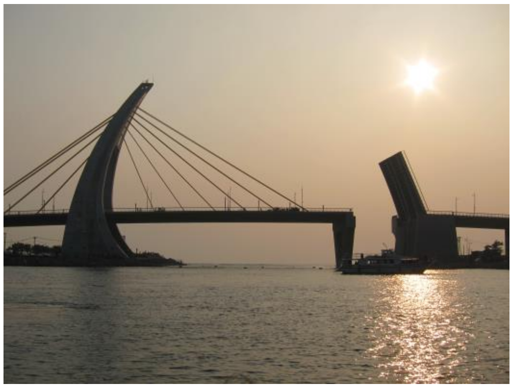
Chapter 7

Properties of Context-free Languages

(2015/12/02)



Peng Bay Bridge, Pingtung, Taiwan

Outline

7.0 Introduction

- **Main concepts to be taught in this chapter ---**
  - CFG's may be simplified to fit certain special forms, like *Chomsky normal form* and *Greiback normal form*.
  - Some, but not all, properties of RL's are also possessed by the CFL's.
  - Unlike the RL, many computational problems about the CFL *cannot* be answered.
  - That is, there are many undecidable problems about CFL's.

7.1 Normal Forms for CFG's

- **Concept ---**

In this section, we want to prove that

every CFG can be transformed into an equivalent grammar in Chomsky normal form,

after simplifying the CFG in the following ways:

- eliminating *useless symbols* (which do not appear in any derivation from the start symbol);
- eliminating *ε-productions* (of the form $A \to \varepsilon$);
- eliminating *unit productions* (of the form $A \to B$);

7.1.1 Eliminating Useless Symbols

- **Some definitions ---**
  - We say symbol X is *useful* for a grammar $G = (V, T, P, S)$ if there is some derivation of the form

$$S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$$

with $w \in T^*$.

  - A symbol is said to be *useless* if not useful.
    - Omitting useless symbols obviously will not change the language generated by the grammar.
  - There are two types of *usefulness* ---
    - X is *generating* if $X \stackrel{*}{\Rightarrow} w$;
    - X is *reachable* if $S \stackrel{*}{\Rightarrow} \alpha X \beta$.

- **Example 7.1 ---**

Eliminate useless symbols in a grammar with the following productions:

$$S \to AB \mid a$$
$$A \to b.$$

  - B is *not generating*, and is so eliminated at first, resulting in $S \to a$, $A \to b$, in which A is *not reachable* and so eliminated too, with $S \to a$ as the only production left.
  - If we eliminate unreachable symbols at first and then non-generating ones, we get the final result $S \to a$, $A \to b$, *which is not what we want!*
  - So, the order of eliminations is *essential*: *eliminate non-generating symbols at first.*

■ **Theorem 7.2 ---**

Let $G = (V, T, P, S)$ be a CFG, and assume that $L(G) \neq$, i.e., assume that G generates at least one string. Let $G_1 = (V_1, T_1, P_1, S)$ be the grammar obtained by the following steps *in order*:

- ♦ eliminate non-generating symbols and all related productions, resulting in grammar G_2;
- ♦ eliminate all symbols not reachable in G_2.

Then, G_1 has no useless symbol and $L(G_1) = L(G)$.

♦ For proof, see the textbook.

7.1.2 <u>Computing Generating and Reachable Symbols</u>

■ **How to compute generating symbols?**

- ♦ *Basis*: every terminal symbol is generating.
- ♦ *Induction*: if every symbol in a in $A \rightarrow \alpha$ is generating, then A is generating.

■ **How to compute reachable symbols?**

- ♦ *Basis*: the start symbol S is reachable.
- ♦ *Induction*: if nonterminal A is reachable, then all the symbols in $A \rightarrow \alpha$ are reachable.

(Both algorithms above are proved correct by Theorems 7.4 and 7.6)

7.1.3 <u>Eliminating ε-Productions</u>

■ **A definition ---** a nonterminal A is said to be *nullable* if $A \Rightarrow^* \varepsilon$.

■ **A Theorem ---** We want to prove that

if a language L has a CFG, then the language $L - \{\varepsilon\}$ can be generated by a CFG without ε-production.

♦ Two steps for the above proof:

- • find "nullable" symbols;
- • transform productions into ones which generate no empty string using the nullable symbols.

■ **Example 7.8 ---**

Given a grammar with productions as follows:

$$S \rightarrow AB$$
$$A \rightarrow aAA \mid \varepsilon$$
$$B \rightarrow bBB \mid \varepsilon$$

then, we can see the following facts:

- ♦ A and B are *nullable* because they derive empty strings;
- ♦ S is also *nullable* because A and B are nullable.

■ How to find nullable symbols systematically?

♦ ***Algorithm 1 ---***

- *Basis*: if $A \to \varepsilon$ is a production, then A is nullable.
- *Induction*: if all C_i in $B \to C_1C_2\ldots C_k$ are nullable, then B is nullable, too.

■ How to transform productions into ones which generate no empty string?

♦ ***Algorithm 2 ---***

- For each production $A \to X_1X_2\ldots X_k$, in which m of the k X_i's are nullable, then generate accordingly 2^m versions of this production where
  (1) the nullable X_i's in all possible combinations are present or absent; and
  (2) if $A \to \varepsilon$ is in the 2^m ones, eliminate it.

■ **Example 7.8** (continued) ---

♦ For $S \to AB$, $A \to aAA \mid \varepsilon$, $B \to bBB \mid \varepsilon$:

- We know S, A, B are *nullable.*
- From $S \to AB$, we get $S \to AB \mid A \mid B \mid \varepsilon$ where $S \to \varepsilon$ should be eliminated.
- From $A \to aAA$, we get $A \to aAA \mid aA \mid aA \mid a$ where the repeated $A \to aA$ should be removed.
- And from $B \to bBB$, similarly we get $B \to bBB \mid bB \mid b$.
- Overall result:

$$S \to AB \mid A \mid B$$
$$A \to aAA \mid aA \mid a$$
$$B \to bBB \mid bB \mid b$$

■ **Theorem 7.7 ---**

Algorithm 1 can be used to find all nullable symbols in a given grammar.

■ **Theorem 7.9 ---**

If G_1 is constructed from a given grammar G by Algorithm 2, then $L(G_1) = L(G) - \{\varepsilon\}$.

(For proofs of the above two theorems, see the textbook.)

7.1.4 Eliminating Unit Productions

■ **Definition** --- a unit production is of the form $A \to B$.

♦ Unit productions sometimes are useful.

♦ For example, use of unit productions $E \to T$ and $T \to F$ removes ambiguity in the 'expression grammar,' resulting in the following unambiguous grammar:

$$E \to T \mid E + T$$
$$T \to F \mid T * F$$
$$F \to I \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

■ But unit productions complicate certain proofs.

■ A two-step technique to eliminate unit productions without changing the generated language:

- ♦ find all "unit pairs"
- ♦ expand productions using unit pairs until all unit productions disappear.

■ **Definition** of *unit pair* ---

- ♦ *Basis*: (A, A) is a unit pair for any nonterminal.
- ♦ *Induction*: If (A, B) is a unit pair and $B \to C$ is a production, then (A, C) is a unit pair.

■ How to find unit pairs?

- ♦ ***Algorithm 3*** ---

  Follow the definition above.

■ **Example 7.10** ---

The unit pairs for the *unambiguous* arithmetic expression grammar mentioned before with the following productions

$$E \to T \mid E + T$$
$$T \to F \mid T * F$$
$$F \to I \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

may be derived as follows:

unit pair (E, E) & $E \to T$ $\Rightarrow$ unit pair (E, T)
unit pair (E, T) & $T \to F$ $\Rightarrow$ unit pair (E, F)
unit pair (E, F) & $F \to I$ $\Rightarrow$ unit pair (E, I)
unit pair (T, T) & $T \to F$ $\Rightarrow$ unit pair (T, F)
unit pair (T, F) & $F \to I$ $\Rightarrow$ unit pair (T, I)
unit pair (F, F) & $F \to I$ $\Rightarrow$ unit pair (F, I)

- ♦ Totally, there are 10 unit pairs --- the above six plus the four (E, E), (T, T), (F, F), (I, I).

■ How to expand productions using unit pairs until all unit productions disappear?

Algorithm 4 ---

Given a grammar $G = (V, T, P, S)$, we construct another $G1 = (V, T, P_1, S)$ as follows:

- ♦ find all the unit pairs of G;
- ♦ for each unit pair (A, B), add to P_1 all the productions $A \to \alpha$, where $B \to \alpha$ is a *non-unit* production in P.

■ **Example 7.12** (continuation of Example 7.10) ---

- ♦ According to Algorithm 4, the unit-production elimination result is shown in Fig. 7.1.
- ♦ The final production set is the *union* of all those on the right column.

Unit pair	Productions
(E, E)	$E \to E + T$ (from $E \to E + T$)
(E, T)	$E \to T * F$ (from $T \to T * F$)
(E, F)	$E \to (E)$
(E, I)	$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

(T, T)	$T \rightarrow T * F$
(T, F)	$T \rightarrow (E)$
(T, I)	$T \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(F, F)	$F \rightarrow (E)$
(F, I)	$F \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(I, I)	$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Fig. 7.1 Unit production elimination result of Example 7.12.

- **Theorem 7.13 ---**

  If grammar G_1 is constructed from Algorithms 3 and 4 above for unit production elimination, then $L(G_1) = L(G)$.

  ♦ For proof, see the textbook.

- **A summary ---**

  Perform eliminations of the following *order* to a grammar G:

  ♦ Elimination of e-productions;
  ♦ Elimination of unit productions;
  ♦ Elimination of useless symbols,

then we can get an equivalent grammar generating the same language *except the empty string* ε. (See the related theorem described next.)

- **Theorem 7.14 ---**

  If G is a CFG generating a language that contains at least one string other than ε, then there is another CFG G_1 such that $L(G_1) = L(G) - \{\varepsilon\}$, and G_1 has no ε-productions, unit productions, or useless symbols.

  ♦ *Proof* --- construct G_1 in an order of three types of eliminations as above. For the rest of the proof, see the textbook.

7.1.5 Chomsky Normal Form

- **Definition ---**

  A grammar G is said to be in *Chomsky Normal form* (*CNF*), if the following two conditions hold:

  ♦ all its productions are in one of the following two simple forms:

  - $A \rightarrow BC$
  - $A \rightarrow a$

  where A, B and C are nonterminals and a is a terminal; and

  ♦ G has no useless symbol.

- **Two-step transformation of a grammar into CNF ---**
  1. Put G into a form said by Theorem 7.14;
  2. transform it into the two production forms of the CNF.

- **Steps to achieve the 2nd step above ---**
  (a) Arrange all production bodies of length 2 or more to consist only of nonterminals;
  (b) break production bodies of length 3 or more into a cascade of productions, each with a body consisting of 2 nonterminals.

- **To perform Step (a) above ---**
  - For every terminal a, create a new nonterminal, say A.
    (Now, every production has a body of a single terminal or at least two nonterminals & no terminal.)
- **To perform Step (b) above:**
  - Break production $A \rightarrow B_1B_2\ldots B_k$, $k \geq 3$, into a group of productions with two nonterminals in each body as follows:

$$A \rightarrow B_1C_1,\ C_1 \rightarrow B_2C_2,\ \ldots,\ C_{k-3} \rightarrow B_{k-2}C_{k-2},\ C_{k-2} \rightarrow B_{k-1}B_k.$$

- **Example 7.15 ---**

  Convert the expression grammar described previously into CNF.

  - For productions in the left column of Fig. 7.1, conduct the following steps:
    (1) create new nonterminals for the terminals to produce the following productions:

$A \rightarrow a$	$B \rightarrow b$	$Z \rightarrow 0$	$O \rightarrow 1$
$P \rightarrow +$	$M \rightarrow *$	$L \rightarrow ($	$R \rightarrow)$

    (2) transformation of $E \rightarrow E + T \mid T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$\Rightarrow E \rightarrow EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$T \rightarrow \ldots$

$F \rightarrow \ldots$

$I \rightarrow \ldots$

$\Rightarrow E \rightarrow EC_1,\ C_1 \rightarrow PT, \ldots$

- **Theorem 7.16 ---**

  If G is a CFG whose language contains at least one string other than ε, then there is a grammar G_1 in CNF such that $L(G_1) = L(G) - \{\varepsilon\}$.

  - *Proof.* See the textbook.

- **Definition --- Greiback Normal Form** (in the box of p. 277) ---

  A production is said to be of the Greiback normal form (GNF) if it is of the form

$$A \rightarrow a\alpha$$

  where a is a terminal and α is a string of zero or more nonterminals.

7.2 Pumping Lemma for CFL's

7.2.1 The Size of Parse Trees

- See the textbook for the detail by yourself (for use in proof of the lemma).

7.2.2 Statement of the Pumping Lemma for CFL's

■ **Theorem 7.18** (pumping lemma for CFL's) ---

Let L be a CFL. There exists an integer constant n such that if $z \in L$ with $|z| \geq n$, then we can write $z = uvwxy$, subject to the following conditions:

1. $|vwx| \leq n$;
2. $vx \neq \varepsilon$ (that is, v, x are not both ε);
3. for all $i \geq 0$, $uv^iwx^iy \in L$.

♦ *Proof*. See the textbook.

7.2.3 Applications of the Pumping Lemma

■ **Example 7.19 ---**

Prove by contradiction the language $L = \{0^n1^n2^n \mid n \geq 1\}$ is not a CFL by the pumping lemma.

Proof.

♦ Suppose L is a CFL. Then there exists an integer n as given by the lemma.

♦ Pick $z = 0^n1^n2^n$ with $|z| = 3n \geq n$, which so can be written as $z = uvwxy$ where
(1) $|vwx| \leq n$;
(2) v, x are not both ε; and
(3) the pumping is true.

♦ By (1), vwx cannot include both 0 and 2 because there are n 1's in between. This can be elaborated by two cases:
(a) vwx has no 2;
(b) vwx has no 0.

♦ The two cases are discussed as follows.

(a) vwx has no 2 ---
- Then v and x consists only 0's and 1's. Now 'pump' up $z' = uv^0wx^0y = uwy$ which, as said by the lemma, is in L.
- However, this is *not* possible because at least one 0 or 1 will be eliminated according to (2) and so z' cannot have n 0's or n 1's, resulting in a form different from that of the strings in L *(because there are n 2's)*.

(b) vwx has no 0 ---
- By symmetry, we can draw the same conclusion as in (a).
- Since no other case exists, we conclude by contradiction that L is not a CFL.

■ **Example 7.21** ---

Prove $L = \{ww \mid w \in \{0, 1\}^*\}$ is not a CFL.

Proof (sketcch only).

♦ Let $z = 0^n1^n0^n1^n$ with n as given by the lemma.

♦ Pump $z' = uv^0wx^0y = uwy$.

♦ Since $|vwx| \leq n$, we know $|z'| = |uwy| \geq 3n$.

♦ If $z' \in L$ is true, then z' is of the form tt with t of length at least $3n/2$.

♦ There are 5 cases to deal with as follows.
(1) $w' = vwx$ is in the first n 0's
(2) w' straddles 1st block of 0's & 1st block of 1's
(3) w' is in 1st block of 1's
(4) w' straddles 1st block of 1's and 0's

(5) w' is in 2nd half of z ---- similar to above 4 cases.

We have to check each case to see contradiction:

- For case (1) ---
  - We have $z = uvwxy = 0^n1^n0^n1^n$.
  - If $w' = vwx$ is in the first n 0's, then let vx consists of k 0's with $k > 0$.
  - Then the pumping result uwy begins with $0^{n-k}1^n$, i.e., it ends in 1.
  - Since $|uwy| = 4n - k$, we know if $uwy = tt$, then $|t| = 2n - k/2$.
  - So, the first t does not end until after the first block of 1's (because uwy begins with $0^{n-k}1^n$), i.e., t ends in 0.
  - So is the second t, which means $tt = uwy$ ends in 0.
  - But the above says that uwy ends in 1. Contradiction!
- The details of (2)~(5) are omitted and can be found in the textbook.

7.3 Closure Properties of CFL's

- **Some differences between CFL's and RL's ---**
  - CFL's are *not* closed under *intersection*, *difference*, or *complementation*
  - But the intersection or difference of a CFL and an RL is still a CFL.
  - We will introduce a new operation --- substitution.

7.3.1 Substitution

- **Definitions ---**
  - A *substitution* s on an alphabet Σ is a function such that for each $a \in \Sigma$, $s(a)$ is a language L_a over any alphabet (not necessarily Σ).
  - For a string $w = a_1a_2\ldots a_n \in \Sigma^*$, $s(w) = s(a_1)s(a_2)\ldots s(a_n) = L_{a1}L_{a2}\ldots L_{an}$, i.e., $s(w)$ is a language which is the concatenation of all L_{ai}'s.
  - Given a language L, $s(L) = \cup_{w \in L}\, s(w)$.

- **Example 7.22 ---**
  - A *substitution* s on an alphabet = {0, 1} is defined as $S(0) = \{a^nb^n \mid n \geq 1\}$, $s(1) = \{aa, bb\}$.
  - Let $w = 01$, then $s(w) = s(0)s(1) = \{a^nb^n \mid n \geq 1\}\{aa, bb\} = \{a^nb^naa \mid n \geq 1\} \cup \{a^nb^{n+2} \mid n \geq 1\}$.
  - Let $L = L(\mathbf{0}^*)$, then

    $s(L) = \cup_{k=0, 1, \ldots}\, s(0^k) = (s(0))^*$ (provable) $= (\{a^nb^n \mid n \geq 1\})^*$

    $= \{ \quad \} \cup \{a^nb^n \mid n \geq 1\} \cup \{a^nb^n \mid n \geq 1\}^2 \cup \ldots$
  - $S(L)$ includes strings like *aabbaaabbb*, *abaabbabab*,…

- **Theorem 7.23 ---**

  If L is a CFL over alphabet , and s is a substitution on such that $s(a)$ is a CFL for each a in , then $s(L)$ is a CFL.

  - *Proof.* See the textbook.

7.3.2 Applications of the Substitution Theorem

- **Theorem 7.24 ---**

  The CFL's are closed under the following operations:

1. Union;
2. Concatenation;
3. Closure (*), and positive closure (+).
4. Homomorphism.

♦ *Proof*. Use the last theorem in the proofs; see the textbook for the detail.

7.3.3 Reversal

- **Theorem 7.25 ---**

  If L is a CFL, so is L^R.

  ♦ *Proof*. See the textbook.

7.3.4 Intersection with an RL

- The CFL is *not* closed under intersection.
- See an example of this fact in the next page.
- **Example 7.26 ---**
  - ♦ $L = \{0^n 1^n 2^n \mid n \geq 1\}$ is not CFL as shown in Example 7.19.
  - ♦ $L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$ and $L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$ are CFL's.
  - ♦ A grammar for L_1 is: $S \rightarrow AB$, $A \rightarrow 0A1 \mid 01$, $B \rightarrow 2B \mid 2$.
  - ♦ A grammar for L_2 is: $S \rightarrow AB$, $A \rightarrow 0A \mid 0$, $B \rightarrow 1B2 \mid 12$.
  - ♦ It is easy to see that $L_1 \cap L_2 = L$ because *both* #0 = #1 in L_1 *and* #1 = # 2 in L_2 means #0 = #1 = #2 as in L.
  - ♦ This shows that intersection of two CFL's L_1 and L_2 yields a non-CFL L.
  - ♦ So CFL's are *not* closed under intersection.

- **Theorem 7.27 ---**

  If L is a CFL and R is an RL, then $L \cap R$ is a CFL.

  ♦ *Proof*. See the textbook.

- For an example, see Example 7.28.

- **Theorem 7.29 ---**

  The following are true about CFL's L, L_1, and L_2, and an RL R:

  1. $L - R$ is a CFL;
  2. $\overline{L}$ is *not* necessarily a CFL;
  3. $L_1 - L_2$ is *not* necessarily a CFL.

  ♦ *Proof*. The proofs are easy to understand. Read by yourself.

7.3.5 Inverse Homomorphism

- **Theorem 7.30 ---**

  Let L be a CFL and h a homomorphism. Then $h^{-1}(L)$ is a CFL.

  ♦ *Proof*. See the textbook.

7.4 Decision Properties of CFL's

- **Facts ---**
  - ♦ Unlike RLs' decision problems which are all solvable, *very little* can be said about CFL's.
  - ♦ Only two problems *can* be *decided* for CFL's:
    - whether the language is empty;
    - whether a given string is in the language.
  - ♦ Computational complexity for conversions between CFG's and PDF's will be investigated.

7.4.1 Complexity of Converting among CFG's and PDA's

- **An assumption** --- n = the length of representation of a PDA or a CFG.

- The following are conversions requiring time of order $O(n)$ (linear time) ---
  - ♦ CFG ⇒ PDA (by the algorithm of Theorem 6.13)
  - ♦ PDA by final state ⇒ PDA by empty stack (by the construction of Theorem 6.11)
  - ♦ PDA by empty stack ⇒ PDA by final state (by the construction of Theorem 6.9)

- Conversion from PDA's to CFG's need *nonlinear* time, as shown by the following theorem.

- **Theorem 7.31 ---**

  There is an $O(n^3)$ algorithm that takes a PDA of length n and produces an equivalent CFG of length at most $O(n^3)$.

  - ♦ *Proof*. See the textbook.

7.4.2 Running Time of Conversion to Chomsky Normal Form

- **Theorem 7.32 ---**

  Given a grammar G of length n, we can find an equivalent CNF grammar for G in time of order $O(n^2)$; and the resulting grammar has length of order $O(n^2)$.

  - ♦ *Proof*. See the textbook.

7.4.3 Testing Emptiness of CFL's

- The problem of testing emptiness of a CFL L is *decidable*.
  - ♦ The algorithm is described in Section 7.1.2 whose main step is:

    decide if the start symbol of the grammar G for L is "generating"; if not, then L is empty.

- A refined algorithm of that in Section 7.1.2 takes time of $O(n)$ (see the textbook for details).

7.4.4 Testing Membership in a CFL

- A way for solving the membership problem for a CFL L is to use the CNF of the CFG G for L in the following way:
  - ♦ The parse tree of an input string w of length n using the CNF grammar G has $2n - 1$

nodes.

- ♦ We can generate all possible parse trees and check if a yield of them is w.
- ♦ The number of such trees is *exponential* in n.

■ A refined way is to use the CYK algorithm which takes time $O(n^3)$.

- ♦ That is, we use the CYK algorithm to check if a given string $w \in L$ in $O(n^3)$ time, assuming the size of the grammar is *constant*. (See the next page for details)
- ♦ See Theorem 7.33 which describes the above facts.
- ♦ **CYK (Cocke, Younger, Kasami) Algorithm ---**
  - This is a *table-filling* algorithm ("tabulation") based on the principle of *dynamic programming*
  - *Input:* grammar G in CNF & string $w = a_1a_2\ldots a_n$.
  - The table entry X_{ij} is the set of nonterminals A such that $A \overset{*}{\Rightarrow} a_ia_{i+1}\ldots.a_j$.
  - If start symbol S is in X_{1n}, then $S \overset{*}{\Rightarrow} a_1a_2\ldots.a_n$ which means that w is generated by the start symbol S and so has answered the problem.
  - To fill the table like the one as follows (for $n = 5$), we start from the bottom row and work upward row-by-row according to the following algorithm:

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
a_1	a_2	a_3	a_4	a_5

- ***CYK (Cocke, Younger, Kasami) Algorithm ---***
  - * *Basis*: for the lowest row, set $Xii = \{A \mid A \to ai$ is a production of $G\}$
  - * *Induction*: for a nonterminal A to be in X_{ij}, try to find nonterminals B and C, and integer k such that
    1. $i \le k < j$.
    2. B is in X_{ik}.
    3. C is in $X_{k+1, j}$.
    4. $A \to BC$ is a production of G.
  - * That is, to find A, we have to compute at most n pairs of previously computed sets: $(X_{ii}, X_{i+1, j})$, $(X_{i, i+1}, X_{i+2, j})$, …, $(X_{i, j-1}, X_{jj})$.
- For example, to compute X_{ij} = **X_{25}**, we have to check the pairs of **(X_{22}, X_{35})**, **(X_{23}, X_{45})**, **(X_{24}, X_{55})** (see the following table for a reference).

X_{15}				
X_{14}	**X_{25}**			
X_{13}	**X_{24}**	**X_{35}**		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	**X_{22}**	X_{33}	X_{44}	**X_{55}**
a_1	a_2	a_3	a_4	a_5

■ **Example 7.34 ---**

Given a grammar G with productions:

$$S \to AB \mid BC \quad A \to BA \mid a$$
$$B \to CC \mid b \quad C \to AB \mid a$$

We want to test if w = *baaba* is generated by G.

♦ A CYK table for the input string is shown in the following.

{*S*, *A*, *C*}				
-	**{*S*, *A*, *C*}**			
-	**{*B*}**	**{*B*}**		
{*S*, *A*}	**{*B*}**	**{*S*, *C*}**	**{*S*, *A*}**	
{*B*}	**{*A*, *C*}**	**{*A*, *C*}**	**{*B*}**	**{*A*, *C*}**
b	***a***	***a***	***b***	***a***

♦ Since S is in X_{15}, so we decide that w is generated by G.

7.4.5 Preview of Undecidable CFL Problems

■ **The following are undecidable CFL problems ---**

♦ Is a given CFG G ambiguous?
♦ Is a given CFL inherently ambiguous?
♦ Is the intersection of two CFL's empty?
♦ Are two CFL's the same?
♦ Is a given CFL equal to S*, where S is the alphabet of this language?

■ These problems will be proved to be *undecidable* in the next chapters.