MODULE-4

NORMAL FORMS FOR CONTEXT-FREE GRAMMARS

The goal of this section is to show that every CFL, without ϵ is 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 make a number of preliminary simplications, which are themselves useful in various ways.

1. We must eliminate useless symbols, those variables or terminals that do not appear in any derivation of a terminal string from the start symbol.
2. We must eliminate ϵ-productions, those of the form A→ ϵ for some variable A.
3. We must eliminate unit productions, those of the form A→B for variables A and B.

1. Eliminating Useless Symbols:

We say a symbol X is *useful* for a grammar $G = (V, T, P, S)$ if there is some derivation of the form $S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$, where w is in T^*. Note that X may be in either V or T, and the sentential form $\alpha X \beta$ might be the first or last in the derivation. If X is not useful, we say it is *useless*. Evidently, omitting useless symbols from a grammar will not change the language generated, so we may as well detect and eliminate all useless symbols.

Our approach to eliminating useless symbols begins by identifying the two things a symbol has to be able to do to be useful:

1. We say X is *generating* if $X \stackrel{*}{\Rightarrow} w$ for some terminal string w. Note that every terminal is generating, since w can be that terminal itself, which is derived by zero steps.

2. We say X is *reachable* if there is a derivation $S \stackrel{*}{\Rightarrow} \alpha X \beta$ for some α and β.

Example 7.1 : Consider the grammar:

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

All symbols but B are generating; a and b generate themselves; S generates a, and A generates b. If we eliminate B, we must eliminate the production $S \to AB$, leaving the grammar:

$$S \to a$$
$$A \to b$$

Now, we find that only S and a are reachable from S. Eliminating A and b leaves only the production $S \to a$. That production by itself is a grammar whose language is $\{a\}$, just as is the language of the original grammar.

Note that if we start by checking for reachability first, we find that all symbols of the grammar

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

are reachable.

Theorem 7.2 : Let $G = (V, T, P, S)$ be a CFG, and assume that $L(G) \neq \emptyset$; i.e., G generates at least one string. Let $G_1 = (V_1, T_1, P_1, S)$ be the grammar we obtain by the following steps:

1. First eliminate nongenerating symbols and all productions involving one or more of those symbols. Let $G_2 = (V_2, T_2, P_2, S)$ be this new grammar. Note that S must be generating, since we assume $L(G)$ has at least one string, so S has not been eliminated.

2. Second, eliminate all symbols that are not reachable in the grammar G_2.

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

2. Eliminating ϵ-Productions:

Nullable Variable: A variable A is nullable if , A=>ϵ

Let $G = (V, T, P, S)$ be a CFG. We can find all the nullable symbols of G by the following iterative algorithm. We shall then show that there are no nullable symbols except what the algorithm finds.

BASIS: If $A \to \epsilon$ is a production of G, then A is nullable.

INDUCTION: If there is a production $B \to C_1 C_2 \cdots C_k$, where each C_i is nullable, then B is nullable. Note that each C_i must be a variable to be nullable, so we only have to consider productions with all-variable bodies.

Example 7.8 : Consider the grammar

$$S \to AB$$
$$A \to aAA \mid \epsilon$$
$$B \to bBB \mid \epsilon$$

First, let us find the nullable symbols. A and B are directly nullable because they have productions with ϵ as the body. Then, we find that S is nullable, because the production $S \to AB$ has a body consisting of nullable symbols only. Thus, all three variables are nullable.

Now, let us construct the productions of grammar G_1. First consider $S \to AB$. All symbols of the body are nullable, so there are four ways we could choose present or absent for A and B, independently. However, we are not allowed to choose to make all symbols absent, so there are only three productions:

$$S \to AB \mid A \mid B$$

Next, consider production $A \to aAA$. The second and third positions hold nullable symbols, so again there are four choices of present/absent. In this case, all four choices are allowable, since the nonnullable symbol a will be present in any case. Our four choices yield productions:

$$A \to aAA \mid aA \mid aA \mid a$$

Note that the two middle choices happen to yield the same production, since it doesn't matter which of the A's we eliminate if we decide to eliminate one of them. Thus, the final grammar G_1 will only have three productions for A.

Similarly, the production B yields for G_1:

$$B \to bBB \mid bB \mid b$$

The two ϵ-productions of G yield nothing for G_1. Thus, the following productions:

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

constitute G_1. □

Eliminating Unit Productions:

"A unit production is a production of the form A → B, where both A and B are variables.

However, unit productions can complicate certain proofs and they also introduce extra steps into derivations that technically need not be there.

Ex : A→B

B-> a | ab

Can be rewritten as, A→a | ab, by eliminating the unit production A→B.

Consider the context free grammar given below and remove unit production for the same.

S->0A|1B|C

A->0S|00

B->1|A

C->01

Step 1:

S->C is unit production but while removing S->C we have to consider what C gives so we can add a rule to S.

S->0A|1B|01

Step 2:

B->A is also unit production

B->1|0S|00

Finally, we can write CFG without unit production as follows −

S->0A|1B|01

A->0S|00

B->1|0S|00

C->01

Therefore, the simplification involves following steps in the same order as mentioned below:

1. Eliminate ϵ-productions
2. Eliminate unit productions
3. Eliminate useless symbols

Chomsky Normal Form:

Every nonempty CFL without ϵ has a grammar G in which all productions are in one of the two simple forms, either:

1. A→BC, where A, B, and C, are each variable, or
2. A →a, where A is a variable and a is a terminal.

Also, G has no useless symbols. **Such a grammar is said to be in Chomsky Normal Form or CNF.**

Steps to convert given CFG to CFN:

1. Eliminate ϵ-productions
2. Eliminate unit productions
3. Eliminate useless symbols
4. Put the resulting grammar to CNF
    - Arrange all the bodies of length 2 or more consist only variables.
    - Break bodies of length 3 or more into a cascade of productions, each with a body consisting of two variables.

Example :

Convert the following grammar to CNF.

S→ABa

A→aab

B→Ac

Ans:

Step 1: Eliminate ϵ-productions:

Given grammar doesn't have any ϵ-productions.

Step 2: Eliminate unit productions

Given grammar doesn't have any unit productions.

Step 3: Eliminate useless symbols

Given grammar doesn't have any useless symbols.

Step 4: Convert to CNF

$S \rightarrow ABT_a$	$S \rightarrow AD_1$ $D_1 \rightarrow BT_a$
$A \rightarrow T_aT_aT_b$	$A \rightarrow T_a D_2$ $D_2 \rightarrow T_aT_b$
$B \rightarrow AT_c$	$B \rightarrow AT_c$
$T_a \rightarrow a$ $T_b \rightarrow b$ $T_c \rightarrow c$	$T_a \rightarrow a$ $T_b \rightarrow b$ $T_c \rightarrow c$

Therefore, final grammar in CNF is G'=(V', T', S, P')

Where V'={S, A, B}

T'={a, b}

S is the start symbol

P'={ $S \rightarrow AD_1$

$D_1 \rightarrow BT_a$

$A \rightarrow T_a D_2$

$D_2 \rightarrow T_a T_b$

$B \rightarrow AT_c$

$T_a \rightarrow a$

$T_b \rightarrow b$

$T_c \rightarrow c$ }

(Note: For More problems refer classwork)

The Pumping Lemma for Context-Free Languages:

The Size of Parse Trees:

Theorem 7.17: Suppose we have a parse tree according to a Chomsky-Normal-Form 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 $|w| \leq 2^{n-1}$.

PROOF: The proof is a simple induction on n.

BASIS: $n = 1$. Recall that the length of a path in a tree is the number of edges, i.e., one less than the number of nodes. Thus, a tree with a maximum path length of 1 consists of only a root and one leaf labeled by a terminal. String w is this terminal, so $|w| = 1$. Since $2^{n-1} = 2^0 = 1$ in this case, we have proved the basis.

INDUCTION: Suppose the longest path has length n, and $n > 1$. The root of the tree uses a production, which must be of the form $A \rightarrow BC$, since $n > 1$; i.e., we could not start the tree using a production with a terminal. No path in the subtrees rooted at B and C can have length greater than $n - 1$, since these paths exclude the edge from the root to its child labeled B or C. Thus, by the inductive hypothesis, these two subtrees each have yields of length at most 2^{n-2}. The yield of the entire tree is the concatenation of these two yields, and therefore has length at most $2^{n-2} + 2^{n-2} = 2^{n-1}$. Thus, the inductive step is proved. □

Statement of the Pumping Lemma for CFL:

Theorem 7.18: (The pumping lemma for context-free languages) Let L be a CFL. Then there exists a constant n such that if z is any string in L such that $|z|$ is at least n, then we can write $z = uvwxy$, subject to the following conditions:

1. $|vwx| \leq n$. That is, the middle portion is not too long.
2. $vx \neq \epsilon$. Since v and x are the pieces to be "pumped," this condition says that at least one of the strings we pump must not be empty.
3. For all $i \geq 0$, uv^iwx^iy is in L. That is, the two strings v and x may be "pumped" any number of times, including 0, and the resulting string will still be a member of L.

Proof:

Now, starting with a CNF grammar $G = (V, T, P, S)$ such that $L(G) = L - \{\epsilon\}$, let G have m variables. Choose $n = 2^m$. Next, suppose that z in L is of length at least n. By Theorem 7.17, 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 $m + 1$.

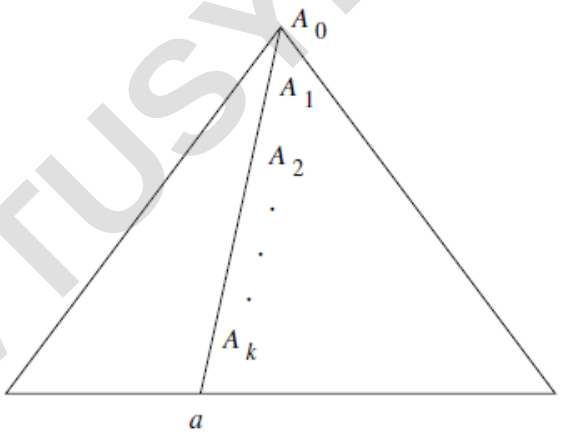


Figure 7.5: Every sufficiently long string in L must have a long path in its parse tree

Figure 7.5 suggests the longest path in the tree for z, where k is at least m and the path is of length $k+1$. Since $k \geq m$, there are at least $m+1$ occurrences of variables $A_0, A_1, \ldots, A_k$ on the path. As there are only m different variables in V, at least two of the last $m + 1$ variables on the path (that is, A_{k-m}

through A_k, inclusive) must be the same variable. Suppose $A_i = A_j$, where $k - m \leq i < j \leq k$.

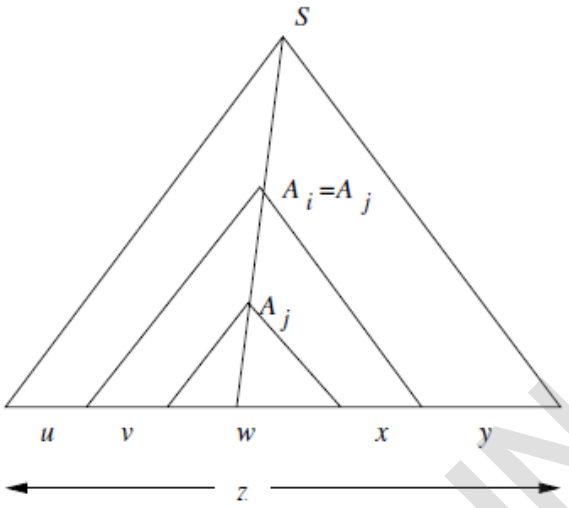


Figure 7.6: Dividing the string w so it can be pumped

Then it is possible to divide the tree as shown in Fig. 7.6. String w is the yield of the subtree rooted at A_j. Strings v and x are the strings to the left and right, respectively, of w in the yield of the larger subtree rooted at A_i. Note that, since there are no unit productions, v and x could not both be ϵ, although one could be. Finally, u and y are those portions of z that are to the left and right, respectively, of the subtree rooted at A_i.

If $A_i = A_j = A$, then we can construct new parse trees from the original tree, as suggested in Fig. 7.7(a). First, we may replace the subtree rooted at A_i, which has yield vwx, by the subtree rooted at A_j, which has yield w. The reason we can do so is that both of these trees have root labeled A. The resulting tree is suggested in Fig. 7.7(b); it has yield uwy and corresponds to the case $i = 0$ in the pattern of strings uv^iwx^iy.

Another option is suggested by Fig. 7.7(c). There, we have replaced the subtree rooted at A_j by the entire subtree rooted at A_i. Again, the justification is that we are substituting one tree with root labeled A for another tree with the same root label. The yield of this tree is uv^2wx^2y. Were we to then replace the subtree of Fig. 7.7(c) with yield w by the larger subtree with yield vwx, we would have a tree with yield uv^3wx^3y, and so on, for any exponent i. Thus, there are parse trees in G for all strings of the form uv^iwx^iy, and we have almost proved the pumping lemma.

The remaining detail is condition (1), which says that $|vwx| \leq n$. However, we picked A_i to be close to the bottom of the tree; that is, $k - i \leq m$. Thus, the longest path in the subtree rooted at A_i is no greater than $m + 1$. By Theorem 7.17, the subtree rooted at A_i has a yield whose length is no greater than $2^m = n$. $\square$

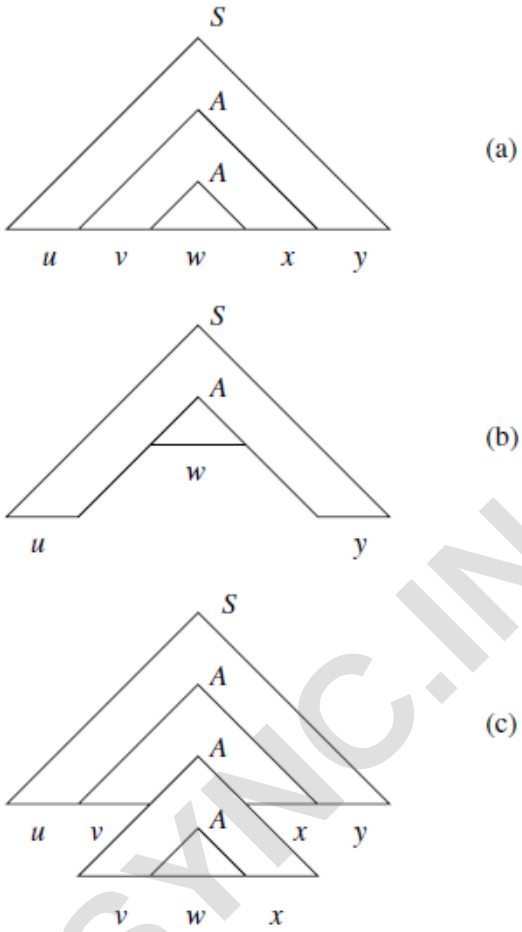


Figure 7.7: Pumping strings v and x zero times and pumping them twice

Application of Pumping Lemma

The pumping lemma for CFL's is used to prove that certain languages are not context free languages.

The general strategy used to prove that a given language is not context free is as follows:

1. Assume that the language L is infinite and is context free.
2. Select the string say z and break it into sub strings u,v,w,x and y such that z=uvwxy where,
   |vwx|≤ & vx≠ϵ
3. Find any i such that $uv^i wx^i y \notin L$. According to pumping lemma,

$uv^i\ wx^i\ y \in L$. So the result is a contradiction to the assumption that the language is context-free. Thus, we can prove that the given language L is not contest-free.

Example:

Show that L={ $a^nb^nc^n$:n>=1} is not context free.

Ans:

Let L is context free. Let z= $a^nb^nc^n \in L$

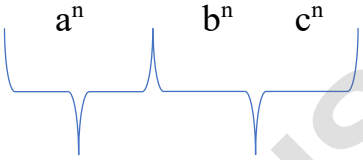
Since |z|=3n>=n, we can split z into uvwxy such that,

|vwx|≤ & vx≠ϵ such that $uv^i\ wx^i\ y \notin L$ for all i=0,1,2,3…

Case 1: String vwx is within a^n

Let v=a^j, x=a^k where j+k>=1 and |vwx|<=n.

Ic, z= a^n b^n c^n



uvwx **y**

For i=2, $z=uv^2wx^2y \in L$

ic, $z = a^{n+j+k}\ b^n\ c^n$

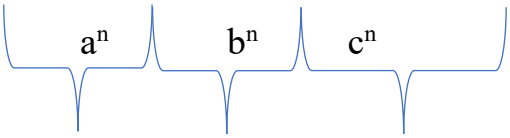
but since $a^{n+j+k}\ b^n\ c^n \neq \epsilon$ L, our assumption is wrong. Thus, given language is not Context-free.

Case 2: String vwx is within b^n

Similarly we can prove with oth

Let v=b^j, x=b^k where j+k>=1 and |vwx|<=n.

Ic, z= a^n b^n c^n



u **vwx** **y**

For i=2, $z=uv^2wx^2y \in L$

ic, $z = a^n b^{n+j+k} c^n$

but since $a^n b^{n+j+k} c^n \neq \in L$, our assumption is wrong. Thus, given language is not Context-free.

(Note: More examples refer class work)

Closure Properties of CFLs:

CFLs are closed under:

1. **Substitution**
2. **Union, concatenation and star closure**
3. **Reverse**
4. **Homomorphism and inverse homomorphism**

CFLs are not closed under

1. **Intersection**
2. **Complement**

7.3.1 Substitutions

Let Σ be an alphabet, and suppose that for every symbol a in Σ, we choose a language L_a. These chosen languages can be over any alphabets, not necessarily Σ and not necessarily the same. This choice of languages defines a function s (a *substitution*) on Σ, and we shall refer to L_a as $s(a)$ for each symbol a.

If $w = a_1 a_2 \cdots a_n$ is a string in Σ^*, then $s(w)$ is the language of all strings $x_1 x_2 \cdots x_n$ such that string x_i is in the language $s(a_i)$, for $i = 1, 2, \ldots, n$. Put another way, $s(w)$ is the concatenation of the languages $s(a_1)s(a_2)\cdots s(a_n)$. We can further extend the definition of s to apply to languages: $s(L)$ is the union of $s(w)$ for all strings w in L.

Example 7.22: Suppose $s(0) = \{a^n b^n \mid n \geq 1\}$ and $s(1) = \{aa, bb\}$. That is, s is a substitution on alphabet $\Sigma = \{0, 1\}$. Language $s(0)$ is the set of strings with one or more a's followed by an equal number of b's, while $s(1)$ is the finite language consisting of the two strings aa and bb.

Theorem 7.23: If L is a context-free language 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.

PROOF: The essential idea is that we may take a CFG for L and replace each terminal a by the start symbol of a CFG for language $s(a)$. The result is a single CFG that generates $s(L)$. However, there are a few details that must be gotten right to make this idea work.

More formally, start with grammars for each of the relevant languages, say $G = (V, \Sigma, P, S)$ for L and $G_a = (V_a, T_a, P_a, S_a)$ for each a in Σ. Since we can choose any names we wish for variables, let us make sure that the sets of variables are disjoint; that is, there is no symbol A that is in two or more of V and any of the V_a's. The purpose of this choice of names is to make sure that when we combine the productions of the various grammars into one set of productions, we cannot get accidental mixing of the productions from two grammars and thus have derivations that do not resemble the derivations in any of the given grammars.

We construct a new grammar $G' = (V', T', P', S)$ for $s(L)$, as follows:

- V' is the union of V and all the V_a's for a in Σ.
- T' is the union of all the T_a's for a in Σ.
- P' consists of:
    1. All productions in any P_a, for a in Σ.
    2. The productions of P, but with each terminal a in their bodies replaced by S_a everywhere a occurs.

CFLs are under Union, Concatenation and Star-closure:

Union

Let L_1 and L_2 be two context free languages. Then we can prove that $L_1 \cup L_2$ is also context free.

Example

Let $L_1 = \{ a^n b^n , n > 0\}$. Corresponding grammar G_1 will have P: S1 → aAb|ab

Let $L_2 = \{ c^m d^m , m \geq 0\}$. Corresponding grammar G_2 will have P: S2 → cBb| ε

Union of L_1 and L_2, $L = L_1 \cup L_2 = \{ a^n b^n \} \cup \{ c^m d^m \}$

The corresponding grammar G will have the additional production S → S1 | S2.

Thus, CFLs are closed under Union.

Concatenation:

If L_1 and L_2 are context free languages, then L_1L_2 is also context free.

Example

Union of the languages L_1 and L_2, $L = L_1L_2 = \{ a^nb^nc^md^m \}$

The corresponding grammar G will have the additional production $S \rightarrow S1\ S2$

Thus, CFLs are closed under concatenation.

Kleene Star or star closure:

If L is a context free language, then L* is also context free.

Example

Let $L = \{ a^nb^n , n \geq 0\}$. Corresponding grammar G will have P: $S \rightarrow aAb| \varepsilon$

Kleene Star $L_1 = \{ a^nb^n \}^*$

The corresponding grammar G_1 will have additional productions $S1 \rightarrow SS_1 | \varepsilon$.

Thus, CFLs are closed under star closure.

CFLs are closed under Reversal:

Theorem 7.25 : If L is a CFL, then so is L^R.

PROOF: 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 $L(G^R) = L^R$. Essentially, all the sentential forms of G^R are reverses of sentential forms of G, and vice-versa. We leave the formal proof as an exercise. □

CFLs are closed under Homomorphism:

Let L be a CFL with grammar G. Let h be a homomorphism on the terminal symbols of G.

We can construct a grammar for h(L) by replacing each terminal symbol a by h(a).

Example:

G has a production S→0S1 |01

h is defined by h(0)=ab, h(1)= ε

h(L(G)) has the grammar with productions S→abS | ab.

Thus, CFLs are closed under homomorphism.

CFLs are closed under Inverse homomorphism:

Theorem 7.30: Let L be a CFL and h a homomorphism. Then $h^{-1}(L)$ is a CFL.

PROOF: Suppose h applies to symbols of alphabet Σ and produces strings in T^*. We also assume that L is a language over alphabet T. As suggested above, we start with a PDA $P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ that accepts L by final state. We construct a new PDA

$$P' = (Q', \Sigma, \Gamma, \delta', (q_0, \epsilon), Z_0, F \times \{\epsilon\}) \tag{7.1}$$

where:

1. Q' is the set of pairs (q, x) such that:
   (a) q is a state in Q, and
   (b) x is a suffix (not necessarily proper) of some string $h(a)$ for some input symbol a in Σ.

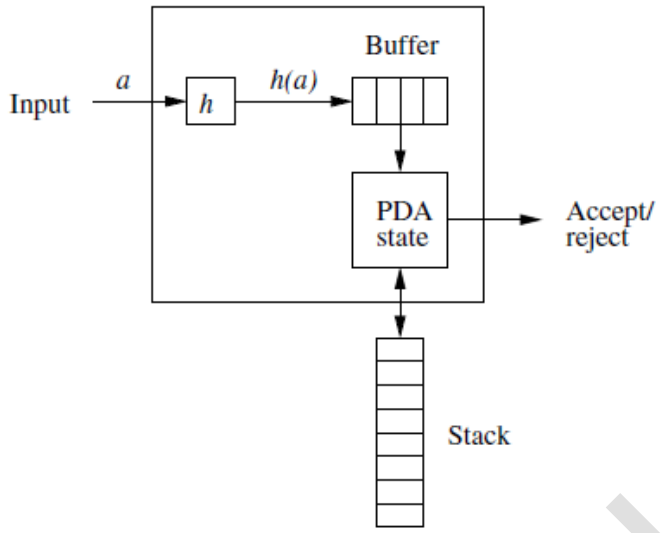


Figure 7.10: Constructing a PDA to accept the inverse homomorphism of what a given PDA accepts

That is, the first component of the state of P' is the state of P, and the second component is the buffer. We assume that the buffer will periodically be loaded with a string $h(a)$, and then allowed to shrink from the front, as we use its symbols to feed the simulated PDA P. Note that since Σ is finite, and $h(a)$ is finite for all a, there are only a finite number of states for P'.

2. δ' is defined by the following rules:

   (a) $\delta'\big((q,\epsilon),a,X\big) = \{\Big(\big(q,h(a)\big),X\Big)\}$ for all symbols a in Σ, all states q in Q, and stack symbols X in Γ. Note that a cannot be ϵ here. When the buffer is empty, P' can consume its next input symbol a and place $h(a)$ in the buffer.

   (b) If $\delta(q,b,X)$ contains (p,γ), where b is in T or $b=\epsilon$, then

   $$\delta'\big((q,bx),\epsilon,X\big)$$

   contains $\big((p,x),\gamma\big)$. That is, P' always has the option of simulating a move of P, using the front of its buffer. If b is a symbol in T, then the buffer must not be empty, but if $b=\epsilon$, then the buffer can be empty.

3. Note that, as defined in (7.1), the start state of P' is (q_0,ϵ); i.e., P' starts in the start state of P with an empty buffer.

4. Likewise, the accepting states of P', as per (7.1), are those states (q,ϵ) such that q is an accepting state of P.

The following statement characterizes the relationship between P' and P:

- $(q_0, h(w), Z_0) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \gamma)$ if and only if $((q_0, \epsilon), w, Z_0) \overset{*}{\underset{P'}{\vdash}} ((p, \epsilon), \epsilon, \gamma)$.

The proofs in both directions are inductions on the number of moves made by the two automata. In the "if" portion, one needs to observe that once the buffer of P' is nonempty, it cannot read another input symbol and must simulate P, until the buffer has become empty (although when the buffer is empty, it may still simulate P). We leave further details as an exercise.

Once we accept this relationship between P' and P, we note that P accepts $h(w)$ if and only if P' accepts w, because of the way the accepting states of P' are defined. Thus, $L(P') = h^{-1}(L(P))$. □

CFLs are not closed under intersection:

Let us prove this with example grammars. Already we know that,

L={ $a^n b^n c^n$:n>=1} is not context free.

However, the following two languages are context-free.

$$L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$$
$$L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$$

A grammar for L_1 is:

$$S \to AB$$
$$A \to 0A1 \mid 01$$
$$B \to 2B \mid 2$$

In this grammar, A generates all strings of the form $0^n 1^n$, and B generates all strings of 2's. A grammar for L_2 is:

$$S \to AB$$
$$A \to 0A \mid 0$$
$$B \to 1B2 \mid 12$$

It works similarly, but with A generating any string of 0's, and B generating matching strings of 1's and 2's.

However, $L = L_1 \cap L_2$. To see why, observe that L_1 requires that there be the same number of 0's and 1's, while L_2 requires the numbers of 1's and 2's to be equal. A string in both languages must have equal numbers of all three symbols and thus be in L.

If the CFL's were closed under intersection, then we could prove the false statement that L is context-free. We conclude by contradiction that the CFL's are not closed under intersection. □

CFLs are not closed under complement:

Theorem: If L is context free, it is not for complement.

Proof. **[Proof 1]** Suppose CFLs were closed under complementation. Then *for any two CFLs* L_1, L_2, we have

- $\overline{L_1}$ and $\overline{L_2}$ are CFL. Then, since CFLs closed under union, $\overline{L_1} \cup \overline{L_2}$ is CFL. Then, again by hypothesis, $\overline{\overline{L_1} \cup \overline{L_2}}$ is CFL.
- i.e., $L_1 \cap L_2$ *is a CFL*

i.e., CFLs are closed under intersection. Contradiction!

Thus, CFLs are not closed under complement.

Theorem: If L is a CFL and R is a regular language, then L ∩ R is a CFL.

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:

1. $s = \hat{\delta}_A(p, a)$, and
2. Pair (r, γ) is in $\delta_P(q, a, X)$.

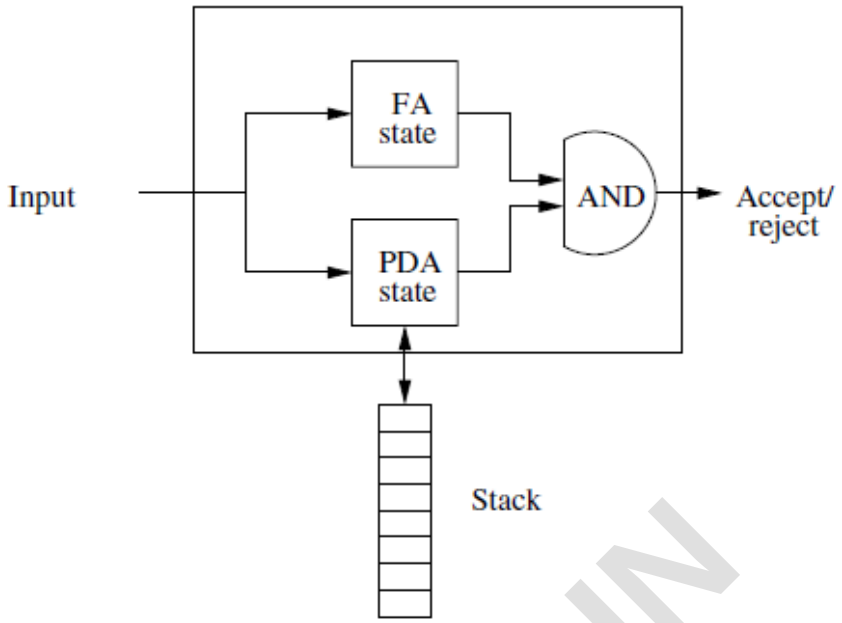


Figure 7.9: A PDA and a FA can run in parallel to create a new PDA

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 $p = \hat{\delta}(q_A, w)$. We leave these inductions as exercises. Since (q, p) is an accepting state of P' if and only if q is an accepting state of P, and p is an accepting state of A, we conclude that P' accepts w if and only if both P and A do; i.e., w is in $L \cap R$. $\square$