Properties of Context-Free Languages

Unit-IV

Outline

Introduction

Normal Forms for CFG’s

The Pumping Lemma for CFL’s

Closure Properties of CFL’s

Decision Properties of CFL’s

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 questions about the CFL cannot be answered. That is, there are many undecidable problems about CFL's.

Normal Forms for CFG's

- Concept:

In this section, we want to prove that

every CFL(without ε)can be generated by a CFG in which all productions are of the form A→BC or A→a where A, B and C are Variables and a is a terminal. This form is called Chomsky Normal Form.

To get there we need to need to make the following simplifications:

- 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 $A \to B$)

Normal Forms for CFG's

- Eliminating Useless Symbols
  - We say symbol X is *useful* for a grammar $G = (V, T, P, S)$ if there is some derivation $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ 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.
  - Two types of *usefulness*:
    - X is *generating* if $X \Rightarrow^* w$
    - X is *reachable* if $S \Rightarrow^* \alpha X \beta$

Normal Forms for CFG's

- Eliminating Useless Symbols
  - **Example**

    Given the grammar

    $$S \rightarrow AB \mid a$$
    $$A \rightarrow b$$

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

Normal Forms of CFG's

- Eliminating Useless Symbols
  - **Theorem**

    Let $G = (V, T, P, S)$ be a CFG, and assume that $L(G) \neq \phi$, 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 $L(G_1) = L(G)$.

Normal Forms of CFG's

- Computing Generating & Reachable Symbols
  - How to compute generating symbols?
    - Basis: every terminal symbol is generating.
    - Induction: if every symbol in α in $A \to \alpha$ 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 $A \to \alpha$ are reachable.

Normal Forms of CFG's

- Eliminating ε-Productions
  - We want to prove that if a language *L* has a CFG, then the language $L - \{\varepsilon\}$ has 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*
  - A nonterminal *A* is said to be *nullable* if $A \Rightarrow^* \varepsilon$.

Normal Forms of CFG's

- Eliminating ε-Productions
  - **Example**
    - Given a grammar with productions

$$S \rightarrow AB$$

$$A \rightarrow aAA \mid \varepsilon$$

$$B \rightarrow bBB \mid \varepsilon$$

    - *A, B* are *nullable* because they derive empty strings
    - *S* is also *nullable* because *A, B* are nullable.

      (to be continued)

Normal Forms of CFG's

- Eliminating ε-Productions
  - How to find nullable symbols systematically? (Algorithm 1)
    - Basis: If $A \rightarrow \varepsilon$ is a production, then A is nullable.
    - Induction: If all C_i in $B \rightarrow C_1C_2 \ldots C_k$ are nullable, then B is nullable, too.

Normal Forms of CFG’s

- Construction of the grammar without ϵ-productions:
  - For each production $A \rightarrow X_1X_2\ldots X_k$, 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 m=k then do not include the case where all X_i’s are absent.
    (3) if $A \rightarrow \varepsilon$ is a production in P, eliminate it.

Normal Forms of CFG's

- Eliminating ε-Productions
  - **Example** (cont'd)
    - For $S \rightarrow AB$, $A \rightarrow aAA \mid \varepsilon$, $B \rightarrow bBB \mid \varepsilon$,
      - We know *S, A, B* are *nullable.*
      - From $S \rightarrow AB$, we get $S \rightarrow AB \mid A \mid B \mid \varepsilon$ where $S \rightarrow \varepsilon$ should be eliminated.
      - From $A \rightarrow aAA$, we get $A \rightarrow aAA \mid aA \mid aA \mid a$ where the repeated $A \rightarrow aA$ should be removed.
      - And from $B \rightarrow bBB$, similarly we get $B \rightarrow bBB \mid bB \mid b$.
      - Overall result:

        $S \rightarrow AB \mid A \mid B$

        $A \rightarrow aAA \mid aA \mid a$

        $B \rightarrow bBB \mid bB \mid b$

Normal Forms of CFG's

- Eliminating Unit Productions
  - A unit production is of the form $A \rightarrow B$.
  - Unit productions sometimes are useful.
    - For example, use of unit productions $E \rightarrow T$ and $T \rightarrow F$ removes ambiguity in the 'expression grammar,' resulting in the following unambiguous grammar:

$$
\begin{aligned}
&E \rightarrow T \mid E + T \\
&T \rightarrow F \mid T * F \\
&F \rightarrow I \mid (E) \\
&I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

Normal Forms of CFG’s

- Eliminating Unit Productions
  - 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.*

Normal Forms of CFG's

- Eliminating Unit Productions
  - Definition of *unit pair*
    - Basis: (A, A) is a unit pair for any nonterminal.
    - Induction: If (A, B) is a unit pair and $B \rightarrow C$ is a production, then (A, C) is a unit pair.

Normal Forms of CFG's

- Eliminating Unit Productions
  - **Example** --- The unit pairs for grammar

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

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).

Normal Forms of CFG's

- Eliminating Unit Productions
  - How to expand productions using unit pairs until all unit productions disappear? :
    - Given a grammar $G = (V, T, P, S)$, we construct another $G_1 = (V, T, P_1, S)$ as follows:
      - Find all the unit pairs of G;
      - For each unit pair (A, B), add to P_1 all the productions $A \rightarrow \alpha$, where $B \rightarrow \alpha$ is a *non-unit* production in P.

Normal Forms of CFG's

- Eliminating Unit Productions
  - **Example** (continuation of Example)

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

- The final production set is the *union* of all those on the right column.

Normal Forms of CFG's

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

  - Elimination of ε-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.

Normal Forms of CFG's

- Chomsky Normal Form
  - A grammar G is said to be in *Chomsky Normal form*, or *CNF*, if 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 further G has no useless symbol.

Normal Forms of CFG's

- Chomsky Normal Form
  - Transformation of a grammar into CNF:
    - (1) Put *G* into a form by eliminating ε-productions, then unit productions and finally useless symbols;
    - (2) Transform it into the two production forms of CNF.
  - Steps to achieve the 2nd goal 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.

Normal Forms of CFG's

- Chomsky Normal Form
  - For goal (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 2 nonterminals & no terminal.)
  - For goal (b) above:
    - Break production $A \rightarrow B_1B_2...B_k$, $k \geq 3$, into a group of productions with 2 nonterminals in each body as follows: $A \rightarrow B_1C_1$, $C_1 \rightarrow B_2C_2$, …,

$$C_{k-3} \rightarrow B_{k-2}C_{k-2},\ C_{k-2} \rightarrow B_{k-1}B_k$$

Normal Forms of CFG's

- **Chomsky Normal Form**
  - **Example** --- Conversion of the expression grammar into CNF.

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

(1) create new nonterminals for the terminals to produce the following productions:

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

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

$$\Rightarrow E \rightarrow EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$$
$$T \rightarrow ...$$
$$F \rightarrow ...$$
$$I \rightarrow ...$$
$$\Rightarrow E \rightarrow EC_1,\ C_1 \rightarrow PT,\ ...$$

Pumping Lemma for CFL’s

- The Size of Parse Trees

Theorem:

Suppose we have a parse tree according to a CNF grammar G=(V,T,P,S), and suppose that the yield of the tree is a terminal string w. If the length of the longest path is n, then $|w| <= 2^{n-1}$.

Basis: n=1. It results in a tree with a maximum path length of 1.

It consists of only a root and one leaf labeled by a terminal.

$|w|=1$ since $2^{n-1} =1$

Induction: Let n>1 .Since n>1 the tree starts with the production A->BC. No path in the subtrees rooted at B and C can have greater than n-1. The subtrees have yield of length 2^{n-2} . The yield of the entire tree is $2^{n-2} + 2^{n-2} = 2^{n-1}$.

Pumping Lemma for CFLs

- Statement of the Pumping Lemma
- **Theorem** (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

Our first step is to find a Chomsky-Normal-Form grammar G for L. Technically, we cannot find such a grammar if L is the CFL ø or {ϵ}.

However, if L == ø then the statement of the theorem, which talks about a string z in L surely cannot be violated, since there is no such z in 0.

Also, the CNF grammar G will actually generate L-{ϵ}, but that is again not of importance, since we shall surely pick n > 0, in which case z cannot be ϵ anyway.

Proof

Starting with a CNF grammar G= (V,T,P,S) such that L(G) = L-{ϵ}, let G have m variables. Choose $n = 2^m$.

Next, suppose that z in L is of length at least n. By the previous theorem, any parse tree whose longest path is of length m or less must have a yield of length $2^{m-1} = n/2$ or less.

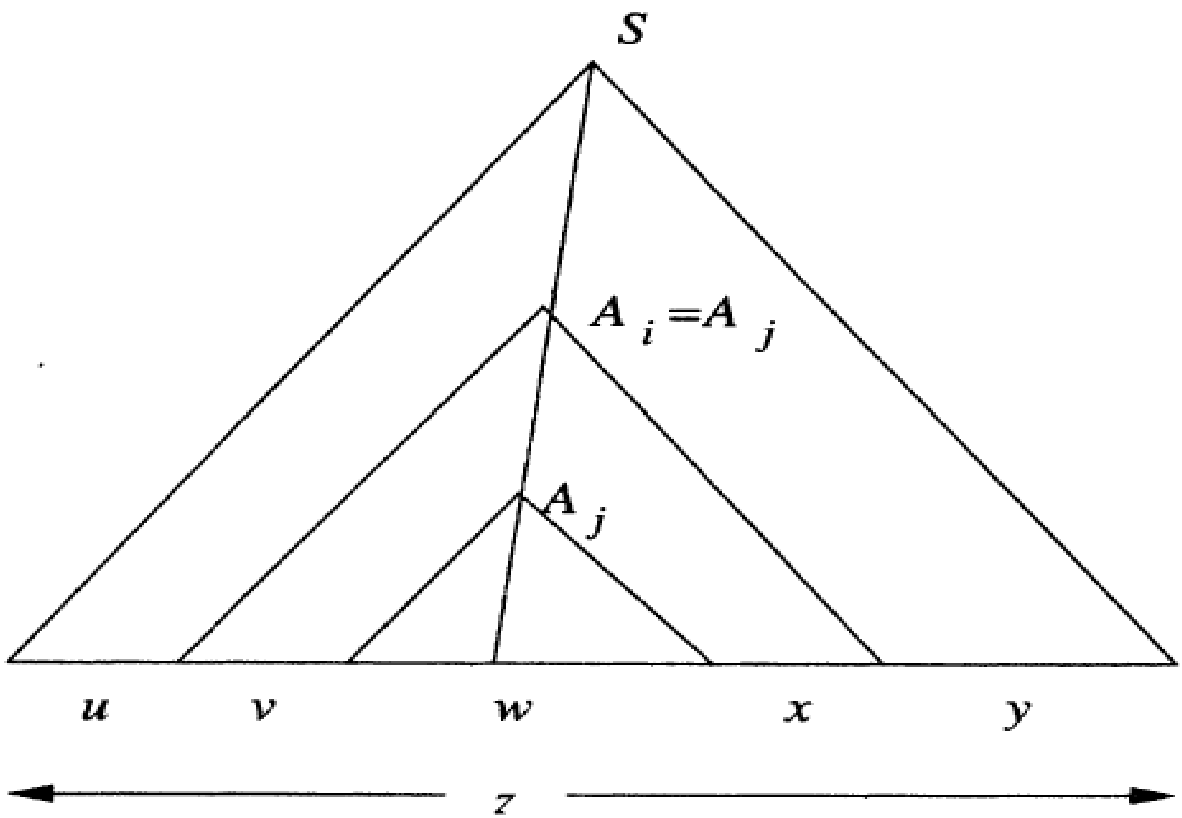
Such a parse tree cannot have yield z, because z is too long. Thus, any parse tree with yield z has a path of length at least m + 1.

Let k+1 be the longest path in the tree where k is at least m. Since k >=m there are at least m+1 occurrences of variables.

As there are only m different variables in V, at least two of the last m + 1 variables on the path must be the same variable. Suppose Ai = Aj, where k-m<= i < J <= k.

The tree is divided into three parts.

Proof



Proof

String w is the yield of the subtree rooted at $A_{j.}$

There are no unit productions so v and x both can not be ϵ.

The strings v and x can be pumped any number of times resulting in uv^iwx^iy

Since k-i<=m the longest path rooted at A_i is no greater than m+1 and its yield is no greater than 2^m=n. Therefore |vwx|<=n.

Applications of Pumping Lemma

1. We pick a language L that we want to show is not a CFL.
2. Our "adversary" gets to pick n, which we do not know, and we therefore must plan for any possible n.
3. We get to pick z, and may use n as a parameter when we do so.
4. Our adversary gets to break z into uvwxy, subject only to the constraints that $|vwx| <= n$ and $vx != \epsilon$
5. If we can pick i and show that $uv^i wx^i y$ is not in L then L is not CFL

Pumping Lemma for CFL's

- Applications of Pumping Lemma
  - **Example**

    Prove by contradiction the language $L = \{0^n 1^n 2^n \mid n \geq 1\}$ 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 $z = 0^n 1^n 2^n$ with $|z| = 3n \geq n$, which can be written as $z = uvwxy$ where

  (1) $|vwx| \leq n$;

  (2) v, x are not both ε; and

  (3) the pumping is true.

Pumping Lemma for CFL's

- Applications of Pumping Lemma
  - **Example**

    Proof (cont'd).

    - 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.

Pumping Lemma for CFL's

- Applications of Pumping Lemma
  - **Example (cont'd)**
    - (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*.
      - It is not possible because the resulting string uwy has n 2's but fewer number of 0's or 1's .

Pumping Lemma for CFL's

- Applications of Pumping Lemma
  - **Example (cont'd)**
    - (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.

Closure Properties of CFL's

- Some differences of CFL's from 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.

Closure Properties of CFL's

- 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 $w = a_1 a_2 \ldots a_n \in \Sigma^*$, $s(w) = s(a_1)s(a_2)\ldots s(a_n) = L_{a1}L_{a2}\ldots L_{an}$, i.e., $s(w)$ is a language which is the concatenation of all L_{ai}'s.
    - Given a language L, $s(L) = \cup_{w \in L} s(w)$.

Closure Properties of CFL's

Substitution

- **Example**
  - A *substitution s* on an alphabet $\Sigma = \{0, 1\}$ is defined as $S(0) = \{a^n b^n \mid n \geq 1\}$, $s(1) = \{aa, bb\}$.
  - Let $w = 01$, then $s(w) = s(0)s(1) = \{a^n b^n \mid n \geq 1\}\{aa, bb\} = \{a^n b^n aa \mid n \geq 1\} \cup \{a^n b^{n+2} \mid n \geq 1\}$.
  - Let $L = L(\mathbf{0}^*)$, then $s(L) = \cup_{k=0,\ 1,\ \ldots} s(0^k)$
    $= (s(0))^*$ (provable) $= (\{a^n b^n \mid n \geq 1\})^*$
    $= \{\varepsilon\} \cup \{a^n b^n \mid n \geq 1\} \cup \{a^n b^n \mid n \geq 1\}^2 \cup \ldots$
  - $S(L)$ includes strings like *aabbaaabbb*, *abaabbabab*,…

Closure Properties of CFL's

- Substitution
  - **Theorem**

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

Closure Properties of CFL's

- Applications of Substitution Theorem
  - **Theorem**

    The CFL's are closed under the following operations:

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

Closure Properties of CFL

- **Union:** Let L1 and L2 be CFL's. Then L1 U L2 is the language s(L), where L is the language {1, 2}, and s is the substitution defined by s(1) =L1 and s(2) = L2.
- **Concatenation:** Again let L1 and L2 be CFL's. Then L1L2 is the language s(L), where L is the language {12}, and s is the same substitution as in union.

Closure Properties of CFL

- **Closure and positive closure:** If L_1 is a CFL, L is the language $\{1\}^*$and s is the substitution $s(1) = L_l$ then $L_1{}^* = s(L)$. Similarly, if L is instead the language $\{1\}^+$, then $L^+ = s(L)$.
- **Homomorphism :** Suppose L is a CFL over alphabet ∑, and h is a homomorphism on ∑. Let s be the substitution that replaces each symbol a in ∑ by the language consisting of the one string that is h(a). That is, s(a) = {h(a)}, for all a in ∑. Then h(L) = s(L).

Closure Properties of CFL's

- Reversal
  - **Theorem**

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

- Intersection with an RL
  - The CFL is *not* closed under intersection.

Reversal

- Let L = L(G) for some CFL G = (V, T,P, S). Construct G^R = (V,T, P^R, S), where P^R is the "reverse" of each production in P. That is, if $A \rightarrow \alpha$ is a production of G, then $A \rightarrow \alpha^R$ is a production of G^R. It is an easy induction on the lengths of derivations in G and G^R to show that $L(G^R)$ = L(R). All the sentential forms of GR are reverses of sentential forms of G.

Closure Properties of CFL's

- The CFL is *not* closed under intersection.
- **Example**
  - $L = \{0^n 1^n 2^n \mid n \geq 1\}$ is not CFL $L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$ & $L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$ 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 L_1 requires same number of 0s and 12 *and* L_2 requires same number of 1s and 2s which means *L must have equal number of 0s, 1s and 2s*.
  - 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.

Closure Properties of CFL's

- Intersection with an RL
  - **Theorem**

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

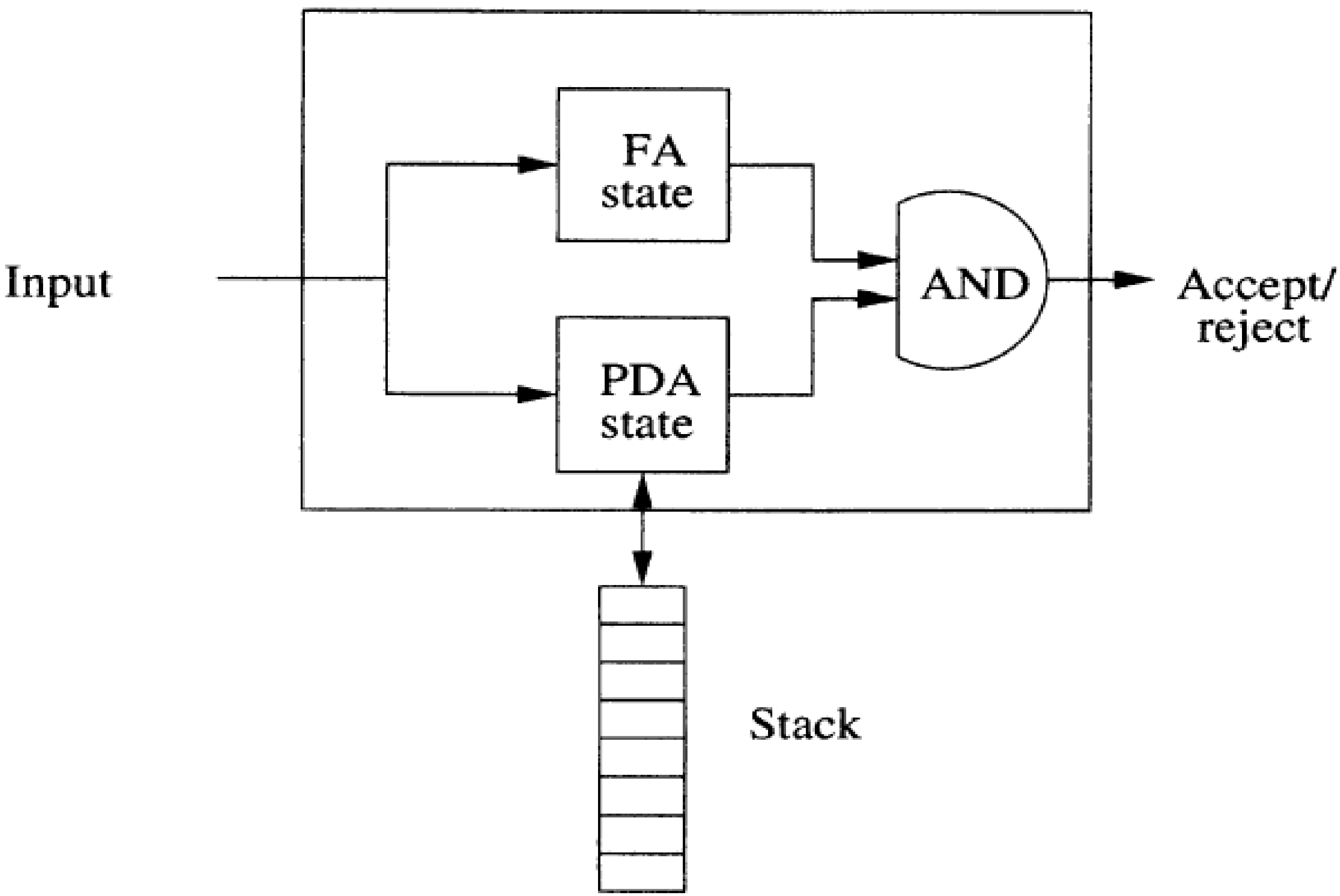
Closure Properties of CFL's

- Intersection with an RL
  - **Theorem**

    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.

Intersection with a RL



Intersection with a RL

PROOF: This proof requires the pushdown-automaton representation of CFL's, as well as the finite-automaton representation of regular languages, and generalizes the proof of Theorem 4.8, where we ran two finite automata "in parallel" to get the intersection of their languages. Here, we run a finite automaton "in parallel" with a PDA, and the result is another PDA, as suggested in Fig. 7.9.

Formally, let

$$P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$$

be a PDA that accepts L by final state, and let

$$A = (Q_A, \Sigma, \delta_A, q_A, F_A)$$

be a DFA for R. Construct PDA

$$P' = (Q_P \times Q_A, \Sigma, \Gamma, \delta, (q_P, q_A), Z_0, F_P \times F_A)$$

where $\delta\big((q,p), a, X\big)$ is defined to be the set of all pairs $\big((r,s), \gamma\big)$ such that:

Intersection with a RL

1. $s = \hat{\delta}_A(p, a)$, and

2. Pair (r, γ) is in $\delta_P(q, a, X)$.

That is, for each move of PDA P, we can make the same move in PDA P', and in addition, we carry along the state of the DFA A in a second component of the state of P'. Note that a may be a symbol of Σ, or $a = \epsilon$. In the former case, $\hat{\delta}(p, a) = \delta_A(p, a)$, while if $a = \epsilon$, then $\hat{\delta}(p, a) = p$; i.e., A does not change state while P makes moves on ϵ input.

It is an easy induction on the numbers of moves made by the PDA's that $(q_P, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \gamma)$ if and only if $\big((q_P, q_A), w, Z_0\big) \overset{*}{\underset{P'}{\vdash}} \big((q, p), \epsilon, \gamma\big)$, where $p = \hat{\delta}(q_A, w)$.

Closure Properties of CFL's

- Inverse Homomorphism
  - **Theorem**

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

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.

Decision Properties of CFL’s

- Complexity of Converting among CFG’s and PDA’s
  - Assume:
    - n = length of representation of a PDA or a CFG
  - The following are conversions of $O(n)$ time (linear time):
    - CFG $\Rightarrow$ PDA (by algorithm of Theorem)
    - PDA by final state $\Rightarrow$ PDA by empty stack (by construction of Theorem)
    - PDA by empty stack $\Rightarrow$ PDA by final state (by construction of Theorem)

Decision Properties of CFL's

- Complexity of Converting among CFG's and PDA's
  - Conversion from PDA's to CFG's is not linear.
  - There is an $O(n^3)$ algorithm that takes a PDA P of length n and produces an equivalent CFG of length at most $O(n^3)$.This CFG generates the same language as P accepts by empty stack. Optionally we can cause G to generate the language that P accepts by final state.

Decision Properties of CFL's

- Running Time of Conversion to Chomsky Normal Form

1) Detecting reachable and generating symbols of a grammar ---- O(n)
2) Construction of unit pairs and elimination of unit pairs------$O(n^2)$
3) Replacement of terminals by variables in production bodies -------------O(n)
4) The breaking of production bodies of length 3 or more into bodies of length 2 ------------------ O(n)

Decision Properties of CFL's

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

Decision Properties of CFL’s

- Testing Emptiness of CFL’s
  - The problem of testing emptiness of a CFL L is *decidable*.
    - decide if the start symbol of the grammar G for L is “generating”; if not, then L is empty.

Decision Properties of CFL’s

- 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:
    - The parse tree of an input string w of length n using the CNF grammar G has $2n - 1$ nodes labelled by variables in that tree. 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.

Decision Properties of CFL's

- Testing Membership in a CFL
  - 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*.

Decision Properties of CFL's

- Testing Membership in a CFL
  - CYK (Cocke, Younger, Kasami) Algorithm ---
    - 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 non-terminals A such that $A \Rightarrow^* a_ia_{i+1} \ldots . a_j$.
    - If start symbol S is in X_{1n}, then $S \Rightarrow^* a_1a_2 \ldots . a_n$ which means that w is generated by the start symbol S and so has answered the problem.

Decision Properties of CFL's

- Testing Membership in a CFL
  - CYK (Cocke, Younger, Kasami) Algorithm ---
    - To fill the table like the one as follows (for *n*=5), start from the bottom row and work upward row-by-row (for details, see the next page).

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

Decision Properties of CFL's

- Testing Membership in a CFL
  - CYK (Cocke, Younger, Kasami) Algorithm ---
    - *Basis*: for the lowest row,

      set $X_{ii} = \{A \mid A \rightarrow a_i$ is a production of $G\}$
    - *Induction*: for a nonterminal A to be in X_{ij}, try to find non-terminals B and C, and integer k such that

      1. $i \leq k < j$.

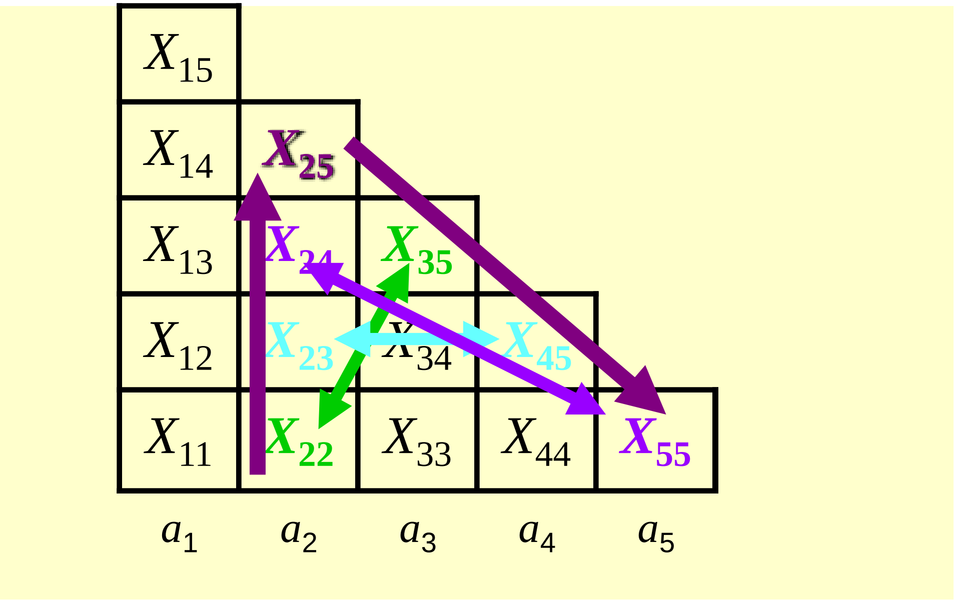
      2. B is in X_{ik}.

      3. C is in $X_{k+1,\ j}$.

      4. $A \rightarrow BC$ is a production of G.
    - That is, to find A, we have to compute at most n pairs of previously computed sets: $(X_{ii}, X_{i+1,j})$, $(X_{i,i+1}, X_{i+2,j})$, …, $(X_{i,j-1}, X_{jj})$.

Decision Properties of CFL's

- Testing Membership in a CFL
  - CYK (Cocke, Younger, Kasami) Algorithm ---
    - For example, to compute X_{ij} = $\boldsymbol{X_{25}}$, we have to check the pairs of **(X_{22}, X_{35}), (X_{23}, X_{45}), (X_{24}, X_{55}).**



- See Fig. for the pattern of this pair computation.

Decision Properties of CFL's

- Testing Membership in a CFL
  - **Example**
    - Given a grammar G with productions:

      $S \rightarrow AB \mid BC$ $\quad A \rightarrow BA \mid a$

      $B \rightarrow CC \mid b$ $\quad C \rightarrow AB \mid a$

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

$\{S, A, C\}$				
-	$\{S, A, C\}$			
-	$\{B\}$	$\{B\}$		
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
b	a	a	b	a

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

Decision Properties of CFL's

- 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 Σ^*, where Σ is the alphabet of this language?