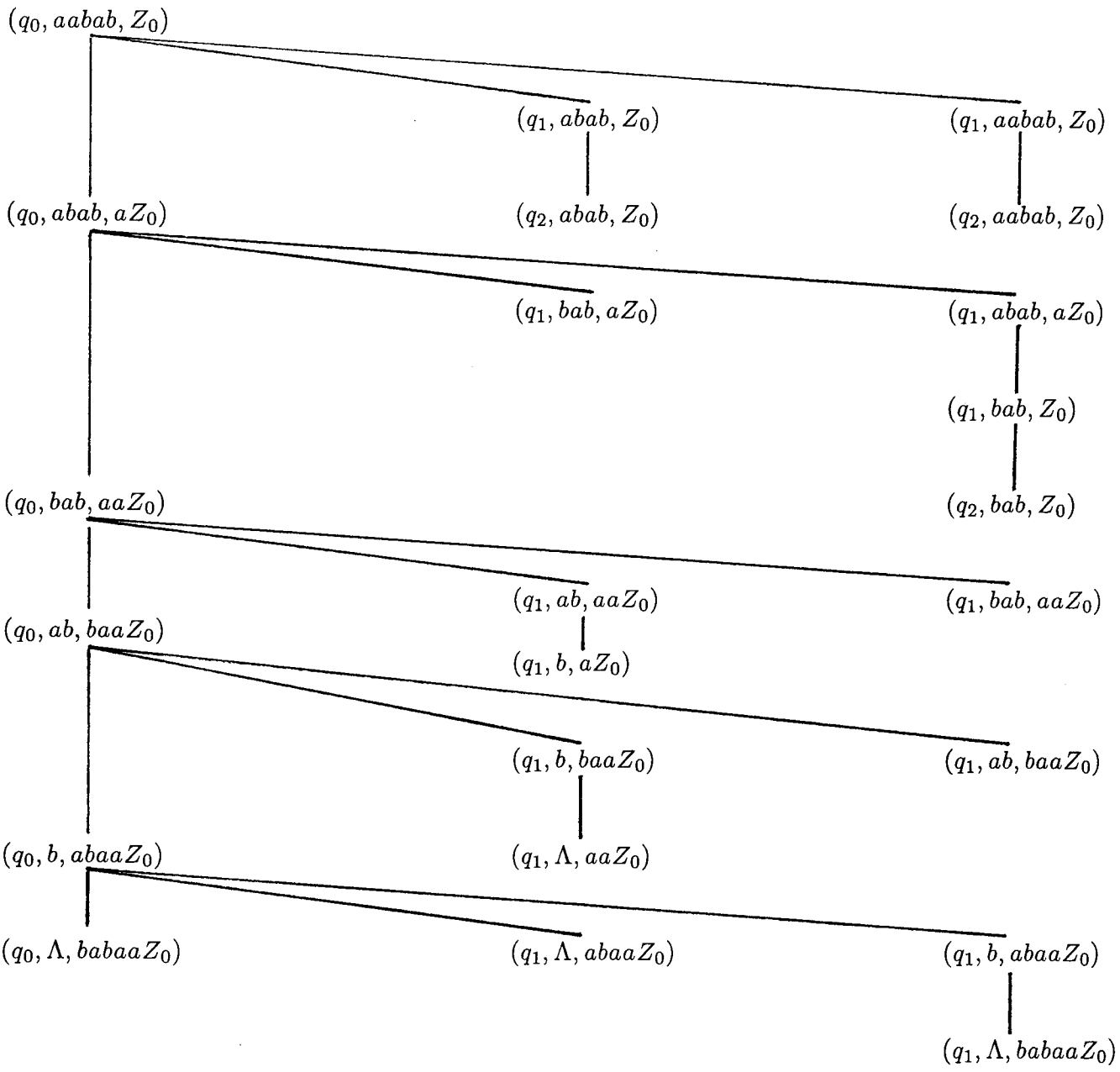
Chapter 7

Pushdown Automata

7.2. (for the string $aabab$)



7.3. $2n + 1$. One possible sequence is to push all the input symbols onto the stack. In addition, for each of the n symbols, there are the two sequences of moves in which the PDA guesses that the symbol is the middle symbol in an odd-length palindrome, and the first symbol in the second half of an even-length palindrome, respectively.

7.4.

(a)

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	(q_0, aZ_0)
2	q_0	b	Z_0	(q_0, bZ_0)
3	q_0	a	a	(q_0, aa)
4	q_0	b	a	(q_0, ba)
5	q_0	a	b	(q_0, ab)
6	q_0	b	b	(q_0, bb)
7	q_0	Λ	Z_0	(q_1, Z_0)
8	q_0	Λ	a	(q_1, a)
9	q_0	Λ	b	(q_1, b)
10	q_1	a	a	(q_1, Λ)
11	q_1	b	b	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)				none

(b)

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	$(q_0, aZ_0), (q_1, Z_0)$
2	q_0	b	Z_0	$(q_0, bZ_0), (q_1, Z_0)$
3	q_0	a	a	$(q_0, aa), (q_1, a)$
4	q_0	b	a	$(q_0, ba), (q_1, a)$
5	q_0	a	b	$(q_0, ab), (q_1, b)$
6	q_0	b	b	$(q_0, bb), (q_1, b)$
7	q_1	a	a	(q_1, Λ)
8	q_1	b	b	(q_1, Λ)
9	q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)				none

7.5. (a) In the following PDA q_3 is the only accepting state.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	$(q_0, xZ_0), (q_1, aZ_0)$
2	q_0	b	Z_0	$(q_0, xZ_0), (q_1, bZ_0)$
3	q_0	a	x	$(q_0, xx), (q_1, ax)$
4	q_0	b	x	$(q_0, xx), (q_1, bx)$
5	q_1	a	a	(q_1, a)
6	q_1	b	b	(q_1, b)
7	q_1	b	a	$(q_1, a), (q_2, \Lambda)$
8	q_1	a	b	$(q_1, b), (q_2, \Lambda)$
9	q_2	a	x	(q_2, Λ)
10	q_2	b	x	(q_2, Λ)
11	q_2	Λ	Z_0	(q_3, Z_0)
(all other combinations)				none

This can be understood as follows. In state q_0 the PDA initially reads input symbols and pushes an x onto the stack for each one read. At some point, it guesses that the next symbol read will be one that fails to match the corresponding symbol in the second half. It pushes that symbol onto the stack and enters the state q_1. In state q_1, the PDA reads input, leaving the stack alone, until it guesses that it has reached the input symbol that corresponds to (i.e., fails to match) the top stack symbol. At this point it enters q_2, having read that input symbol and popped the nonmatching stack symbol. From that point on, the PDA is concerned only with the length of the remaining input. It pops an x from the stack for each input symbol read until the stack is empty except for Z_0, and at this point it enters the accepting state.

(b) In the PDA below, both states are accepting. (A string not in the language will cause the machine to crash before all the input is read.)

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	(q_0, aZ_0)
2	q_0	a	a	(q_0, aa)
3	q_0	b	a	(q_1, Λ)
4	q_1	a	a	(q_1, Λ)
5	q_1	b	a	(q_1, Λ)
(all other combinations)				none

(c)

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	Λ	Z_0	$(q_1, Z_0), (q_4, Z_0)$
2	q_1	Λ	Z_0	(q_3, Z_0)
3	q_1	a	Z_0	(q_1, aZ_0)
4	q_1	a	a	(q_1, aa)
5	q_1	b	a	(q_2, Λ)
6	q_2	b	a	(q_2, Λ)
7	q_2	Λ	Z_0	(q_3, Z_0)
8	q_3	c	Z_0	(q_3, Z_0)
9	q_3	Λ	Z_0	(q_f, Z_0)
10	q_4	Λ	Z_0	(q_f, Z_0)
11	q_4	a	Z_0	(q_4, Z_0)
12	q_4	b	Z_0	(q_5, bZ_0)
13	q_5	b	b	(q_5, bb)
14	q_5	c	b	(q_6, Λ)
15	q_6	c	b	(q_6, Λ)
16	q_6	Λ	Z_0	(q_f, Z_0)
(all other combinations)				none

In this PDA, q_f is the accepting state. The machine makes an initial choice (move 1) as to whether it will compare the number of b's to the number of a's or to the number of

c's; q_1 and q_4 are the states that begin these respective processes. In q_1, one possibility is that there are no a's or b's at all; this is the reason for the Λ-transition in move 2. Otherwise, the PDA saves a's on the stack (moves 3 and 4), and when it sees a b pops a's off the stack to match inputs of b (moves 5 and 6). In either case (either from q_1 or from q_2), when the stack is empty except for Z_0 the PDA goes to q_3 (moves 2 and 7), from which the machine can move to the accepting state any time (move 9), and from which the only other move is to read a c, leaving the stack alone (move 8).

Starting in q_4 the moves are similar. The PDA moves to q_5 at any time, and the only move it can make before that is to read an a, leaving the stack alone. From q_5, the PDA pushes b's onto the stack, and when it sees a c it starts to remove b's from the stack to match the c's in the input. It enters the accepting state on a Λ-transition whenever the stack is empty except for Z_0.

(d) This language is the union of $L_1 = \{x \mid n_a(x) < n_b(x)\}$ and $L_2 = \{x \mid n_a(x) < n_c(x)\}$. Each can be accepted by a PDA similar to those in Example 7.4. For a general method of constructing a PDA to accept the union of two CFLs, see Exercise 7.14. (We could construct one directly by using the same approach as in part (c).)

7.6. (a) The language of strings over $\{a, b\}$ of length 2 or greater with first and last symbols the same.

(b) $\{xcy \mid x, y \in \{a, b\}^* \text{ and } |x| = |y|\}$.

7.7.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	c	Z_0	(q_2, Z_0)
2	q_0	a	Z_0	$(q_0, xA'Z_0)$
3	q_0	b	Z_0	$(q_0, xB'Z_0)$
4	q_0	a	x	(q_0, xA)
5	q_0	b	x	(q_0, xB)
6	q_0	c	x	(q_0, Λ)
7	q_0	a	A	(q_0, Λ)
8	q_0	b	B	(q_0, Λ)
9	q_0	a	A'	(q_2, Λ)
10	q_0	b	B'	(q_2, Λ)
(all other combinations)				none

Here the symbol x on top of the stack tells the PDA that the input symbol is to be pushed onto the stack (and x placed on top). If A or B is on top of the stack, with no x, the PDA should try to match the input symbol with the stack symbol. (Once the symbol c is encountered, x will be removed from the stack and never placed there again.) The symbols A' and B' are used instead of A and B in the first step so that when they are matched with an input symbol the PDA can go to the accepting state.

7.8. Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an FA accepting L. If we let p_0 and p_1 be the states

of the PDA, with p_0 being the initial state, then exactly one will be accepting: p_0 if $q_0 \in A$, and p_1 otherwise. The stack symbols of the PDA other than Z_0 will correspond to elements of Q. The PDA simulates the moves of M as follows. State p_0 and top stack symbol Z_0 is interpreted to mean that M is in state q_0. Thereafter, a move by M to state q is simulated by placing q on the stack (unless $q = q_0$, in which case Z_0 is placed on the stack), and going to the accepting state of the PDA if $q \in A$ and to the nonaccepting state otherwise. Thus, top stack symbol q is interpreted to mean that M is currently in state q.

7.9. We may construct an NFA-Λ M accepting L, by letting states of M be pairs (p, X), where p is a state of the PDA and X is a stack symbol. If p_0 is the initial state of the PDA, (p_0, Z_0) is the initial state of M, and a pair (p, X) is accepting if and only if p is an accepting state of the PDA. The PDA move from state p, using input a (either an alphabet symbol or Λ), with X on top of the stack, which moves to state q and places a string α on the stack, corresponds to the move $(p, X) \xrightarrow{a} (q, \sigma)$, where σ is the first symbol of α if $\alpha \neq \Lambda$ and X otherwise.

7.10. An NFA-Λ can be constructed to accept L, having as states pairs (q, α), where q is a state in M and α is a string of k or fewer stack symbols, representing the current contents of M's stack. Such a pair provides a complete description of M's current status, and for any such pair and any possible input (either Λ or an input symbol) it is possible using M's definition to specify the pairs that might result. Furthermore, there are only finitely many such pairs. The initial state of the NFA-Λ is (q_0, Z_0), where q_0 is the initial state of M, and the pairs (q, α) that are designated as accepting states are those for which q is an accepting state of M.

7.11. The original PDA M can be modified in two ways. The first is to have a preliminary sequence of moves that causes a new stack symbol U to be inserted underneath Z_0; the second is to add one more new state, i, having the property that once i is entered, the PDA continues to loop back to i on any input. The new machine enters the state i in either of two situations: when the top stack symbol is U (which in the original machine M would mean that the stack was empty), or as a result of any combination of state, input symbol, and stack symbol for which no move was defined in M. Since these modifications do not affect sequences of moves in M that end at an accepting state, and since they do not create any new sequences of moves leading to an accepting state, the new PDA still accepts the same language.

7.12. Any more general move, say $(q, \alpha) \in \delta(p, a, X)$, can be replaced by a sequence of moves of the restricted type that have the same ultimate effect. The new states introduced in order to accomplish this are specific to this move, and thus there is no change in the language accepted by the machine.

7.13. (a, b) In part (a), the state q_0 is the only accepting state. In part (b), the two states q_a and q_b are the accepting states. The approach taken by the PDA is similar to the one in Table 7.5.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	(q_a, Z_0)
2	q_0	b	Z_0	(q_b, Z_0)
3	q_a	a	Z_0	(q_a, aZ_0)
4	q_a	a	a	(q_a, aa)
5	q_a	b	a	(q_a, Λ)
6	q_a	b	Z_0	(q_0, Z_0)
7	q_b	b	Z_0	(q_b, bZ_0)
8	q_b	b	b	(q_b, bb)
9	q_b	a	b	(q_b, Λ)
10	q_b	a	Z_0	(q_0, Z_0)
(all other combinations)				none

(c) In the PDA shown, the initial state is q_0 and the only accepting state is q_1. Let us denote by x the current string. Similar to what we did in Table 7.5 (in Example 7.4), if the PDA is in state q_1, Z_0 on top of the stack means $n_a(x) = 2n_b(x) - 1$ and b on top of the stack means $n_a(x) \leq 2n_b(x) - 2$. Since we wish to compare $n_a(x)$ to $2n_b(x)$, we will essentially treat every b we read as if it is two b's. This means that if the stack has no a's on it (which means we are in state q_1 already), we push two b's onto the stack. If there is an a on top of the stack, then we pop it off; at that point, we pop a second a off if there is one, and otherwise we go to state q_1.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	(q_0, aZ_0)
2	q_0	b	Z_0	(q_1, bZ_0)
3	q_0	a	a	(q_0, aa)
4	q_0	b	a	(q_t, Λ)
5	q_t	Λ	a	(q_0, Λ)
6	q_t	Λ	Z_0	(q_1, Z_0)
7	q_1	a	Z_0	(q_0, Z_0)
8	q_1	b	Z_0	(q_1, bbZ_0)
9	q_1	b	b	(q_1, bbb)
10	q_1	a	b	(q_1, Λ)
(all other combinations)				none

(d) The accepting states in this PDA are q_0, q_3, and q_6. The state q_1 is for pushing a's onto the stack, q_2 is for matching a's on the stack with b's in the input, q_4 is for pushing subsequent b's onto the stack, and q_5 is for matching them with a's in the input.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	(q_1, aZ_0)
2	q_0	b	Z_0	(q_4, bZ_0)
3	q_1	a	a	(q_1, aa)
4	q_1	b	a	(q_2, Λ)
5	q_2	b	a	(q_2, Λ)
6	q_2	Λ	Z_0	(q_3, Z_0)
7	q_3	b	Z_0	(q_4, bZ_0)
8	q_4	b	b	(q_4, bb)
9	q_4	a	b	(q_5, Λ)
10	q_5	a	b	(q_5, Λ)
11	q_5	Λ	Z_0	(q_6, Z_0)
(all other combinations)				none

7.14. (a) In all three parts of this problem, we start by making sure (relabeling if necessary) that the states of the two machines don't overlap. We can take care of unions the same way we did for FAs. The composite machine can choose one of two Λ-transitions from the initial state: one to the initial state of M_1, the other to the initial state of M_2. Thereafter, it executes the moves of the appropriate M_i.

(b) The basic idea is the same as for FAs, which is that the composite machine acts like M_1 until it reaches an accepting state, then takes a Λ-transition to the initial state of M_2. However, we have to be more careful because of the stack. For example, it's possible for M_1 to accept a string but to have an empty stack when it's done processing that string; however, if our composite machine ever has an empty stack, it can't move. The solution is to fix it so that the composite machine can never empty its stack. This can be done as in Exercise 7.11, by inserting a new stack symbol Z under the initial stack symbol of M_1 and then returning to the initial state of M_1.

From that point on, the moves are the same as those of M_1, unless Z appears on top of the stack or unless the state is an accepting state of M_1. If Z is on top of the stack and the state is not an accepting state in M_1, the machine crashes. If it reaches an accepting state of M_1, then (regardless of what's on top of the stack) it takes a Λ-transition to the initial state of M_2 and pushes the initial stack symbol of M_2 onto the stack. From then on, it acts like M_2, except that it never removes the initial stack symbol of M_2 from the stack. (If M_2 ever removed its initial stack symbol, that would be the last move it made, so our new machine can achieve the same result as M_2 without ever removing this symbol.)

(c) Same general idea as (b). The new PDA has the same states as the original, except that there's one new state q, which is the only accepting state. Every time the machine is in q, it makes a Λ-transition to the initial state of the original machine, in which it also pushes two symbols onto the stack, first a new symbol Z and then the initial stack symbol of the original machine on top of Z. From that point, the machine makes the same moves as the original machine, except that if it ever sees Z on the stack and it is not in one of the accepting states of the original machine, it crashes. Whenever it's in one of the accepting states of the original machine, it has the same moves as originally, plus one more, which is a Λ-transition to q.

7.15. If M is a DPDA accepting L by empty stack, x and y are distinct strings in L, and x is a prefix of y, then the sequence of moves M must make in order to accept x leaves the stack empty, since $x \in L$. But then y cannot be accepted, since M can't move with an empty stack.

7.16. Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a DPDA accepting L. We wish to construct a DPDA M_1 that accepts $L\{\$\}$ by empty stack. M_1 has all the states in Q and two new ones, q_e and q_i; it has all the stack symbols in Γ as well as one new one U. It doesn't matter what its accepting states are, since the mode of acceptance doesn't involve states. Denote its transition function by δ_1.

q_i is the initial state of M_1, and from this state M_1 places the stack symbol U underneath Z_0 before entering q_0. For states in Q, inputs in $\Sigma \cup \{\Lambda\}$, and stack symbols in Γ, the moves of M_1 are identical to those of M. For every $q \in A$ and every symbol X in $\Gamma \cup \{U\}$, $\delta_1(q, \$, X) = \{(q_e, X)\}$. Finally, for every $X \in \Gamma \cup \{U\}$, $\delta_1(q_e, \Lambda, X) = \{(q_e, \Lambda)\}$.

It is clear that M_1 is deterministic since M is. For any $x \in L$, there is a sequence of moves of M that lead to an accepting state q; from q there is a Λ-transition to q_e on input \$; and from q_e there is a sequence of Λ-transitions that causes the stack to empty. Therefore, $x\$$ is accepted by M_1, by empty stack. Conversely, it is clear that the only way M_1 can empty its stack is by entering q_e first, and the only strings that cause it to do this are strings $x\$$, where x is accepted by M. Therefore, M_1 accepts the language $L\{\$\}$.

7.17. The property of *pal* that is used in the proof is the one established in Theorem 3.3: any two distinct strings in Σ^* are distinguishable with respect to *pal*. The argument in Theorem 3.3 can be adapted easily to show that the languages of even-length and odd-length palindromes both have this property. We show that the language in (d) also does.

Let $x, y \in \{0,1\}^*$ with $x \neq y$. If $|x| = |y|$, let $z = 0x0$. Then $xz = x0x0$ is obviously in the language. The string $yz = y0x0$, however, cannot be in the language: It cannot be of the form $ww^\sim$ because of the 0's, and it can't be of the form ww, since $x \neq y$. If the lengths are not equal, say $|x| < |y|$, then if $|y| - |x|$ is odd, the string x distinguishes x and y relative to the language, since $|yx|$ is odd. If $|y| - |x| = 2m$, then let $y = y_1\alpha\beta$, where $|y_1| = |x|$ and $|\alpha| = |\beta| = m$, and let $z = \beta\beta^\sim x\beta\beta^\sim$. Then $xz = (x\beta\beta^\sim)(x\beta\beta^\sim)$ is in the language. However, $yz = (y_1\alpha\beta\beta)(\beta^\sim x\beta\beta^\sim)$. This string cannot be of the form ww, because the first half ends with β and the second with $\beta^\sim$; and it can't be $ww^\sim$, because the preceding portions of the two halves agree. Therefore, z distinguishes x and y.

7.18. (a) The PDA works as follows. Instead of saving excess 0's or excess 1's on the stack, we save $*$'s, and use two different states to indicate which symbol there is currently a surplus of. The state q_0 is the initial state and the only accepting state.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	0	Z_0	$(p_0, *Z_0)$
2	q_0	1	Z_0	$(p_1, *Z_0)$
3	p_0	0	$*$	$(p_0, **)$
4	p_1	1	$*$	$(p_1, **)$
5	p_0	1	$*$	(p_0, Λ)
6	p_1	0	$*$	(p_1, Λ)
7	p_0	Λ	Z_0	(q_0, Z_0)
8	p_1	Λ	Z_0	(q_0, Z_0)
(all other combinations)				none

(b) For $x \in \{0,1\}^*$, let $d(x) = 2n_1(x) - n_0(x)$. We use the three states q_0, q_+, and q_- to indicate that the quantity $d(x)$ is currently 0, positive, or negative, respectively, and the number of $*$'s on the stack indicates the current absolute value of $d(x)$. q_+ is the accepting state.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	0	Z_0	$(q_-, *Z_0)$
2	q_0	1	Z_0	$(q_+, * * Z_0)$
3	q_-	0	$*$	$(q_-, **)$
4	q_-	1	$*$	(q_t, Λ)
5	q_t	Λ	Z_0	$(q_+, *Z_0)$
6	q_t	Λ	$*$	(q_-, Λ)
7	q_-	Λ	Z_0	(q_0, Z_0)
8	q_+	1	$*$	$(q_+, * * *)$
9	q_+	0	$*$	(q_0, Λ)
10	q_0	Λ	$*$	$(q_+, *)$
(all other combinations)				none

Lines 4-7 of the table are the moves required in the case where $d(x) < 0$ and we read the symbol 1. If there are at least two $*$'s on the stack, we pop two off, and if this empties the stack we go to q_0. If there is only one $*$ on the stack, then we go to q_-, ending up with one $*$ on the stack. The state q_t is a temporary state used in this procedure.

Note that it's possible for the PDA to be temporarily in state q_- when the stack is empty, but if this happens, the machine makes a Λ-transition to q_0. In move 9, we are forced to go to some state other than q_+, in case popping the $*$ would cause us accidentally to accept a string x for which $d(x) = 0$. This means that the PDA can also be temporarily in q_0 when there is at least one $*$ on the stack; but again, if this happens, we immediately make a Λ-transition back to q_+.

7.19. The PDA in Example 7.3 (Table 7.3) is such an example. The string }, for example, causes the machine to crash immediately.

7.20. Let us use {} and () as our two types of parentheses. One definition is to say that a string is balanced if three conditions hold: i) When {'s and }'s are ignored, the remainder

is a balanced string, as in Definition 6.5; ii) when ('s and)'s are ignored, the remainder is a balanced string; and iii) For any substring starting with a left parenthesis of either type and ending with its mate, no prefix has more right parentheses of the other type than left. Condition iii) is equivalent to saying that for each left parenthesis a of either type, the string of parentheses of the other type appearing between a and its mate is balanced.

7.21. (b)

$$
\begin{array}{lll}
 & (q_0, (a*a+a), Z_0) & \\
\vdash & (q_1, (a*a+a), SZ_0) & \qquad S \\
\vdash & (q_1, (a*a+a), (S)Z_0) & \Rightarrow \ (S) \\
\vdash & (q_1, a*a+a), S)Z_0) & \\
\vdash & (q_1, a*a+a), S+S)Z_0) & \Rightarrow \ (S+S) \\
\vdash & (q_1, a*a+a), S*S+S)Z_0) & \Rightarrow \ (S*S+S) \\
\vdash & (q_1, a*a+a), a*S+S)Z_0) & \Rightarrow \ (a*S+S) \\
\vdash & (q_1, a*a+a), S*S+S)Z_0) & \\
\vdash & (q_1, *a+a), *S+S)Z_0) & \\
\vdash & (q_1, a+a), S+S)Z_0) & \\
\vdash & (q_1, a+a), a+S)Z_0) & \Rightarrow \ (a*a+S) \\
\vdash & (q_1, +a), +S)Z_0) & \\
\vdash & (q_1, a), S)Z_0) & \\
\vdash & (q_1, a), a)Z_0) & \Rightarrow \ (a*a+a) \\
\vdash & (q_1,),)Z_0) & \\
\vdash & (q_1, \Lambda, Z_0) & \\
\vdash & (q_2, \Lambda, Z_0) &
\end{array}
$$

7.22. A sequence of moves causing aba to be accepted is

$$
\begin{aligned}
(q_0, ababa, Z_0) &\vdash (q_0, baba, aZ_0) \\
&\vdash (q_0, aba, baZ_0) \\
&\vdash (q_1, ba, baZ_0) \\
&\vdash (q_1, a, aZ_0) \\
&\vdash (q_1, \Lambda, Z_0) \\
&\vdash (q_2, \Lambda, \Lambda)
\end{aligned}
$$

The corresponding derivation in the CFG is as follows.

$$
\begin{aligned}
S \Rightarrow [q_0, Z_0, q_2] &\Rightarrow a[q_0, A, q_1][q_1, Z_0, q_2] \\
&\Rightarrow ab[q_0, B, q_1][q_1, A, q_1][q_1, Z_0, q_2] \\
&\Rightarrow aba[q_1, B, q_1][q_1, A, q_1][q_1, Z_0, q_2] \\
&\Rightarrow abab[q_1, A, q_1][q_1, Z_0, q_2] \\
&\Rightarrow ababa[q_1, Z_0, q_2] \\
&\Rightarrow ababa
\end{aligned}
$$

(As in Example 7.7, we use upper-case letters as stack symbols.)

7.23. The PDA is deterministic only if, for each variable A, there is at most one production with left side A. In this case, there is no choice whatsoever in a leftmost derivation; in other words, there is only one leftmost derivation possible, and only one string in the language.

7.24. (a)

Type of Move	State	Stack	Unread Input	Production used
(initial)	q	Z_0	[]][[]]	
reduce	q	SZ_0	[]][[]]	$\Rightarrow$ []][[]]
shift	q	$[SZ_0$	][[]]	
reduce	q	$S[SZ_0$	][[]]	$\Rightarrow S$][[]]
shift	q	$]S[SZ_0$	[[]]	
reduce	$q_{1,1}$	$S[SZ_0$	[[]]	$\Rightarrow S[S]$[[]]
	$q_{1,2}$	$[SZ_0$	[[]]	
	$q_{1,3}$	SZ_0	[[]]	
	q	SZ_0	[[]]	
shift	q	$[SZ_0$	[]]	
reduce	q	$S[SZ_0$	[]]	$\Rightarrow S$[[]]
shift	q	$[S[SZ_0$	]]	
reduce	q	$S[S[SZ_0$	]]	$\Rightarrow S[S$[]]
shift	q	$]S[S[SZ_0$	]	
reduce	$q_{1,1}$	$S[S[SZ_0$	]	$\Rightarrow S[S[S]]$
	$q_{1,2}$	$[S[SZ_0$	]	
	$q_{1,3}$	$S[SZ_0$	]	
	q	$S[SZ_0$	]	
shift	q	$]S[SZ_0$	Λ	
reduce	$q_{1,1}$	$S[SZ_0$	Λ	$S \Rightarrow S[S]$
	$q_{1,2}$	$[SZ_0$	Λ	
	$q_{1,3}$	SZ_0	Λ	
	q	SZ_0	Λ	
(pop S)	q_1	Z_0	Λ	
(accept)	q_2	Z_0	Λ	

7.25. If the PDA is deterministic, the resulting grammar is unambiguous. This could happen even with a nondeterministic PDA, however; it could happen that the PDA has a choice of moves at some point but that only one choice leads to acceptance.

7.27. (a) If we follow Example 7.8 and make the PDA operate by replacing any variable V on the stack by the right side of a production $V \rightarrow \alpha$, we obtain this PDA.

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, S_1\$)$
3	q_1	a	S_1	(q_a, AS_1)
4	q_1	b	S_1	(q_b, AS_1)
5	q_1	\$	S_1	$(q_\$, \Lambda)$
6	q_1	a	A	(q_a, aA)
7	q_1	b	A	(q_b, b)
8	q_1	a	a	(q_1, Λ)
9	q_1	b	b	(q_1, Λ)
10	q_1	\$	\$	(q_1, Λ)
11	q_a	Λ	A	(q_a, aA)
12	q_a	Λ	a	(q_1, Λ)
13	q_b	Λ	A	(q_b, b)
14	q_b	Λ	b	(q_1, Λ)
15	$q_\$$	Λ	\$	(q_1, Λ)
16	q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)				none

For example, in state q_1, if S_1 is on the stack and the next input is a, we first replace S_1 by AS_1, then (without reading any more input) replace A by aA, and finally pop the a, which matches the input symbol already read. The intermediate state q_a is used to accomplish these last two steps. It is obvious that the PDA could be made simpler and more efficient by compressing these steps into one move: S_1 is replaced on the stack by AS_1.

7.28. (c)

$$S \to S_1\$ \qquad S_1 \to abX \qquad X \to TX \mid \Lambda \qquad T \to aU \qquad U \to Tbb \mid b$$

7.29. The resulting grammar would be the same except that the last production $U \to b$ would be replaced by $U \to \Lambda$. Consider the string $abaabbaa\$$. The moves by which the nondeterministic PDA accepts this string are given below.

$(q_0, abaabbaa\$, S) \vdash (q_1, abaabbaa\$, S_1\$Z_0) \vdash (q_1, abaabbaa\$, abX\$Z_0)$
$\vdash (q_1, baabbaa\$, bX\$Z_0) \vdash (q_1, aabbaa\$, X\$Z_0) \vdash (q_1, aabbaa\$, TX\$Z_0)$
$\vdash (q_1, aabbaa\$, aUX\$Z_0) \vdash (q_1, abbaa\$, UX\$Z_0) \vdash (q_1, abbaa\$, TbbX\$Z_0)$
$\vdash (q_1, abbaa\$, aUbbX\$Z_0) \vdash (q_1, bbaa\$, UbbX\$Z_0) \vdash (q_1, bbaa\$, bbX\$Z_0)$
$\vdash (q_1, baa\$, bX\$Z_0) \vdash (q_1, aa\$, X\$Z_0) \vdash (q_1, aa\$, TX\$Z_0)$
$\vdash (q_1, aa\$, aUX\$Z_0) \vdash (q_1, a\$, UX\$Z_0) \vdash (q_1, a\$, X\$Z_0)$
$\vdash (q_1, a\$, TX\$Z_0) \vdash (q_1, a\$, aUX\$Z_0) \vdash (q_1, \$, UX\$Z_0)$
$\vdash (q_1, \$, X\$Z_0) \vdash (q_1, \$, \$Z_0) \vdash (q_1, \Lambda, Z_0)$
$\vdash (q_2, \Lambda, Z_0)$

At both places that are underlined, U is on top of the stack and a is the next input symbol, but the moves are different. This shows that the grammar is not LL(1).

7.32. The PDA executes a reduction whenever \$, a, or) is on top of the stack. In the case of), the reduction replaces either $(S_1 + S_1)$ or $(S_1 * S_1)$ by S_1; which one is determined entirely by what is on the stack. For any other stack symbol (either S_1 or Z_0), the correct move is to shift.

7.33. (a)

Move no.	State	Input	Stack symbol	Move(s)
		Shift moves		
1	q	σ	X	$(q, \sigma X)$
	(X any stack symbol other than] or \$)			
		Moves to reduce S_1\$ to S		
2	q	Λ	\$	$(q_{0,1}, \Lambda)$
3	$q_{0,1}$	Λ	S_1	(q, S)
		Moves to reduce some string to S_1		
4	q	Λ	]	$(q_{1,1}, \Lambda)$
5	$q_{1,1}$	Λ	[	$(q_{1,2}, \Lambda)$
6	$q_{1,2}$	Λ	S_1	(q, S_1)
7	$q_{1,2}$	Λ	X	(q, S_1X)
	(X any stack symbol other than S_1)			
8	$q_{1,1}$	Λ	S_1	$(q_{1,3}, \Lambda)$
9	$q_{1,3}$	Λ	[	$(q_{1,4}, \Lambda)$
10	$q_{1,4}$	Λ	S_1	(q, S_1)
11	$q_{1,4}$	Λ	X	(q, S_1X)
	(X any stack symbol other than S_1)			
		Move to accept		
12	q	Λ	S	(q_{acc}, Λ)
	(all other combinations)			none

(b) One reason is that if S_1 is on the stack and the next input symbol is], the next correct move might be a shift (if the S_1 is the one on the right side of the production $S \rightarrow [S_1]$, for example) or a reduction (if the S_1 is the one on the right side of the production $S \rightarrow []S_1$, for example). A string can easily be found so that both these moves occur in the course of accepting the string.

7.36. (a) The pairs (X, σ) for which it is correct to reduce when the top stack symbol is X and the next input is σ are the following: $(\$, \sigma)$ (for any input σ), (a, σ) (for any input σ), $(T, +)$, and $(T, \$)$.

7.37. (a)

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	$(q_1, aZ_0), (q_1, aaZ_0)$
2	q_1	a	a	$(q_1, aa), (q_1, aaa)$
3	q_1	b	a	(q_2, Λ)
4	q_2	b	a	(q_2, Λ)
5	q_2	Λ	Z_0	(q_3, Z_0)
(all other combinations)				none

The initial state is q_0 and the accepting states are q_0 and q_3. Each a is used to cancel either one or two b's.

(b)

Move no.	State	Input	Stack symbol	Move(s)
1	q_0	a	Z_0	(q_1, aZ_0)
2	q_0	b	Z_0	(q_0, bZ_0)
3	q_0	a	b	(q_1, Λ)
4	q_0	b	b	(q_0, bb)
5	q_1	a	Z_0	(q_2, aaZ_0)
6	q_1	b	Z_0	(q_1, bZ_0)
7	q_1	a	a	(q_2, aaa)
8	q_1	a	b	(q_2, a)
9	q_1	b	a	(q_1, Λ)
10	q_1	b	b	(q_1, bb)
11	q_2	a	Z_0	$(q_2, aZ_0), (q_2, aaZ_0)$
12	q_2	a	a	$(q_2, aa), (q_2, aaa)$
13	q_2	a	b	$(q_2, \Lambda), (q_2, a)$
14	q_2	b	a	(q_2, Λ)
15	q_2	b	b	(q_2, bb)
16	q_2	Λ	Z_0	(q_3, Z_0)
(all other combinations)				none

The initial state is q_0 and the accepting state q_3. The first a read will be used to cancel one b, the second will be used to cancel two b's, and subsequent a's can be used either way. (Note that every string in the language has at least two a's.)

The PDA stays in q_0 as long as no a's have been read. If and when the first a is read, it will be used to cancel a single b, either by removing b from the stack, or, if a is the first input symbol, saving it on the stack for a subsequent b to cancel. The state q_1 means that a single a has been read; if and when a second a is read, it will be used to cancel two b's (either by popping two b's from the stack, or by popping one from the stack and pushing an a to be canceled later, or by pushing two a's), and the PDA will go to state q_2. In q_2, each subsequent a will cancel either one or two b's, and the machine can enter q_3 if the stack is empty except for Z_0.

7.38. No. We can take any PDA that accepts a nonregular CFL, add a new nonaccepting state q_n, and allow the PDA to enter q_n immediately. Once it enters q_n, it simply reads the rest of the input and stays in q_n without changing the stack. The moves that allow the PDA to accepts strings are not changed. The effect is that for any input string, there is a rejecting sequence of moves that processes the string without adding any symbols to the stack.

7.39. Yes. We can carry out the construction in the solution to Exercise 7.10, with one addition. There is one additional state s to which any transition goes that would otherwise result in a pair (q, α) with $|\alpha| > k$. All transitions from s return to s. This guarantees that the NFA-Λ functions correctly, because for any input string x that is in L, there is a sequence of moves corresponding to x that never causes the NFA-Λ to enter s.

7.40. Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a DPDA accepting L. We construct a DPDA $M_1 = (Q_1, \Sigma, \Gamma, q_0', Z_0, A, \delta_1)$ accepting the new language as follows. $Q_1 = Q \cup Q_1$, where Q_1 is a set containing another copy q' of every $q \in Q$. (In particular, the initial state of M_1 is the primed version of q_0.) For each $q' \in Q'$, and each $a \in \Sigma \cup \{\Lambda\}$ and $X \in \Gamma$, $\delta_1(q', a, X) = \delta(q, a, X)'$. (In other words, for inputs that are Λ or ordinary alphabet symbols, the new PDA behaves the same way with the states q' that M does with the original states. In addition, if $q \in A$, then for every stack symbol X, $\delta_1(q', \#, X) = \{(q, X)\}$. Finally, δ_1 agrees with δ for elements of Q and inputs other than $\#$.

What this means is that M_1 acts the same way as M up to the point where the input $\#$ is encountered, except that the new states allow it to remember that it has not yet seen $\#$. Once this symbol is seen, if the current state is q' for some $q \in A$ (i.e., if M would have accepted the current string), then the machine switches over to the original states for the rest of the processing. It enters an accepting state subsequently if and only if both the substring preceding the $\#$ and the entire string except for the $\#$ would be accepted by M.

7.42. Let M be a PDA accepting L by empty stack. We wish to construct M_1 accepting L by final state. M_1 will have an extra state q_f, which will be the accepting state. There are no transitions from q_f. M_1 will also have a new initial state, and the first thing it will do from this state is to insert a new stack symbol U under Z_0 (necessary in order to allow M_1 to move to q_f after M's stack empties). It then enters the initial state of M.

The transitions of M_1 are exactly the same as those of M, for all the states in M, all inputs, and all stack symbols other than U. However, from every $q \in Q$, there is a Λ-transition to q_f if the top stack symbol is U. The new PDA M_1 accepts L, because for any string x that causes M's stack to empty, there is a sequence of moves by which x causes M_1 to move to q_f, and the only way M_1 can enter q_f is for it to have processed a string that causes M's stack to empty.

7.45. Suppose the two stack symbols other than Z_0 are 0 and 1. One approach is simply to encode the kth stack symbol by a string of k 1's, and to use 0's as separators. Auxiliary states can be used during the moves that pop the stack, in order to count the number of 1's, and during the moves that push a sequence of k 1's onto the stack.