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.

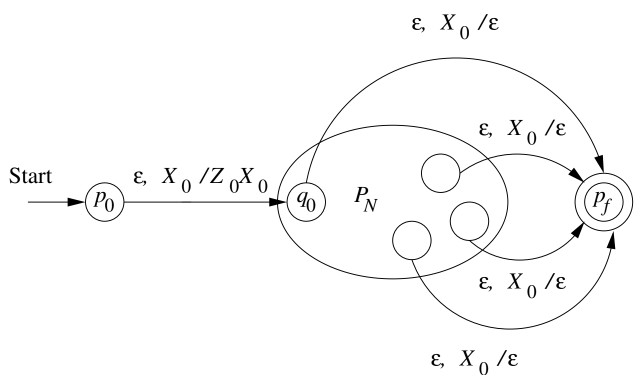

Figure 6.4: P_F simulates P_N and accepts if P_N empties its stack

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 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. $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$. 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 $(q_0, w, Z_0X_0) \overset{*}{\underset{P_N}{\vdash}} (q, \epsilon, X_0)$. Since by rule (2) above, P_F has all the moves of P_N, we may also conclude that $(q_0, w, Z_0X_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, X_0)$. If we put this sequence of moves together with the initial and final moves from rules (1) and (3) above, we get:

$$(p_0, w, X_0) \underset{P_F}{\vdash} (q_0, w, Z_0X_0) \overset{*}{\underset{P_F}{\vdash}} (q, \epsilon, X_0) \underset{P_F}{\vdash} (p_f, \epsilon, \epsilon) \tag{6.1}$$

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 $N(P_N)$. □

Example 6.10: Let us design a PDA that processes sequences of `if`'s and `else`'s in a C program, where i stands for `if` and e stands for `else`. Recall from Section 5.3.1 that there is a problem whenever the number of `else`'s in any prefix exceeds the number of `if`'s, because then we cannot match each `else` against its previous `if`. Thus, we shall use a stack symbol Z to count the difference between the number of i's seen so far and the number of e's. This simple, one-state PDA, is suggested by the transition diagram of Fig. 6.5.

We shall push another Z whenever we see an i and pop a Z whenever we see an e. Since we start with one Z on the stack, we actually follow the rule that if the stack is Z^n, then there have been $n-1$ more i's than e's. In particular, if the stack is empty, then we have seen one more e than i, and the input read so far has just become illegal for the first time. It is these strings that our PDA accepts by empty stack. The formal specification of P_N is:

$$P_N = (\{q\}, \{i, e\}, \{Z\}, \delta_N, q, Z)$$

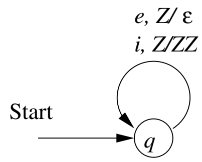

Figure 6.5: A PDA that accepts the if/else errors by empty stack

where δ_N is defined by:

1. $\delta_N(q, i, Z) = \{(q, ZZ)\}$. This rule pushes a Z when we see an i.
2. $\delta_N(q, e, Z) = \{(q, \epsilon)\}$. This rule pops a Z when we see an e.

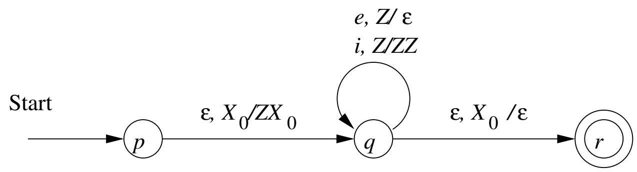

Figure 6.6: Construction of a PDA accepting by final state from the PDA of Fig. 6.5

Now, let us construct from P_N a PDA P_F that accepts the same language by final state; the transition diagram for P_F is shown in Fig. 6.6.[3] We introduce a new start state p and an accepting state r. We shall use X_0 as the bottom-of-stack marker. P_F is formally defined:

$$P_F = (\{p, q, r\}, \{i, e\}, \{Z, X_0\}, \delta_F, p, X_0, \{r\})$$

where δ_F consists of:

1. $\delta_F(p, \epsilon, X_0) = \{(q, ZX_0)\}$. This rule starts P_F simulating P_N, with X_0 as a bottom-of-stack-marker.
2. $\delta_F(q, i, Z) = \{(q, ZZ)\}$. This rule pushes a Z when we see an i; it simulates P_N.
3. $\delta_F(q, e, Z) = \{(q, \epsilon)\}$. This rule pops a Z when we see an e; it also simulates P_N.
4. $\delta_F(q, \epsilon, X_0) = \{(r, \epsilon)\}$. That is, P_F accepts when the simulated P_N would have emptied its stack.

□

[3]Do not be concerned that we are using new states p and r here, while the construction in Theorem 6.9 used p_0 and p_f. Names of states are arbitrary, of course.

6.2.4 From Final State to Empty Stack

Now, let us go in the opposite direction: take a PDA P_F that accepts a language L by final state and construct another PDA P_N that accepts L by empty stack. The construction is simple and is suggested in Fig. 6.7. From each accepting state of P_F, add a transition on ϵ to a new state p. When in state p, P_N pops its stack and does not consume any input. Thus, whenever P_F enters an accepting state after consuming input w, P_N will empty its stack after consuming w.

To avoid simulating a situation where P_F accidentally empties its stack without accepting, P_N must also use a marker X_0 on the bottom of its stack. The marker is P_N's start symbol, and like the construction of Theorem 6.9, P_N must start in a new state p_0, whose sole function is to push the start symbol of P_F on the stack and go to the start state of P_F. The construction is sketched in Fig. 6.7, and we give it formally in the next theorem.

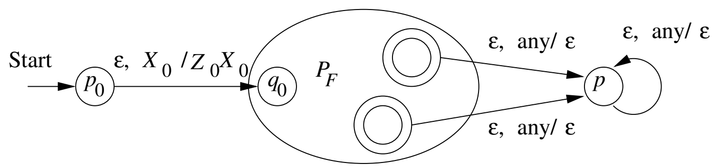

Figure 6.7: P_N simulates P_F and empties its stack when and only when P_N enters an accepting state

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 $(q, \epsilon, \alpha X_0)$.[4] 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 $L(P_F)$. $\Box$

6.2.5 Exercises for Section 6.2

Exercise 6.2.1: Design a PDA to accept each of the following languages. You may accept either by final state or by empty stack, whichever is more convenient.

* a) $\{0^n1^n \mid n \geq 1\}$.

b) The set of all strings of 0's and 1's such that no prefix has more 1's than 0's.

c) The set of all strings of 0's and 1's with an equal number of 0's and 1's.

! **Exercise 6.2.2:** Design a PDA to accept each of the following languages.

* a) $\{a^ib^jc^k \mid i = j \text{ or } j = k\}$. Note that this language is different from that of Exercise 5.1.1(b).

[4] Although α could be ϵ, in which case P_F has emptied its stack at the same time it accepts.