Any derivation has an equivalent leftmost and an equivalent rightmost derivation. That is, if w is a terminal string, and A a variable, then $A \stackrel{*}{\Rightarrow} w$ if and only if $A \stackrel{*}{\underset{lm}{\Rightarrow}} w$, and $A \stackrel{*}{\Rightarrow} w$ if and only if $A \stackrel{*}{\underset{rm}{\Rightarrow}} w$. We shall also prove these claims in Section 5.2.

5.1.5 The Language of a Grammar

If $G(V, T, P, S)$ is a CFG, the *language* of G, denoted $L(G)$, is the set of terminal strings that have derivations from the start symbol. That is,

$$L(G) = \{w \text{ in } T^* \mid S \stackrel{*}{\underset{G}{\Rightarrow}} w\}$$

If a language L is the language of some context-free grammar, then L is said to be a *context-free language*, or CFL. For instance, we asserted that the grammar of Fig. 5.1 defined the language of palindromes over alphabet $\{0, 1\}$. Thus, the set of palindromes is a context-free language. We can prove that statement, as follows.

Theorem 5.7: $L(G_{pal})$, where G_{pal} is the grammar of Example 5.1, is the set of palindromes over $\{0, 1\}$.

PROOF: We shall prove that a string w in $\{0,1\}^*$ is in $L(G_{pal})$ if and only if it is a palindrome; i.e., $w = w^R$.

(If) Suppose w is a palindrome. We show by induction on $|w|$ that w is in $L(G_{pal})$.

BASIS: We use lengths 0 and 1 as the basis. If $|w| = 0$ or $|w| = 1$, then w is ϵ, 0, or 1. Since there are productions $P \to \epsilon$, $P \to 0$, and $P \to 1$, we conclude that $P \stackrel{*}{\Rightarrow} w$ in any of these basis cases.

INDUCTION: Suppose $|w| \geq 2$. Since $w = w^R$, w must begin and end with the same symbol That is, $w = 0x0$ or $w = 1x1$. Moreover, x must be a palindrome; that is, $x = x^R$. Note that we need the fact that $|w| \geq 2$ to infer that there are two distinct 0's or 1's, at either end of w.

If $w = 0x0$, then we invoke the inductive hypothesis to claim that $P \stackrel{*}{\Rightarrow} x$. Then there is a derivation of w from P, namely $P \Rightarrow 0P0 \stackrel{*}{\Rightarrow} 0x0 = w$. If $w = 1x1$, the argument is the same, but we use the production $P \to 1P1$ at the first step. In either case, we conclude that w is in $L(G_{pal})$ and complete the proof.

(Only if) Now, we assume that w is in $L(G_{pal})$; that is, $P \stackrel{*}{\Rightarrow} w$. We must conclude that w is a palindrome. The proof is an induction on the number of steps in a derivation of w from P.

has ... prove the harder steps of this equivalence.

5.2.4 From Inferences to Trees

Theorem 5.12: Let $G = (V, T, P, S)$ be a CFG. If the recursive inference procedure tells us that terminal string w is in the language of variable A, then there is a parse tree with root A and yield w.

PROOF: The proof is an induction on the number of steps used to infer that w is in the language of A.

BASIS: One step. Then only the basis of the inference procedure must have been used. Thus, there must be a production $A \to w$. The tree of Fig. 5.8, where there is one leaf for each position of w, meets the conditions to be a parse tree for grammar G, and it evidently has yield w and root A. In the special case that $w = \epsilon$, the tree has a single leaf labeled ϵ and is a legal parse tree with root A and yield w.

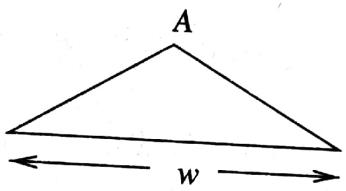

Figure 5.8: Tree constructed in the basis case of Theorem 5.12

INDUCTION: Suppose that the fact w is in the language of A is inferred after $n+1$ inference steps, and that the statement of the theorem holds for all strings x and variables B such that the membership of x in the language of B was inferred using n or fewer inference steps. Consider the last step of the inference that w is in the language of A. This inference uses some production for A, say $A \to X_1 X_2 \cdots X_k$, where each X_i is either a variable or a terminal.

We can break w up as $w_1 w_2 \cdots w_k$, where:

1. If X_i is a terminal, then $w_i = X_i$; i.e., w_i consists of only this one terminal from the production.

2. If X_i is a variable, then w_i is a string that was previously inferred to be in the language of X_i. That is, this inference about w_i took at most n of the $n + 1$ steps of the inference that w is in the language of A. It cannot take all $n + 1$ steps, because the final step, using production $A \to X_1 X_2 \cdots X_k$, is surely not part of the inference about w_i. Consequently, we may apply the inductive hypothesis to w_i and X_i, and conclude that there is a parse tree with yield w_i and root X_i.

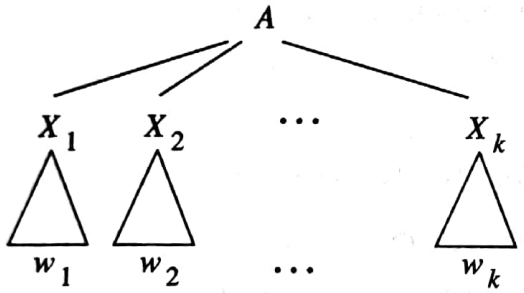

Figure 5.9: Tree used in the inductive part of the proof of Theorem 5.12

We then construct a tree with root A and yield w, as suggested in Fig. 5.9. There is a root labeled A, whose children are $X_1, X_2, \ldots, X_k$. This choice is valid, since $A \to X_1 X_2 \cdots X_k$ is a production of G.

The node for each X_i is made the root of a subtree with yield w_i. In case (1), where X_i is a terminal, this subtree is a trivial tree with a single node labeled X_i. That is, the subtree consists of only this child of the root. Since $w_i = X_i$ in case (1), we meet the condition that the yield of the subtree is w_i.

In case (2), X_i is a variable. Then, we invoke the inductive hypothesis to claim that there is some tree with root X_i and yield w_i. This tree is attached to the node for X_i in Fig. 5.9.

The tree so constructed has root A. Its yield is the yields of the subtrees, concatenated from left to right. That string is $w_1 w_2 \cdots w_k$, which is w. □

5.2.5 From Trees to Derivations

Theorem 5.14: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree with root labeled by variable A and with yield w, where w is in T^*. Then there is a leftmost derivation $A \underset{lm}{\overset{*}{\Rightarrow}} w$ in grammar G.

PROOF: We perform an induction on the height of the tree.

BASIS: The basis is height 1, the least that a parse tree with a yield of terminals can be. In this case, the tree must look like Fig. 5.8, with a root labeled A and children that read w, left-to-right. Since this tree is a parse tree, $A \to w$ must be a production. Thus, $A \underset{lm}{\Rightarrow} w$ is a one-step, leftmost derivation of w from A.

INDUCTION: If the height of the tree is n, where $n > 1$, it must look like Fig 5.9. That is, there is a root labeled A, with children labeled $X_1, X_2, \ldots, X_k$ from the left. The X's may be either terminals or variables.

1. If X_i is a terminal, define w_i to be the string consisting of X_i alone.

---

[1] In fact, it is this property of being able to make a string-for-variable substitution regardless of context that gave rise originally to the term "context-free." There is a more powerful classes of grammars, called "context-sensitive," where replacements are permitted only if certain strings appear to the left and/or right. Context-sensitive grammars do not play a major role in practice today.

2. If X_i is a variable, then it must be the root of some subtree with a yield of terminals, which we shall call w_i. Note that in this case, the subtree is of height less than n, so the inductive hypothesis applies to it. That is, there is a leftmost derivation $X_i \underset{lm}{\overset{*}{\Rightarrow}} w_i$.

Note that $w = w_1 w_2 \cdots w_k$.

We construct a leftmost derivation of w as follows. We begin with the step $A \underset{lm}{\Rightarrow} X_1 X_2 \cdots X_k$. Then, for each $i = 1, 2, \ldots, k$, in order, we show that

$$A \underset{lm}{\overset{*}{\Rightarrow}} w_1 w_2 \cdots w_i X_{i+1} X_{i+2} \cdots X_k$$

This proof is actually another induction, this time on i. For the basis, $i = 0$, we already know that $A \underset{lm}{\Rightarrow} X_1 X_2 \cdots X_k$. For the induction, assume that

$$A \underset{lm}{\overset{*}{\Rightarrow}} w_1 w_2 \cdots w_{i-1} X_i X_{i+1} \cdots X_k$$

a) If X_i is a terminal, do nothing. However, we shall subsequently think of X_i as the terminal string w_i. Thus, we already have

$$A \underset{lm}{\overset{*}{\Rightarrow}} w_1 w_2 \cdots w_i X_{i+1} X_{i+2} \cdots X_k$$

b) If X_i is a variable, continue with a derivation of w_i from X_i, in the context of the derivation being constructed. That is, if this derivation is

$$X_i \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \cdots \underset{lm}{\Rightarrow} w_i$$

we proceed with

$$\begin{aligned}
&w_1 w_2 \cdots w_{i-1} X_i X_{i+1} \cdots X_k \underset{lm}{\Rightarrow} \\
&w_1 w_2 \cdots w_{i-1} \alpha_1 X_{i+1} \cdots X_k \underset{lm}{\Rightarrow} \\
&w_1 w_2 \cdots w_{i-1} \alpha_2 X_{i+1} \cdots X_k \underset{lm}{\Rightarrow} \\
&\cdots \\
&w_1 w_2 \cdots w_i X_{i+1} X_{i+2} \cdots X_k
\end{aligned}$$

The result is a derivation $A \underset{lm}{\overset{*}{\Rightarrow}} w_1 w_2 \cdots w_i X_{i+1} \cdots X_k$.

When $i = k$, the result is a leftmost derivation of w from A. □

6.2.3 From Empty Stack to Final State

We shall show that the classes of languages that are $L(P)$ for some PDA P is the same as the class of languages that are $N(P)$ for some PDA P. This class is also exactly the context-free languages, as we shall see in Section 6.3. Our first construction shows how to take a PDA P_N that accepts a language L by empty stack and construct a PDA P_F that accepts L by final state.

Theorem 6.9: If $L = N(P_N)$ for some PDA $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, then there is a PDA P_F such that $L = L(P_F)$.

PROOF: The idea behind the proof is in Fig. 6.4. We use a new symbol X_0, which must not be a symbol of Γ; X_0 is both the start symbol of P_F and a marker on the bottom of the stack that lets us know when P_N has reached an empty stack. That is, if P_F sees X_0 on top of its stack, then it knows that P_N would empty its stack on the same input.

We also need a new start state, p_0, whose sole function is to push Z_0, the start symbol of P_N, onto the top of the stack and enter state q_0, the start state of P_N. Then, P_F simulates P_N, until the stack of P_N is empty, which P_F detects because

[2]The N in $N(P)$ stands for "null stack," a synonym for "empty stack."

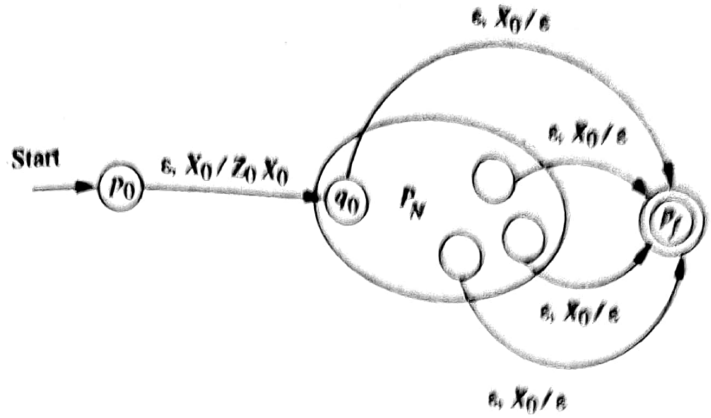

Figure 6.4: P_F simulates P_N and accepts if P_N empties its stack

it sees X_0 on the top of the stack. Finally, we need another new state, p_f, which is the accepting state of P_F; this PDA transfers to state p_f whenever it discovers that P_N would have emptied its stack.

The specification of P_F is as follows:

$$P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

where δ_F is defined by:

1. $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$. In its start state, P_F makes a spontaneous transition to the start state of P_N, pushing its start symbol Z_0 onto the stack.

2. For all states q in Q, inputs a in Σ or $a = \epsilon$, and stack symbols Y in Γ, $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$.

3. In addition to rule (2), $\delta_F(q, \epsilon, X_0)$ contains (p_f, ϵ) for every state q in Q.

We must show that w is in $L(P_F)$ if and only if w is in $N(P_N)$.

(If) We are given that $(q_0, w, Z_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \epsilon)$ for some state q. Theorem 6.5 lets us insert X_0 at the bottom of the stack and conclude $(q_0, w, Z_0 X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, X_0)$. Since by rule (2) above, P_F has all the moves of P_N, we may also conclude that $(q_0, w, Z_0 X_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, X_0)$. If we put this sequence of moves together with the initial and final moves from rules (1) and (3) above, we get:

$$(p_0, w, X_0) \underset{P_F}{\vdash} (q_0, w, Z_0 X_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, X_0) \underset{P_F}{\vdash} (p_f, \epsilon, \epsilon) \tag{6.1}$$

Thus, P_F accepts w by final state.

(Only-if) The converse requires only that we observe the additional transitions of rules (1) and (3) give us very limited ways to accept w by final state. We must use rule (3) at the last step, and we can only use that rule if the stack of P_F contains

only X_0. No X_0's ever appear on the stack except at the bottommost position. Further, rule (1) is only used at the first step, and it *must* be used at the first step.

Thus, any computation of P_F that accepts w must look like sequence (6.1). Moreover, the middle of the computation — all but the first and last steps — must also be a computation of P_N with X_0 below the stack. The reason is that, except for the first and last steps, P_F cannot use any transition that is not also a transition of P_N, and X_0 cannot be exposed or the computation would end at the next step. We conclude that $(q_0, w, Z_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \epsilon)$. That is, w is in $N(P_N)$. □

Theorem 6.11: Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Then there is a PDA P_N such that $L = N(P_N)$.

PROOF: The construction is as suggested in Fig. 6.7. Let

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

where δ_N is defined by:

1. $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$. We start by pushing the start symbol of P_F onto the stack and going to the start state of P_F.

2. For all states q in Q, input symbols a in Σ or $a = \epsilon$, and Y in Γ, $\delta_N(q, a, Y)$ contains every pair that is in $\delta_F(q, a, Y)$. That is, P_N simulates P_F.

3. For all accepting states q in F and stack symbols Y in Γ or $Y = X_0$, $\delta_N(q, \epsilon, Y)$ contains (p, ϵ). By this rule, whenever P_F accepts, P_N can start emptying its stack without consuming any more input.

4. For all stack symbols Y in Γ or $Y = X_0$, $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$. Once in state p, which only occurs when P_F has accepted, P_N pops every symbol on its stack, until the stack is empty. No further input is consumed.

Now, we must prove that w is in $N(P_N)$ if and only if w is in $L(P_F)$. The ideas are similar to the proof for Theorem 6.9. The "if" part is a direct simulation, and the "only-if" part requires that we examine the limited number of things that the constructed PDA P_N can do.

(If) Suppose $(q_0, w, Z_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, \alpha)$ for some accepting state q and stack string α. Using the fact that every transition of P_F is a move of P_N, and invoking Theorem 6.5 to allow us to keep X_0 below the symbols of Γ on the stack, we know that $(q_0, w, Z_0X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \alpha X_0)$. Then P_N can do the following:

$$(p_0, w, X_0) \underset{P_N}{\vdash} (q_0, w, Z_0X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \alpha X_0) \overset{*}{\underset{P_N}{\vdash}} (p, \epsilon, \epsilon)$$

The first move is by rule (1) of the construction of P_N, while the last sequence of moves is by rules (3) and (4). Thus, w is accepted by P_N, by empty stack.

(Only-if) The only way P_N can empty its stack is by entering state p, since X_0 is sitting at the bottom of stack and X_0 is not a symbol on which P_F has any moves. The only way P_N can enter state p is if the simulated P_F enters an accepting state. The first move of P_N is surely the move given in rule (1). Thus, every accepting computation of P_N looks like

$$(p_0, w, X_0) \underset{P_N}{\vdash} (q_0, w, Z_0X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, \alpha X_0) \overset{*}{\underset{P_N}{\vdash}} (p, \epsilon, \epsilon)$$

where q is an accepting state of P_F.

Moreover, between ID's (q_0, w, Z_0X_0) and $(q, \epsilon, \alpha X_0)$, all the moves are moves of P_F. In particular, X_0 was never the top stack symbol prior to reaching ID $(q, \epsilon, \alpha X_0)$.[3] Thus, we conclude that the same computation can occur in P_F, without the X_0 on the stack; that is, $(q_0, w, Z_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, \alpha)$. Now we see that P_F accepts w by final state, so w is in $L(P_F)$. □

1. The net popping of some symbol X from the stack, and

2. A change in state from some p at the beginning to q when X has finally been replaced by ϵ on the stack.

We represent such a variable by the composite symbol $[pXq]$. Remember that this sequence of characters is our way of describing *one* variable; it is not five grammar symbols. The formal construction is given by the next theorem.

Theorem 6.14: Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ be a PDA. Then there is a context-free grammar G such that $L(G) = N(P)$.

PROOF: We shall construct $G = (V, \Sigma, R, S)$, where the set of variables V consists of:

1. The special symbol S, which is the start symbol, and

2. All symbols of the form $[pXq]$, where p and q are states in Q, and X is a stack symbol, in Γ.

The productions of G are as follows:

a) For all states p, G has the production $S \to [q_0 Z_0 p]$. Recall our intuition that a symbol like $[q_0 Z_0 p]$ is intended to generate all those strings w that cause

P to pop Z_0 from its stack while going from state q_0 to state p. That is, $(q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$. If so, then these productions say that start symbol S will generate all strings w that cause P to empty its stack, after starting in its initial ID.

b) Let $\delta(q, a, X)$ contain the pair $(r, Y_1Y_2 \cdots Y_k)$, where:

1. a is either a symbol in Σ or $a = \epsilon$.
2. k can be any number, including 0, in which case the pair is (r, ϵ).

Then for all lists of states $r_1, r_2, \ldots, r_k$, G has the production

$$[qXr_k] \to a[rY_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kr_k]$$

This production says that one way to pop X and go from state q to state r_k is to read a (which may be ϵ), then use some input to pop Y_1 off the stack while going from state r to state r_1, then read some more input that pops Y_2 off the stack and goes from state r_1 to r_2, and so on.

We shall now prove that the informal interpretation of the variables $[qXp]$ is correct:

- $[qXp] \stackrel{*}{\Rightarrow} w$ if and only if $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$.

(If) Suppose $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$. We shall show $[qXp] \stackrel{*}{\Rightarrow} w$ by induction on the number of moves made by the PDA.

BASIS: One step. Then (p, ϵ) must be in $\delta(q, w, X)$, and w is either a single symbol or ϵ. By the construction of G, $[qXp] \to w$ is a production, so $[qXp] \Rightarrow w$.

INDUCTION: Suppose the sequence $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$ takes n steps, and $n > 1$. The first move must look like

$$(q, w, X) \vdash (r_0, x, Y_1Y_2 \cdots Y_k) \vdash^* (p, \epsilon, \epsilon)$$

where $w = ax$ for some a that is either ϵ or a symbol in Σ. It follows that the pair $(r_0, Y_1Y_2 \cdots Y_k)$ must be in $\delta(q, a, X)$. Further, by the construction of G, there is a production $[qXr_k] \to a[r_0Y_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kr_k]$, where:

1. $r_k = p$, and
2. $r_1, r_2, \ldots, r_{k-1}$ are any states in Q.

In particular, we may observe, as was suggested in Fig. 6.10, that each of the symbols $Y_1, Y_2, \ldots, Y_k$ gets popped off the stack in turn, and we may choose p_i to be the state of the PDA when Y_i is popped, for $i = 1, 2, \ldots, k-1$. Let $x = w_1w_2 \cdots w_k$, where w_i is the input consumed while Y_i is popped off the stack. Then we know that $(r_{i-1}, w_i, Y_i) \vdash^* (r_i, \epsilon, \epsilon)$.

As none of these sequences of moves can take as many as n moves, the inductive hypothesis applies to them. We conclude that $[r_{i-1}Y_ir_i] \stackrel{*}{\Rightarrow} w_i$. We may put these derivations together with the first production used to conclude:

$$\begin{aligned}
&[qXr_k] \Rightarrow a[r_0Y_1r_1][r_1Y_1r_2]\cdots[r_{k-1}Y_kr_k] \stackrel{*}{\Rightarrow} \\
&aw_1[r_1Y_2r_2][r_2Y_3r_3]\cdots[r_{k-1}Y_kr_k] \stackrel{*}{\Rightarrow} \\
&aw_1w_2[r_2Y_3r_3]\cdots[r_{k-1}Y_kr_k] \stackrel{*}{\Rightarrow} \\
&\cdots \\
&aw_1w_2\cdots w_k = w
\end{aligned}$$

where $r_k = p$.

(Only-if) The proof is an induction on the number of steps in the derivation.

BASIS: One step. Then $[qXp] \to w$ must be a production. The only way for this production to exist is if there is a transition of P in which X is popped and state q becomes state p. That is, (p, ϵ) must be in $\delta(q, a, X)$, and $a = w$. But then $(q, w, X) \vdash (p, \epsilon, \epsilon)$.

INDUCTION: Suppose $[qXp] \stackrel{*}{\Rightarrow} w$ by n steps, where $n > 1$. Consider the first sentential form explicitly, which must look like

$$[qXr_k] \Rightarrow a[r_0Y_1r_1][r_1Y_2r_2]\cdots[r_{k-1}Y_kr_k] \stackrel{*}{\Rightarrow} w$$

where $r_k = p$. This production must come from the fact that $(r_0, Y_1Y_2\cdots Y_k)$ is in $\delta(q, a, X)$.

We can break w into $w = aw_1w_2\cdots w_k$ such that $[r_{i-1}Y_ir_i] \stackrel{*}{\Rightarrow} w_i$ for all $i = 1, 2, \ldots, r_k$. By the inductive hypothesis, we know that for all i,

$$(r_{i-1}, w_i, Y_i) \stackrel{*}{\vdash} (r_i, \epsilon, \epsilon)$$

If we use Theorem 6.5 to put the correct strings beyond w_i on the input and below Y_i on the stack, we also know that

$$(r_{i-1}, w_iw_{i+1}\cdots w_k, Y_iY_{i+1}\cdots Y_k) \stackrel{*}{\vdash} (r_i, w_{i+1}\cdots w_k, Y_{i+1}\cdots Y_k)$$

If we put all these sequences together, we see that

$$\begin{aligned}
&(q, aw_1w_2\cdots w_k, X) \vdash (r_0, w_1w_2\cdots w_k, Y_1Y_2\cdots Y_k) \stackrel{*}{\vdash} \\
&(r_1, w_2w_3\cdots w_k, Y_2Y_3\cdots Y_k) \stackrel{*}{\vdash} (r_2, w_3\cdots w_k, Y_3\cdots Y_k) \stackrel{*}{\vdash} \cdots \stackrel{*}{\vdash} (r_k, \epsilon, \epsilon)
\end{aligned}$$

Since $r_k = p$, we have shown that $(q, w, X) \stackrel{*}{\vdash} (p, \epsilon, \epsilon)$.

We complete the proof as follows. $S \stackrel{*}{\Rightarrow} w$ if and only if $[q_0Z_0p] \stackrel{*}{\Rightarrow} w$ for some p, because of the way the rules for start symbol S are constructed. We just proved that $[q_0Z_0p] \stackrel{*}{\Rightarrow} w$ if and only if $(q, w, Z_0) \stackrel{*}{\vdash} (p, \epsilon, \epsilon)$, i.e., if and only if P accepts x by empty stack. Thus, $L(G) = N(P)$. □

ϵ-productions, such that there is a grammar G_1 with no ϵ-productions, such that

$$L(G_1) = L(G) - \{\epsilon\}$$

Theorem 7.9: If the grammar G_1 is constructed from G by the above construction for eliminating ϵ-productions, then $L(G_1) = L(G) - \{\epsilon\}$.

PROOF: We must show that if $w \neq \epsilon$, then w is in $L(G_1)$ if and only if w is in $L(G)$. As is often the case, we find it easier to prove a more general statement. In this case, we need to talk about the terminal strings that each variable generates, even though we only care what the start symbol S generates. Thus, we shall prove:

- $A \underset{G_1}{\overset{*}{\Rightarrow}} w$ if and only if $A \underset{G}{\overset{*}{\Rightarrow}} w$ and $w \neq \epsilon$.

In each case, the proof is an induction on the length of the derivation.

(Only-if) Suppose that $A \underset{G_1}{\overset{*}{\Rightarrow}} w$. Then surely $w \neq \epsilon$, because G_1 has no ϵ-productions. We must show by induction on the length of the derivation that $A \underset{G}{\overset{*}{\Rightarrow}} w$.

BASIS: One step. Then there is a production $A \to w$ in G_1. The construction of G_1 tells us that there is some production $A \to \alpha$ of G, such that α is w, with zero or more nullable variables interspersed. Then in G, $A \underset{G}{\Rightarrow} \alpha \underset{G}{\overset{*}{\Rightarrow}} w$, where the steps after the first, if any, derive ϵ from whatever variables there are in α.

INDUCTION: Suppose the derivation takes $n > 1$ steps. Then the derivation looks like $A \underset{G_1}{\Rightarrow} X_1X_2\cdots X_k \underset{G_1}{\overset{*}{\Rightarrow}} w$. The first production used must come from a production $A \to Y_1Y_2\cdots Y_m$, where the Y's are the X's, in order, with zero or more additional, nullable variables interspersed. Also, we can break w into $w_1w_2\cdots w_k$, where $X_i \underset{G_1}{\overset{*}{\Rightarrow}} w_i$ for $i = 1, 2, \ldots, k$. If X_i is a terminal, then $w_i = X_i$, and if X_i is a variable, then the derivation $X_i \underset{G_1}{\overset{*}{\Rightarrow}} w_i$ takes fewer than n steps. By the inductive hypothesis, we can conclude $X_i \underset{G}{\overset{*}{\Rightarrow}} w_i$.

Now, we construct a corresponding derivation in G as follows:

$$A \underset{G}{\Rightarrow} Y_1Y_2\cdots Y_m \underset{G}{\overset{*}{\Rightarrow}} X_1X_2\cdots X_k \underset{G}{\overset{*}{\Rightarrow}} w_1w_2\cdots w_k = w$$

The first step is application of the production $A \to Y_1Y_2\cdots Y_k$ that we know exists in G. The next group of steps represents the derivation of ϵ from each of the Y_j's that is not one of the X_i's. The final group of steps represents the derivations of the w_i's from the X_i's, which we know exist by the inductive hypothesis.

(If) Suppose $A \underset{G}{\overset{*}{\Rightarrow}} w$ and $w \neq \epsilon$. We show by induction on the length n of the derivation, that $A \underset{G_1}{\overset{*}{\Rightarrow}} w$.

has no unit productions, yet generates the same set of expressions as the grammar of Fig. 5.19. □

Theorem 7.13: If grammar G_1 is constructed from grammar G by the algorithm described above for eliminating unit productions, then $L(G_1) = L(G)$.

PROOF: We show that w is in $L(G)$ if and only if w is in $L(G_1)$.

(If) Suppose $S \underset{G_1}{\overset{*}{\Rightarrow}} w$. Since every production of G_1 is equivalent to a sequence of zero or more unit productions of G followed by a nonunit production of G, we know that $\alpha \underset{G_1}{\Rightarrow} \beta$ implies $\alpha \underset{G}{\overset{*}{\Rightarrow}} \beta$. That is, every step of a derivation in G_1 can be replaced by one or more derivation steps in G. If we put these sequences of steps together, we conclude that $S \underset{G}{\overset{*}{\Rightarrow}} w$.

(Only-if) Suppose now that w is in $L(G)$. Then by the equivalences in Section 5.2, we know that w has a leftmost derivation, i.e., $S \underset{lm}{\Rightarrow} w$. Whenever a unit production is used in a leftmost derivation, the variable of the body becomes the leftmost variable, and so is immediately replaced. Thus, the leftmost derivation in grammar G can be broken into a sequence of steps in which zero or more unit productions are followed by a nonunit production. Note that any nonunit production that is not preceded by a unit production is a "step" by itself. Each of these steps can be performed by one production of G_1, because the construction of G_1 created exactly the productions that reflect zero or more unit productions followed by a nonunit production. Thus, $S \underset{G_1}{\overset{*}{\Rightarrow}} w$. □

We can now summarize the various simplifications described so far. We want to convert any CFG G into an equivalent CFG that has no useless symbols, ϵ-productions, or unit productions. Some care must be taken in the order of application of the constructions. A safe order is:

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

You should notice that, just as in Section 7.1.1, where we had to order the two steps properly or the result might have useless symbols, we must order the three steps above as shown, or the result might still have some of the features we thought we were eliminating.

Theorem 7.27: If L is a CFL and R is a regular language, then $L \cap R$ 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

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

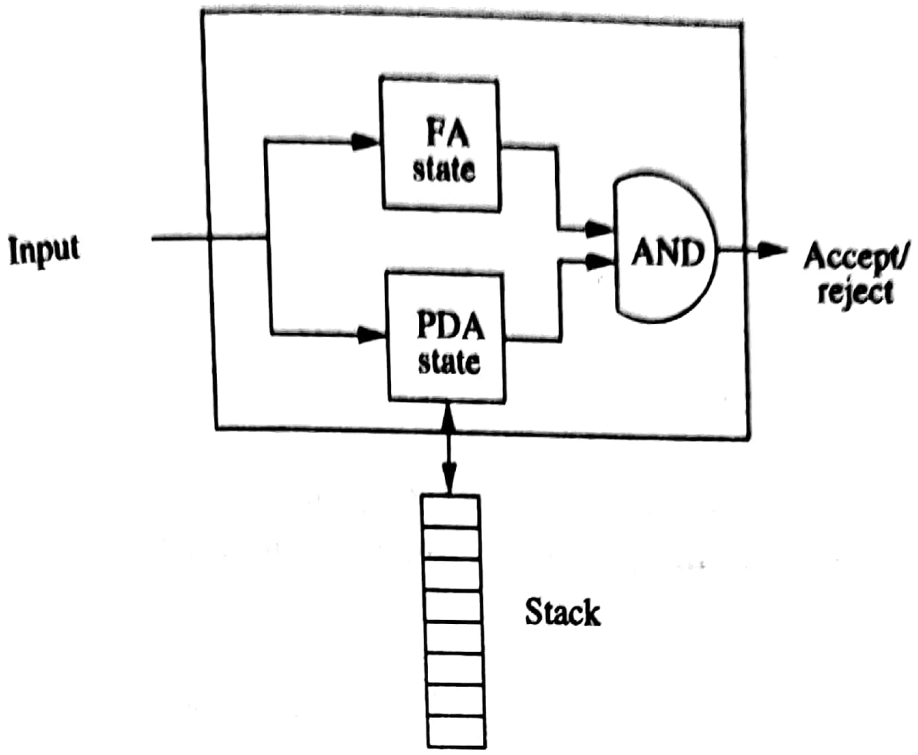

Figure 7.9: A PDA and a FA can run in parallel to create a new PDA

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 $\delta_P(q,a,X)$.

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, $\hat{\delta}(p,a) = \delta_A(p)$, 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 $p = \hat{\delta}(p_A,w)$. 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 $L \cap R$. □

Theorem 7.29: The following are true about a CFL's L, L_1, and L_2, and a regular language R.

1. $L - R$ is a context-free language.
2. $\overline{L}$ is not necessarily a context-free language.
3. $L_1 - L_2$ is not necessarily context-free.

PROOF: For (1), note that $L - R = L \cap \overline{R}$. If R is regular, so is $\overline{R}$ regular by Theorem 4.5. Then $L - R$ is a CFL by Theorem 7.27.

For (2), suppose that $\overline{L}$ is always context-free when L is. Then since

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

and the CFL's are closed under union, it would follow that the CFL's are closed under intersection. However, we know they are not from Example 7.26.

Lastly, let us prove (3). We know Σ^* is a CFL for every alphabet Σ; designing a grammar or PDA for this regular language is easy. Thus, if $L_1 - L_2$ were always a CFL when L_1 and L_2 are, it would follow that $\Sigma^* - L$ was always a CFL when L is. However, $\Sigma^* - L$ is $\overline{L}$ when we pick the proper alphabet Σ. Thus, we would contradict (2) and we have proved by contradiction that $L_1 - L_2$ is not necessarily a CFL. □

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

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

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.

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) $\delta'\big((q,\epsilon),a,X\big) = \{\big((q,h(a)\big),X)\}$ 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 $\big((q_0,\epsilon),w,Z_0\big) \overset{*}{\underset{P'}{\vdash}} \big((p,\epsilon),\epsilon,\gamma\big)$.

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, $L(P') = h^{-1}\big(L(P)\big)$. □