7. PUSHDOWN AUTOMATA

7.1 Introduction by Way of an Example

- It is possible to extend the finite-state model of computation to create abstract machines that accept the context-free languages. In fact, such a machine can be constructed very simply from a CFG.

- **Example 7.1:** An Abstract Machine to Accept Simple Palindromes

  Let G be the context-free grammar having productions

  $S \rightarrow aSa \mid bSb \mid c$

  G generates the language

  $L = \{xcx^r \mid x \in \{a, b\}^*\}$

  The strings in L are odd-length palindromes over $\{a, b\}$, except that the middle symbol is c.

  Strings in L can be recognized using a single left-to-right pass. It will be necessary to save the symbols in the first half of a string so that they can be matched with symbols in the second half.

  Such an algorithm requires a "last in, first out" (LIFO) data structure—a *stack*. Items are always added ("pushed") and deleted ("popped") at one end (the *top*). The only element of the stack that is immediately accessible is the one on top.

  An abstract machine that incorporates a stack will still have a finite set of states. Each move will depend not only on the current state and input, but also on the symbol currently on top of the stack.

  The set Q of states for a machine that accepts L will need only three elements, q_0, q_1, and q_2.

  The machine stays in q_0 as long as it has not yet received the symbol c; when that happens, the machine moves to state q_1, leaving the stack unchanged. In state q_0, each input symbol is pushed onto the stack.

  In state q_1, each input symbol is compared to the symbol currently on top of the stack. If they agree, that symbol is popped off the stack and both are discarded; otherwise, the machine will crash and the string will not be accepted.

  The machine enters the accepting state, q_2, when the stack is empty.

Introduction by Way of an Example (Continued)

Each move of the abstract machine will be determined by three things:

1. The current state
2. The next input
3. The symbol on top of the stack

and will consist of two parts:

1. Changing states (or staying in the same state)
2. Replacing the top stack symbol by a string of zero or more symbols

The two basic stack moves are special cases: Popping the top symbol means replacing it by Λ, and pushing Y means replacing the top symbol X by YX (assuming that the left end of the string corresponds to the top).

The transition function for a finite automaton has the form

$$\delta : Q \times \Sigma \to Q$$

Here, if Γ is the stack alphabet, the transition function would seem to have the form

$$\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$$

For a state q, an input a, and a stack symbol X,

$$\delta(q, a, X) = (p, \alpha)$$

means that in state q, with X on top of the stack, the machine will read the symbol a, move to state p, and replace X on the stack by the string α.

There are problems with this definition, however.

First, how can a machine test whether the stack is empty? This problem can be avoided by placing a special *start symbol* Z_0 on the stack. Whenever Z_0 is on top, the stack is effectively empty.

Second, how can a machine move when the input is exhausted? The solution is to allow moves that use only Λ as input, corresponding to Λ-transitions in an NFA-Λ. This suggests that the transition function have the form

$$\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \to Q \times \Gamma^*$$

Introduction by Way of an Example (Continued)

It will also be useful for a machine to be able to crash (have no moves) or to be nondeterministic (have more than one possible move). The transition function now becomes

$\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \rightarrow$ the set of finite subsets of $Q \times \Gamma^*$

The "finite subsets" restriction is necessary because $Q \times \Gamma^*$ is an infinite set.

The machine that recognizes simple palindromes can now be precisely defined. Q will be the set $\{q_0, q_1, q_2\}$, q_0 is the initial state, and q_2 is the only accepting state. The input alphabet Σ is $\{a, b, c\}$, and the stack alphabet Γ is $\{a, b, Z_0\}$.

The transition function δ is given by the following table.

Move number	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	c	Z_0	(q_1, Z_0)
8	q_0	c	a	(q_1, a)
9	q_0	c	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

When a string is specified to be placed on the stack, the top of the stack corresponds to the left end of the string.

Introduction by Way of an Example (Continued)

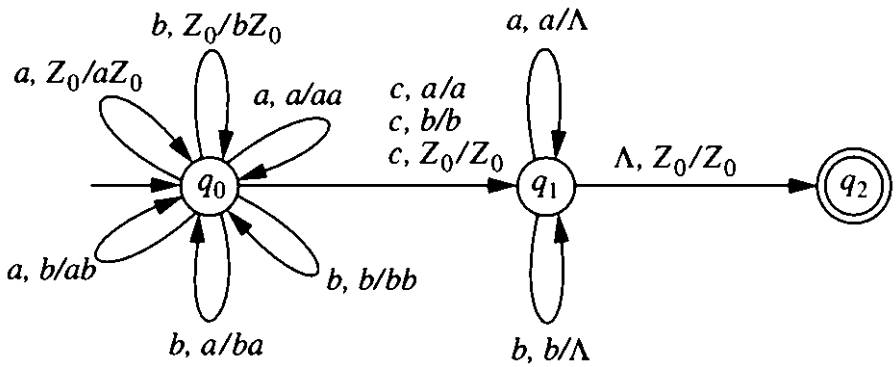
The following examples trace the moves of the machine for three input strings: *abcba*, *ab*, and *acaa*.

Move number	Resulting state	Unread input	Stack
(initially)	q_0	*abcba*	Z_0
1	q_0	*bcba*	aZ_0
4	q_0	*cba*	baZ_0
9	q_1	*ba*	baZ_0
11	q_1	*a*	aZ_0
10	q_1	–	Z_0
12	q_2	–	Z_0
(accept)			
(initially)	q_0	*ab*	Z_0
1	q_0	*b*	aZ_0
4	q_0	–	baZ_0
(crash)			
(initially)	q_0	*acaa*	Z_0
1	q_0	*caa*	aZ_0
8	q_1	*aa*	aZ_0
10	q_1	*a*	Z_0
12	q_2	*a*	Z_0

Note that *acaa* is not accepted. The machine reaches an accepting state, but fails to read the entire input string.

Introduction by Way of an Example (Continued)

It is possible to draw a transition diagram for this machine:



Each transition is labeled with an input symbol or Λ, a stack symbol *X*, a slash, and a string α of stack symbols, indicating that the transition may occur on the specified input and involves replacing *X* on the stack by α.

Such a diagram is not as useful as a transition diagram for an FA, because keeping track of the status of a computation involves remembering the stack contents as well as the state the machine is in.

7.2 The Definition of a Pushdown Automaton

- The following is a precise definition of the type of abstract machine illustrated in Example 7.1.

- **Definition 7.1:** A *pushdown automaton* (PDA) is a 7-tuple $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$, where

  Q is a finite set of states.
  Σ and Γ are finite sets (the input and stack alphabets, respectively).
  q_0, the initial state, is an element of Q.
  Z_0, the initial stack symbol, is an element of Γ.
  A, the set of accepting states, is a subset of Q.
  $\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \rightarrow$ the set of finite subsets of $Q \times \Gamma^*$.

  The function δ is called the transition function of M.

- Tracing the operation of a PDA involves keeping track of the current state and the stack contents. It is also helpful to monitor the portion of the input string yet to be read.

- A *configuration* of the PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is a triple

  (q, x, α)

  where $q \in Q$ is the current state, $x \in \Sigma^*$ is the string of remaining unread input, and $\alpha \in \Gamma^*$ is the current stack contents. The left end of α corresponds to the top of the stack.

- Writing

  $(p, x, \alpha) \vdash_M (q, y, \beta)$

  means that one of the possible moves in the first configuration takes M to the second. The move either consumes an input symbol or is a Λ-transition, so $x = ay$ for some $a \in \Sigma \cup \{\Lambda\}$.

  The string β is obtained from α by replacing the first symbol X by a string ξ (in other words, $\alpha = X\gamma$ for some $X \in \Gamma$ and some $\gamma \in \Gamma^*$, and $\beta = \xi\gamma$ for some $\xi \in \Gamma^*$), and $(q, \xi) \in \delta(p, a, X)$.

The Definition of a Pushdown Automaton (Continued)

- The notation

  $(p, x, \alpha) \vdash_M^* (q, y, \beta)$

  means that there is a sequence of zero or more moves that takes M from the first configuration to the second.

- If there is no possibility of confusion, $\vdash_M$ can be shortened to $\vdash$ and $\vdash_M^*$ can be shortened to $\vdash^*$.

- **Definition 7.2:** If $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is a PDA and $x \in \Sigma^*$, x is *accepted* by M if

  $(q_0, x, Z_0) \vdash_M^* (q, \Lambda, \alpha)$

  for some $\alpha \in \Gamma^*$ and some $q \in A$. A language $L \subseteq \Sigma^*$ is said to be accepted by M if L is precisely the set of strings accepted by M; in this case, we write $L = L(M)$.

- An *accepting configuration* is any configuration in which the state is an accepting state. This type of acceptance is sometimes called *acceptance by final state*.

- Section 7.5 discusses acceptance by *empty stack*. In this approach, a string is said to be accepted if it allows the PDA to reach a configuration in which the stack is empty, regardless of whether the state is an accepting state.

- The two types of acceptance are equivalent: if a language is accepted by some PDA using one mode of acceptance, there is another PDA using the other mode that also accepts the language.

A PDA Accepting the Language of Palindromes

- **Example 7.2:** A PDA Accepting the Language of Palindromes

  Let *pal* be the set of palindromes over $\{a, b\}$ (both even-length and odd-length), without the marker in the middle. A PDA that accepts *pal* will need to guess where the middle of the input string is.

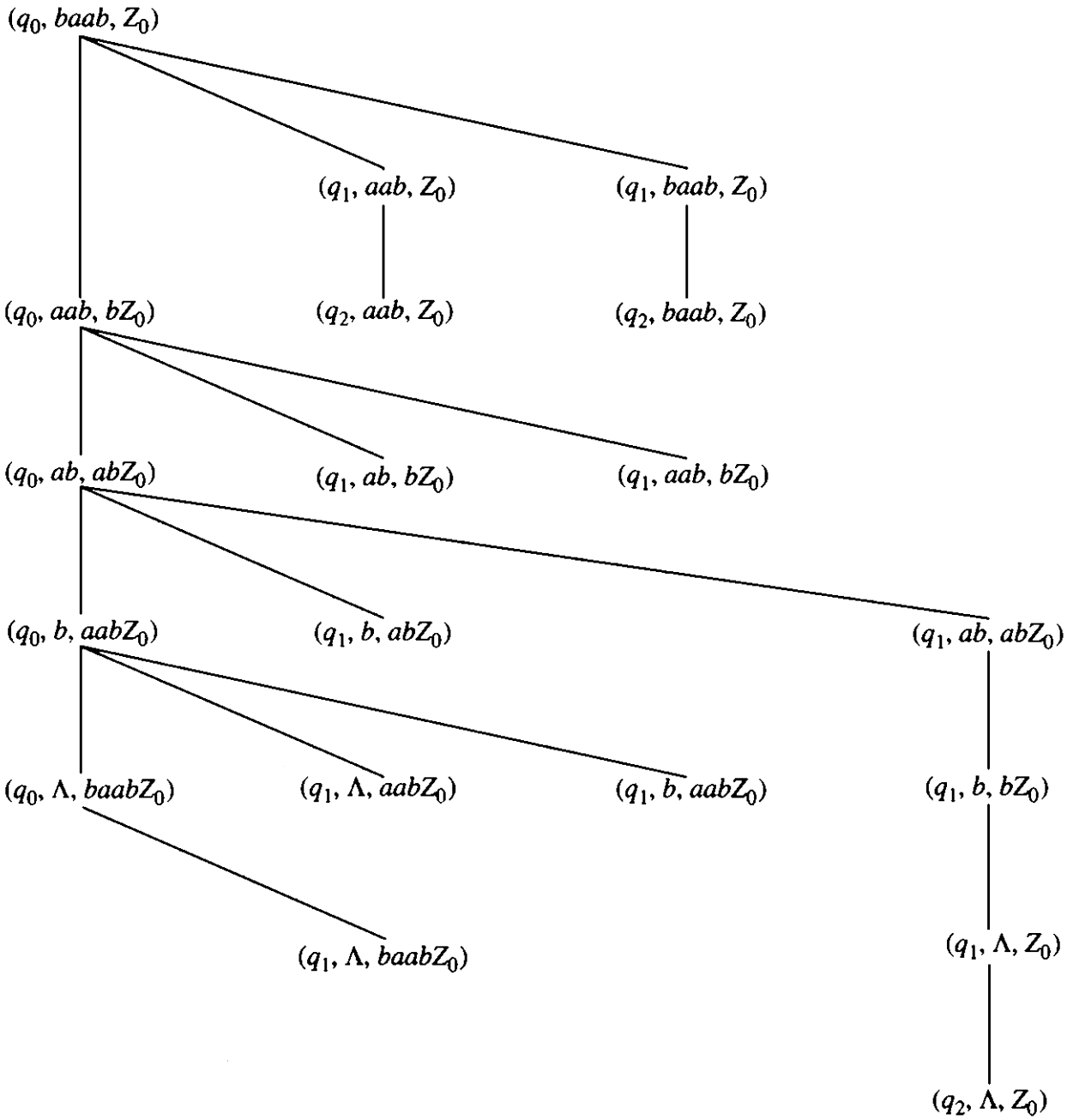
  In addition, the PDA will need to guess whether the input string has odd or even length. If it has odd length, the symbol in the middle should be discarded.

  A transition table for the PDA:

Move number	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_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

A PDA Accepting the Language of Palindromes (Continued)

A computation tree for a PDA shows the configuration at each step and the possible choices of moves at each step:



A PDA Accepting the Language of Palindromes (Continued)

The sequence of moves that leads to acceptance is

$(q_0, baab, Z_0) \vdash (q_0, aab, bZ_0)$
$\vdash (q_0, ab, abZ_0)$
$\vdash (q_1, ab, abZ_0)$
$\vdash (q_1, b, bZ_0)$
$\vdash (q_1, \Lambda, Z_0)$
$\vdash (q_2, \Lambda, Z_0)$ (accept)

7.3 Deterministic Pushdown Automata

- Some PDAs, such as the one in Example 7.1, are deterministic: they never have to guess which move to make.

- **Definition 7.3:** Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a pushdown automaton. M is *deterministic* if there is no configuration for which M has a choice of more than one move. In other words, M is deterministic if it satisfies both the following conditions.

  1. For any $q \in Q$, $a \in \Sigma \cup \{\Lambda\}$, and $X \in \Gamma$, the set $\delta(q, a, X)$ has at most one element.
  2. For any $q \in Q$ and $X \in \Gamma$, if $\delta(q, \Lambda, X) \neq \varnothing$, then $\delta(q, a, X) = \varnothing$ for every $a \in \Sigma$.

  A language L is a *deterministic context-free language* (DCFL) if there is a deterministic PDA (DPDA) accepting L.

- Note that the definition does not require the transition function to be defined for every combination of state, input, and stack symbol.

- The languages that can be accepted by PDAs are precisely the context-free languages. However, not every context-free language can be accepted by a deterministic PDA. In particular, the language of palindromes is not a DCFL.

Examples of Deterministic Pushdown Automata

- **Example 7.3:** A DPDA Accepting Balanced Strings of Brackets

  Consider the language L of all balanced strings involving two types of brackets: {} and []. L is generated by the context-free grammar with productions

  $S \rightarrow SS \mid [S] \mid \{S\} \mid \Lambda$

  A PDA that accepts L needs only two states: the initial state q_0, which is also the accepting state, and another state q_1.

  Left brackets of either type are saved on the stack, and one is discarded whenever it is on top of the stack and a right bracket of the same type is encountered in the input.

  Z_0 will be the top symbol whenever the string read so far is balanced. If this happens in state q_1 the PDA will return to the accepting state q_0 via a Λ-transition, leaving the stack unchanged.

  A transition table for the PDA:

Move number	State	Input	Stack symbol	Move
1	q_0	{	Z_0	$q_1, \{Z_0$
2	q_0	[	Z_0	$q_1, [Z_0$
3	q_1	{	{	$q_1, \{\{$
4	q_1	[	{	$q_1, [\{$
5	q_1	{	[	$q_1, \{[$
6	q_1	[	[	$q_1, [[$
7	q_1	}	{	q_1, Λ
8	q_1	]	[	q_1, Λ
9	q_1	Λ	Z_0	q_0, Z_0
(all other combinations)				none

  The parentheses that normally enclose each move have been omitted.

Examples of Deterministic Pushdown Automata (Continued)

The PDA will make the following moves when the input string is {[]}[]:

$(q_0, \{[]\}[], Z_0)$ $\vdash (q_1, []\}[], \{Z_0)$
$\vdash (q_1,]\}[], [\{Z_0)$
$\vdash (q_1, \}[], \{Z_0)$
$\vdash (q_1, [], Z_0)$
$\vdash (q_0, [], Z_0)$
$\vdash (q_1,], [Z_0)$
$\vdash (q_1, \Lambda, Z_0)$
$\vdash (q_0, \Lambda, Z_0)$ (accept)

Examples of Deterministic Pushdown Automata (Continued)

- **Example 7.4:** A DPDA to Accept Strings with More *a*'s Than *b*'s

  Consider the problem of building a DPDA for the following language:

  $L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$

  The following PDA uses the stack to save excess symbols of either type, so that they can eventually be matched by symbols of the opposite type:

Move number	State	Input	Stack symbol	Move
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_0, Λ)
4	q_0	b	b	(q_0, bb)
5	q_1	a	a	(q_1, aa)
6	q_1	b	a	(q_0, Λ)
7	q_0	Λ	a	(q_1, a)
(all other combinations)				none

  The only two states are the initial state q_0 and the accepting state q_1.

  At any point when the stack contains only Z_0, the string read so far has equal numbers of *a*'s and *b*'s. The machine enters the accepting state, q_1, whenever there is at least one *a* on the stack.

  Note that input *b* in state q_1 requires removing *a* from the stack and returning to q_0, at least temporarily. The machine uses a Λ-transition to return to q_1 once it determines that there are additional *a*'s remaining on the stack.

  If the PDA can determine in advance whether an *a* on the stack is the only one, then the Λ-transition can be eliminated. There are at least three natural ways to accomplish this:

  Have the PDA push *a*'s onto the stack only when there is an excess of at least two.
  Use a different stack symbol, say *A*, for the first extra *a*.
  Introduce a new state for the case in which there is exactly one extra *a*.

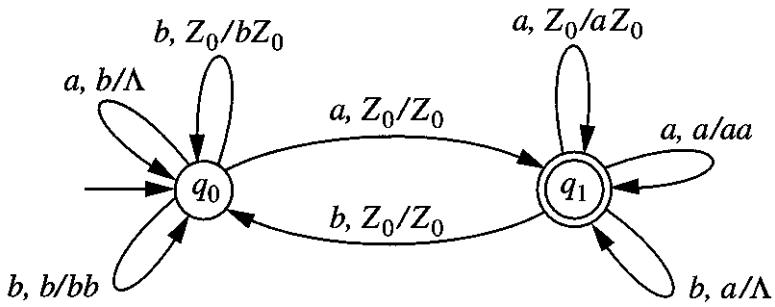
Examples of Deterministic Pushdown Automata (Continued)

The following DPDA takes the first approach:

Move number	State	Input	Stack symbol	Move
1	q_0	a	Z_0	(q_1, Z_0)
2	q_0	b	Z_0	(q_0, bZ_0)
3	q_0	a	b	(q_0, Λ)
4	q_0	b	b	(q_0, bb)
5	q_1	a	Z_0	(q_1, aZ_0)
6	q_1	b	Z_0	(q_0, Z_0)
7	q_1	a	a	(q_1, aa)
8	q_1	b	a	(q_1, Λ)
	(all other combinations)			none

As before, q_1 is the accepting state.

A transition diagram for the PDA:



The moves made by the PDA when the input string is *abbabaa*:

$(q_0, abbabaa, Z_0) \vdash (q_1, bbabaa, Z_0)$
$\vdash (q_0, babaa, Z_0)$
$\vdash (q_0, abaa, bZ_0)$
$\vdash (q_0, baa, Z_0)$
$\vdash (q_0, aa, bZ_0)$
$\vdash (q_0, a, Z_0)$
$\vdash (q_1, \Lambda, Z_0)$ (accept)

A Language That Cannot Be Accepted by a DPDA

- The following result shows that not every language that can be accepted by a PDA can be accepted by a deterministic PDA.

- **Theorem 7.1.** The language $pal = \{x \in \{a, b\}^* \mid x = x^r\}$ cannot be accepted by any DPDA.

  Proof: Suppose for the sake of contradiction that $M = (Q, \{a, b\}, \Gamma, q_0, Z_0, A, \delta)$ is a DPDA accepting *pal*. It is easy to modify M if necessary so that every move is either one of the form

  $\delta(p, s, X) = (q, \Lambda)$

  or one of the form

  $\delta(p, s, X) = (q, \alpha X)$

  where $s \in \{a, b, \Lambda\}$ and $\alpha \in \Gamma^*$. M can still remove a symbol from the stack or place another symbol on the stack, but it cannot do both in the same move.

  For any string x, M must eventually read every symbol of x, because x might be the beginning of a palindrome.

  After M has processed a string x, its stack will have a certain height. Consider how much shorter the stack can ever get as a result of reading subsequent input symbols.

  For each x, let y_x be a string for which this resulting stack height is as small as possible. (It cannot be 0, because M must still be able to process longer strings with prefix xy_x.) In other words,

  $(q_0, xy_x, Z_0) \vdash^* (q_x, \Lambda, \alpha_x)$

  for some state q_x and some string $\alpha_x \in \Gamma^*$ with $|\alpha_x| > 0$; and for any string y, any state p, and any string $\beta \in \Gamma^*$,

  if $(q_0, xy, Z_0) \vdash^* (p, \Lambda, \beta)$ then $|\beta| \geq |\alpha_x|$

A Language That Cannot Be Accepted by a DPDA (Continued)

Because of the initial assumption about the moves of M, any move that involves removing a stack symbol decreases the stack height. Therefore, once M reaches the configuration (q_x, Λ, α_x), as a result of processing the string xy_x, no symbols of the string α_x will ever be removed from the stack subsequently.

Let A_x be the first symbol of the string α_x. The set S of all strings of the form xy_x is infinite (because x can be of any length), and the set of all ordered pairs (q_x, A_x) is finite (because $Q \times \Gamma$ is finite). Therefore, there must be an infinite subset T of S so that for all the elements xy_x of T, the pairs (q_x, A_x) are equal.

In particular, it is possible to find two different strings $u_1 = xy_x$ and $u_2 = wy_w$ so that

$(q_0, u_1, Z_0) \vdash^* (q, \Lambda, A\beta_1)$

and

$(q_0, u_2, Z_0) \vdash^* (q, \Lambda, A\beta_2)$

for some $q \in Q$, some $A \in \Gamma$, and some strings $\beta_1, \beta_2 \in \Gamma^*$.

Now consider longer strings of the form u_1z and u_2z:

$(q_0, u_1z, Z_0) \vdash^* (q, z, A\beta_1)$

and

$(q_0, u_2z, Z_0) \vdash^* (q, z, A\beta_2)$

and the symbol A is never removed from the stack as a result of processing the string z. After z has been processed, the PDA will be in the same state regardless of whether the original input was u_1z or u_2z.

In other words, u_1 and u_2 are not distinguishable with respect to *pal*. This is a contradiction, because the proof of Theorem 3.3 showed that any two distinct strings are distinguishable with respect to *pal*.

7.4 A PDA Corresponding to a Given Context-Free Grammar

- Every context-free language can be recognized by a PDA. This can be shown by starting with a context-free grammar G and then building a PDA that can test an arbitrary string and determine whether it can be derived from G.

- The basic strategy is to *simulate* a derivation of the string in G. This will require guessing the steps of the derivation, so the PDA will be nondeterministic.

- As the simulation progresses, the machine will test the input string to make sure that it is still consistent with the derivation-in-progress. If the input string has a derivation in G, and if the PDA's guesses are the ones that correctly simulate this derivation, the machine will reach an accepting state.

- There are at least two natural ways a PDA can simulate a derivation in the grammar, corresponding to two techniques for constructing a derivation tree: *top-down* and *bottom-up*.

- With the **top-down** approach, the PDA first pushes the start symbol S onto the stack. The two types of moves made by the PDA, after S is placed on the stack, are

  1. Replace a variable A on top of the stack by the right side α of some production $A \rightarrow \alpha$.
  2. Pop a terminal symbol from the stack, provided it matches the next input symbol. Both symbols are then discarded.

- At each step, the string of input symbols already read, followed by the contents of the stack, exclusive of Z_0, constitutes the current string in the derivation.

- When a variable appears on top of the stack, it is the leftmost variable in the current string. Therefore, the derivation being simulated is a *leftmost* derivation.

- The PDA can accept when the stack contains only Z_0.

A PDA Corresponding to a Given Context-Free Grammar (Continued)

- **Definition 7.4:** Let $G = (V, \Sigma, S, P)$ be a context-free grammar. Define $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ as follows:

  $Q = \{q_0, q_1, q_2\}$ $A = \{q_2\}$ $\Gamma = V \cup \Sigma \cup \{Z_0\}$ (where $Z_0 \notin V \cup \Sigma$)

  M initially places S on the stack and moves to q_1: $\delta(q_0, \Lambda, Z_0) = \{(q_1, SZ_0)\}$. The only move to the accepting state q_2 is from q_1, when the stack is empty except for Z_0: $\delta(q_1, \Lambda, Z_0) = \{(q_2, Z_0)\}$. Otherwise, the only moves of M are:

  1. For every $A \in V$, $\delta(q_1, \Lambda, A) = \{(q_1, \alpha) \mid A \rightarrow \alpha$ is a production in $G\}$
  2. For every $a \in \Sigma$, $\delta(q_1, a, a) = \{(q_1, \Lambda)\}$

- **Theorem 7.2.** Let $G = (V, \Sigma, S, P)$ be a context-free grammar. Then the top-down PDA M described in Definition 7.4 accepts $L(G)$.

  Proof: **First, to show $L(G) \subseteq L(M)$.** If x is any string in $L(G)$, then a step in a leftmost derivation of x looks like

  $yA\alpha \Rightarrow yy'\beta$

  where y and y' are strings of terminals for which yy' is a prefix of x, and β either is Λ or begins with a variable.

  The goal is to show that some sequence of PDA moves reads the string yy', matching it with terminals from the stack, leaving the string β (or actually βZ_0) on the stack, and leaving the PDA in state q_1.

  It will follow, from the special case in which $\beta = \Lambda$, that with the input string x the PDA can reach the configuration (q_1, Λ, Z_0); therefore, because of the move to q_2 described in the definition, the PDA accepts x.

A PDA Corresponding to a Given Context-Free Grammar (Continued)

A precise version of the statement to be proved:

> For any $n \geq 1$, if $x \in L(G)$ and the nth step in a leftmost derivation of x is $yA\alpha \Rightarrow yy'\beta$, where $x = yy'z$, yy' is a string of terminals, and β either is Λ or begins with a variable, then
>
> $$(q_0, x, Z_0) = (q_0, yy'z, Z_0) \vdash^* (q_1, z, \beta Z_0) \qquad (7.1)$$

For the basis step of the proof, let $n = 1$. The first step in a derivation of x is to use a production of the form $S \rightarrow y'\beta$, so that the string y in (7.1) is null. Using the initial move in the definition of M and a move of type (1) yields

$$(q_0, x, Z_0) = (q_0, y'z, Z_0) \vdash (q_1, y'z, SZ_0) \vdash (q_1, y'z, y'\beta Z_0)$$

However, using as many moves of type (2) as there are symbols in y', the PDA is able to get from the last configuration shown to (q_1, z, β), and therefore

$$(q_0, x, Z_0) \vdash^* (q_1, z, \beta Z_0)$$

Now suppose that $k \geq 1$ and that the statement is true for every $n \leq k$. The goal is to show that if the $(k + 1)$th step in a leftmost derivation of x is $yA\alpha \Rightarrow yy'\beta$, then (7.1) is true. The kth step in the leftmost derivation of x has the form

$$wBy \Rightarrow ww'\beta' = ww'A\alpha$$

where $ww' = y$. This means that $x = ww'(y'z)$, and the induction hypothesis implies that

$$(q_0, x, Z_0) \vdash^* (q_1, y'z, A\alpha Z_0)$$

If the production used in the $(k + 1)$th step is $A \rightarrow \alpha'$, so that $\alpha'\alpha = y'\beta$, then using a move of type (1) gives

$$(q_1, y'z, A\alpha Z_0) \vdash (q_1, y'z, \alpha'\alpha Z_0) = (q_1, y'z, y'\beta Z_0)$$

Now, just as in the basis step, the configuration $(q_1, z, \beta Z_0)$ can be reached by using moves of type (2); this completes the proof that $L(G) \subseteq L(M)$.

A PDA Corresponding to a Given Context-Free Grammar (Continued)

Next, to show $L(M) \subseteq L(G)$. The statement to be proved is now the following:

> For any $n \geq 1$, if there is a sequence of n moves that leads from the configuration (q_0, x, Z_0) to the configuration $(q_1, z, \beta Z_0)$, then for some $y \in \Sigma^*$, $x = yz$ and $S \Rightarrow^* y\beta$.

The basis step is easy because the only single move producing a configuration of the right form is the initial move of M; in this case $z = x$ and $\beta = S$, so that y can be chosen to be Λ.

Suppose $k \geq 1$ and the statement is true for any $n \geq k$. Suppose also that some sequence of $k + 1$ moves produces the configuration $(q_1, z, \beta Z_0)$. The goal is to show that $x = yz$ for some y and that $S \Rightarrow^* y\beta$.

There are two possibilities for the $(k + 1)$th move in this sequence. One is that the next-to-last configuration is $(q_1, az, \alpha\beta Z_0)$ and the last move is of type (1). In this case the induction hypothesis implies that for some $y' \in \Sigma^*$, $x = y'az$ and $S \Rightarrow^* y'a\beta$. The desired conclusion follows by letting $y = y'a$.

The other possibility is that the next-to-last configuration is $(q_1, z, A\gamma Z_0)$ for some variable A for which there is a production $A \rightarrow \alpha$ with $\alpha\gamma = \beta$. In this case, the induction hypothesis says that $x = yz$ for some $y \in \Sigma^*$ and $S \Rightarrow^* yA\gamma$. But then

$$S \Rightarrow^* yA\gamma \Rightarrow y\alpha\gamma = y\beta$$

This completes the induction proof.

Now, if x is any string in $L(M)$,

$$(q_0, x, Z_0) \vdash^* (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, Z_0)$$

because M can enter q_2 only via a Λ-transition with Z_0 on top of the stack. Letting $y = x$ and $z = \beta = \Lambda$ in the statement that was proved by induction, the conclusion is that $S \Rightarrow^* x$ and therefore that $x \in L(G)$.

An Example of the Top-Down Approach

- **Example 7.5:** A Top-down PDA for the Strings with More *a*'s than *b*'s

  The language $L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$ can be generated by a context-free grammar with the following productions:

  $S \rightarrow a \mid aS \mid bSS \mid SSb \mid SbS$

  The construction in the proof of Theorem 7.2 yields a PDA with the following transition function:

State	Input	Stack symbol	Move(s)
q_0	Λ	Z_0	(q_1, SZ_0)
q_1	Λ	S	(q_1, a), (q_1, aS), (q_1, bSS), (q_1, SSb), (q_1, SbS)
q_1	a	a	(q_1, Λ)
q_1	b	b	(q_1, Λ)
q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)			none

  The following sequence of moves allows *M* to accept the string *abbaaa*:

  $(q_0, abbaaa, Z_0)$
  $\vdash (q_1, abbaaa, SZ_0)$ S
  $\vdash (q_1, abbaaa, SbSZ_0)$ $\Rightarrow SbS$
  $\vdash (q_1, abbaaa, abSZ_0)$ $\Rightarrow abS$
  $\vdash (q_1, bbaaa, bSZ_0)$
  $\vdash (q_1, baaa, SZ_0)$
  $\vdash (q_1, baaa, bSSZ_0)$ $\Rightarrow abbSS$
  $\vdash (q_1, aaa, SSZ_0)$
  $\vdash (q_1, aaa, aSZ_0)$ $\Rightarrow abbaS$
  $\vdash (q_1, aa, SZ_0)$
  $\vdash (q_1, aa, aSZ_0)$ $\Rightarrow abbaaS$
  $\vdash (q_1, a, SZ_0)$
  $\vdash (q_1, a, aZ_0)$ $\Rightarrow abbaaa$
  $\vdash (q_1, \Lambda, Z_0)$
  $\vdash (q_2, \Lambda, Z_0)$

The Bottom-Up Approach

- The opposite approach to top-down is **bottom-up.** Instead of replacing a variable A on the stack by the right side α of a production $A \rightarrow \alpha$, the PDA removes α from the stack and replaces it by (or "reduces it to") A.

- A bottom-up PDA also has moves that allow it to "shift" a terminal symbol from the input to the stack, in order to prepare for a reduction.

- Because shifting input symbols onto the stack reverses their order, the string α that is to be reduced to A will appear on the stack in reverse; thus the PDA begins the reduction with the *last* symbol of α on top of the stack.

- Note that a reduction step may require a sequence of moves, one for each symbol in the string α.

- The process terminates when the start symbol S, left on the stack by the last reduction, is popped off the stack and Z_0 is the only thing left.

- The entire process simulates a *rightmost* derivation, in reverse order, of the input string. At each step, the current string in the derivation is formed by the contents of the stack (in reverse), followed by the unread input.

An Example of the Bottom-Up Approach

- **Example 7.6:** A Bottom-up PDA for Simple Algebraic Expressions

  The following grammar G will be used to illustrate the bottom-up approach:

  (1) $S \rightarrow S + T$
  (2) $S \rightarrow T$
  (3) $T \rightarrow T * a$
  (4) $T \rightarrow a$

  Suppose the input string is $a + a * a$, which has the rightmost derivation

  $S \Rightarrow S + T$
  $\Rightarrow S + T * a$
  $\Rightarrow S + a * a$
  $\Rightarrow T + a * a$
  $\Rightarrow a + a * a$

  The following table shows the corresponding steps or groups of steps executed by the bottom-up PDA as the string $a + a * a$ is processed:

Move	Production	Stack	Unread input
–		Z_0	$a + a * a$
shift		aZ_0	$+ a * a$
reduce	$T \rightarrow a$	TZ_0	$+ a * a$
reduce	$S \rightarrow T$	SZ_0	$+ a * a$
shift		$+ SZ_0$	$a * a$
shift		$a + SZ_0$	$* a$
reduce	$T \rightarrow a$	$T + SZ_0$	$* a$
shift		$* T + SZ_0$	a
shift		$a * T + SZ_0$	–
reduce	$T \rightarrow T * a$	$T + SZ_0$	–
reduce	$S \rightarrow S + T$	SZ_0	–
(pop S)		Z_0	–
(accept)			

An Example of the Bottom-Up Approach (Continued)

The following table shows the details of the nondeterministic bottom-up PDA that carries out these moves:

State	Input	Stack symbol	Move(s)
Shift moves (σ and X are arbitrary)			
q	σ	X	$(q, \sigma X)$
Moves to reduce $S + T$ to S			
q	Λ	T	$(q_{1,1}, \Lambda)$
$q_{1,1}$	Λ	+	$(q_{1,2}, \Lambda)$
$q_{1,2}$	Λ	S	(q, S)
Moves to reduce T to S			
q_0	Λ	T	(q, S)
Moves to reduce $T * a$ to T			
q	Λ	a	$(q_{3,1}, \Lambda)$
$q_{3,1}$	Λ	*	$(q_{3,2}, \Lambda)$
$q_{3,2}$	Λ	T	(q, T)
Moves to reduce a to T			
q	Λ	a	(q, T)
Moves to accept			
q	Λ	S	(q_1, Λ)
q_1	Λ	Z_0	(q_2, Λ)
(all other combinations)			none

The shift moves allow the next input to be shifted onto the stack, regardless of the current stack symbol.

A reduction can begin when the top stack symbol is the last symbol of α, for some production $A \rightarrow \alpha$. If $|\alpha| > 1$, the PDA passes through a sequence of unique states (such as $q_{3,1}$ and $q_{3,2}$) as it removes the symbols of α from the stack.

A similar nondeterministic PDA can be constructed from any CFG.

7.5 A Context-Free Grammar Corresponding to a Given PDA

- Constructing a context-free grammar that generates the language accepted by a given PDA is more complicated than the construction in the other direction.

- To simplify the construction, it will be convenient to assume that the PDA accepts the language by empty stack. The following theorem states that this assumption can be made without loss of generality.

- **Theorem 7.3.** Suppose $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is a pushdown automaton accepting the language $L \subseteq \Sigma^*$. Then there is another PDA $M_1 = (Q_1, \Sigma, \Gamma_1, q_1, Z_1, A_1, \delta_1)$ accepting L by empty stack. In other words, for any string x, $x \in L$ if and only if $(q_1, x, Z_1) \vdash_{M_1}^* (q, \Lambda, \Lambda)$ for some state $q \in Q_1$.

  Sketch of proof: At the point when the original PDA M accepts a string, its stack may not be empty. M_1 will be constructed so that as the two machines M and M_1 process the same input string, M_1's stack will be empty precisely when M enters an accepting state.

  Suppose that M_1 is a replica of M, but with the ability to empty its stack automatically, without reading any more input, whenever M enters an accepting state. Then any time M enters an accepting state, the string read so far will be accepted by M_1 by empty stack.

  This is not quite sufficient, because M might crash (in a nonaccepting state) with an empty stack; in this case, since M_1 copies M exactly, its stack will be empty too, and it will therefore accept by empty stack a string that M does not accept.

  To avoid this, M_1 will need to start by placing on its stack a special symbol under the start symbol of M. This special symbol will allow M_1 to avoid emptying its stack until it is appropriate to do so.

  So that M_1 can empty its stack automatically when M enters an accepting state, it will have a Λ-transition from this state to a "stack-emptying" state, from which there are Λ-transitions that pop symbols off the stack until it is empty.

A Context-Free Grammar Corresponding to a Given PDA (Continued)

- Suppose that M is a PDA that accepts a language L by empty stack (represented by the notation $L = L_e(M)$). The goal is to construct a CFG that generates L.

- It will be helpful to define the CFG so that the current string in a leftmost derivation consists of two parts, the string of input symbols read by the PDA so far and a remaining portion corresponding to the current stack contents.

- The remaining portion will consist entirely of variables. In order to produce a string of terminals, all variables will eventually have to be eliminated, just as for an input string to be accepted by a PDA (by empty stack), all the symbols on the stack must eventually be popped off.

- An initial attempt at constructing a CFG from a PDA:

  Let the variables in the grammar be all possible stack symbols in the PDA, renamed if necessary so that no input symbols are included.
  Let the start symbol be Z_0.
  Ignore the states of the PDA completely.
  For each PDA move that reads a (either Λ or an element of Σ) and replaces A on the stack by $B_1B_2\ldots B_m$, introduce the production

  $A \rightarrow aB_1B_2\ldots B_m$

- This approach will allow the grammar to generate all strings accepted by the PDA. However, it may allow other strings to be derived as well.

A Context-Free Grammar Corresponding to a Given PDA (Continued)

- Consider the PDA in Example 7.1, which accepts the language $\{xcx^r \mid x \in \{a, b\}^*\}$. This PDA accepts by final state, rather than by empty stack, but that can be fixed by changing move 12 to

  $\delta(q_1, \Lambda, Z_0) = \{(q_1, \Lambda)\}$

  instead of $\{(q_2, Z_0)\}$. The PDA will also be modified to use A and B as stack symbols instead of a and b.

  The moves of the PDA include these:

  $\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$
  $\delta(q_0, c, A) = \{(q_1, A)\}$
  $\delta(q_1, a, A) = \{(q_1, \Lambda)\}$
  $\delta(q_1, A, Z_0) = \{(q_1, \Lambda)\}$

  Applying the construction just described yields the corresponding productions

  $Z_0 \rightarrow aAZ_0$
  $A \rightarrow cA$
  $A \rightarrow a$
  $Z_0 \rightarrow \Lambda$

  The string aca has the leftmost derivation

  $Z_0 \Rightarrow aAZ_0 \Rightarrow acAZ_0 \Rightarrow acaZ_0 \Rightarrow aca$

  corresponding to the sequence of moves

  $(q_0, aca, Z_0) \vdash (q_0, ca, AZ_0) \vdash (q_1, a, AZ_0) \vdash (q_1, \Lambda, Z_0) \vdash (q_1, \Lambda, \Lambda)$

  If the PDA is given the input string aa instead, the initial move is

  $(q_0, aa, Z_0) \vdash (q_0, a, AZ_0)$

  and at this point the machine crashes, because it is only in state q_1 that it is allowed to read a and replace A on the stack by Λ. However, the grammar also allows the derivation

  $Z_0 \Rightarrow aAZ_0 \Rightarrow aaZ_0 \Rightarrow aa$

A Context-Free Grammar Corresponding to a Given PDA (Continued)

- In order to eliminate this problem, the grammar must be modified so as to incorporate the states of the PDA. This is done by introducing variables of the form

  $[p, A, q]$

  where p and q are states.

- For the variable $[p, A, q]$ to be replaced by a (either Λ or a terminal symbol), it must be the case that there is a PDA move that reads a, pops A from the stack, and takes the machine from state p to state q.

- More general productions involving the variable $[p, A, q]$ are to be thought of as representing any *sequence* of moves that takes the PDA from state p to state q and has the ultimate effect of removing A from the stack.

- Suppose that there is a move from p to some state p_1 that reads a and replaces A on the stack by $B_1B_2\ldots B_m$. The grammar will need productions of the form

  $[p, A, q] \rightarrow a[p_1, B_1, p_2][p_2, B_2, p_3]\ldots[p_m, B_m, q]$

  for all possible sequences of states $p_2, \ldots, p_m$.

- A derivation in the grammar will begin with a production of the form

  $S \rightarrow [q_0, Z_0, q]$

  where S is the start symbol of the grammar and q is an arbitrary state of the PDA.

A Context-Free Grammar Corresponding to a Given PDA (Continued)

- **Theorem 7.4.** Let $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ be a pushdown automaton accepting a language L by empty stack; that is, $L = L_e(M)$. Then there is a context-free grammar G with $L(G) = L$.

  Proof: Define $G = (V, \Sigma, S, P)$ as follows:

  $V = \{S\} \cup \{[p, A, q] \mid A \in \Gamma, p, q \in Q\}$

  The set P contains the following productions and only these:

  1. For every $q \in Q$, the production $S \rightarrow [q_0, Z_0, q]$ is in P.
  2. For every $q, q_1 \in Q$, $a \in \Sigma \cup \{\Lambda\}$, and $A \in \Gamma$, if $\delta(q, a, A)$ contains (q_1, Λ), then the production $[q, A, q_1] \rightarrow a$ is in P.
  3. For every $q, q_1 \in Q$, $a \in \Sigma \cup \{\Lambda\}$, $A \in \Gamma$, and $m \geq 1$, if $\delta(q, a, A)$ contains $(q_1, B_1B_2\ldots B_m)$ for some $B_1, \ldots, B_m \in \Gamma$, then for every choice of $q_2, \ldots, q_{m+1} \in Q$, the production

     $[q, A, q_{m+1}] \Rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3]\ldots[q_m, B_m, q_{m+1}]$

     is in P.

  The main idea of the proof is to characterize the strings of terminals that can be derived from a variable $[q, A, q']$; specifically, to show that for any $q, q' \in Q$, $A \in \Gamma$, and $x \in \Sigma^*$,

  (1) $[q, A, q'] \Rightarrow^* x$ if and only if $(q, x, A) \vdash^* (q', \Lambda, \Lambda)$

  From this result, the theorem will follow. On the one hand, if $x \in L_e(M)$, then $(q_0, x, Z_0) \vdash^* (q, \Lambda, \Lambda)$ for some $q \in Q$; then (1) implies that $[q_0, Z_0, q] \Rightarrow^* x$; therefore, $x \in L(G)$, because a derivation can start with a production of type 1.

  On the other hand, if $x \in L(G)$, then the first step in any derivation of x must be $S \Rightarrow [q_0, Z_0, q]$ for some $q \in Q$, which means that $[q_0, Z_0, q] \Rightarrow^* x$. It then follows from (1) that $x \in L_e(M)$.

  Both parts of (1) are proved using mathematical induction. The notations $\Rightarrow^n$ and $\vdash^n$ (where n is a nonnegative integer) refer to n-step derivations in the grammar and sequences of n moves of the PDA, respectively.

A Context-Free Grammar Corresponding to a Given PDA (Continued)

The first goal is to show that for every $n \geq 1$,

(2) If $[q, A, q'] \Rightarrow^n x$, then $(q, x, A) \vdash^* (q', \Lambda, \Lambda)$

For the basis step of the proof, suppose that $[q, A, q'] \Rightarrow^1 x$. This one-step derivation can happen only with a production of type 2, and only if $x \in \Sigma \cup \{\Lambda\}$ and $\delta(q, x, A)$ contains (q', Λ). In this case, it is obvious that $(q, x, A) \vdash (q', \Lambda, \Lambda)$.

For the induction step, suppose that $k \geq 1$ and that whenever $[q, A, q'] \Rightarrow^n x$ for some $n \leq k$, $(q, x, A) \vdash^* (q', \Lambda, \Lambda)$. Now suppose that $[q, A, q'] \Rightarrow^{k+1} x$. The goal is to show that $(q, x, A) \vdash^* (q', \Lambda, \Lambda)$. Since $k \geq 1$, the first step of the derivation of x must be

$$[q, A, q'] \Rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3]\ldots[q_m, B_m, q']$$

for some $m \geq 1$, some $a \in \Sigma \cup \{\Lambda\}$, some sequence $B_1, B_2, \ldots, B_m \in \Gamma$, and some sequence $q_1, q_2, \ldots, q_m \in Q$, so that $\delta(q, a, A)$ contains $(q_1, B_1 \ldots B_m)$.

The remainder of the derivation takes each $[q_i, B_i, q_{i+1}]$ to some string x_i and $[q_m, B_m, q']$ to a string x_m. The strings $x_1, \ldots, x_m$ satisfy the formula $ax_1x_2\ldots x_m = x$, and each x_i is derived from its respective variable in k or fewer steps.

The induction hypothesis, therefore, implies that for each i with $1 \leq i \leq m - 1$,

$$(q_i, x_i, B_i) \vdash^* (q_{i+1}, \Lambda, \Lambda)$$

and that

$$(q_m, x_m, B_m) \vdash^* (q', \Lambda, \Lambda)$$

Suppose M is in the configuration $(q, x, A) = (q, ax_1x_2\ldots x_m, A)$. Because $\delta(q, a, A)$ contains $(q_1, B_1B_2\ldots B_m)$, M can go in one step to the configuration

$$(q_1, x_1x_2\ldots x_m, B_1B_2\ldots B_m)$$

M can then go in a sequence of steps to

$$(q_2, x_2\ldots x_m, B_2\ldots B_m)$$

then to $(q_3, x_3\ldots x_m, B_3\ldots B_m)$, and ultimately to (q', Λ, Λ). Thus the result (2) follows.

A Context-Free Grammar Corresponding to a Given PDA (Continued)

To complete the proof of (1), it is necessary to show that for every $n \geq 1$,

(3) If $(q, x, A) \vdash^n (q', \Lambda, \Lambda)$, then $[q, A, q'] \Rightarrow^* x$

If $n = 1$, a string x satisfying the hypothesis in (3) must be of length 0 or 1, and $\delta(q, x, A)$ must contain (q', Λ). In this case, a production of type 2 can be used to derive x from $[q, A, q']$.

For the induction step, suppose that $k \geq 1$ and that for any $n \leq k$, and combination of $q, q' \in Q$, $x \in \Sigma^*$, and $A \in \Gamma$, if $(q, x, A) \vdash^n (q', \Lambda, \Lambda)$, then $[q, A, q'] \Rightarrow^* x$.

Next suppose that $(q, x, A) \vdash^{k+1} (q', \Lambda, \Lambda)$; the goal is to show that $[q, A, q'] \Rightarrow^* x$. For some $a \in \Sigma \cup \{\Lambda\}$ and some $y \in \Sigma^*$, $x = ay$ and the first move is

$(q, x, A) = (q, ay, A) \vdash (q_1, y, B_1B_2\ldots B_m)$

Here $m \geq 1$, since $k \geq 1$ and the B_i's are elements of Γ. In other words, $\delta(q, a, A)$ contains $(q_1, B_1\ldots B_m)$. The k subsequent moves end in the configuration (q', Λ, Λ); therefore, for each i with $1 \leq i \leq m$ there must be intermediate points at which the stack contains precisely the string $B_iB_{i+1}\ldots B_m$.

For each such i, let q_i be the state M is in the first time the stack contains $B_i\ldots B_m$, and let x_i be the portion of the input string that is consumed in going from q_i to q_{i+1} (or, if $i = m$, in going from q_m to the configuration (q', Λ, Λ)). Then

$(q_i, x_i, B_i) \vdash^* (q_{i+1}, \Lambda, \Lambda)$

for each i with $1 \leq i \leq m - 1$, and

$(q_m, x_m, B_m) \vdash^* (q', \Lambda, \Lambda)$

where each indicated sequence of moves has k or fewer. By the induction hypothesis, $[q_i, B_i, q_{i+1}] \Rightarrow^* x_i$ for each i with $1 \leq i \leq m - 1$, and $[q_m, B_m, q'] \Rightarrow^* x_m$. Since $\delta(q, a, A)$ contains $(q_1, B_1\ldots B_m)$, it must be the case that

$[q, A, q'] \Rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3]\ldots[q_m, B_m, q']$

(this is a production of type 3), and so

$[q, A, q'] \Rightarrow^* ax_1x_2\ldots x_m = x$

An Example of Obtaining a CFG from a PDA

- **Example 7.7:** Obtaining a CFG from a PDA Accepting Simple Palindromes

  Let $L = \{xcx^r \mid x \in \{a, b\}^*\}$. The following PDA is similar to the one in Example 7.1, except that it uses uppercase letters for stack symbols and accepts by empty stack.

Move number	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	c	Z_0	(q_1, Z_0)
8	q_0	c	A	(q_1, A)
9	q_0	c	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_1, Λ)
(all other combinations)				none

  In the grammar $G = (V, \Sigma, S, P)$ obtained from the construction in Theorem 7.4, V contains S as well as every object of the form $[p, X, q]$, where X is a stack symbol and p and q can each be either q_0 or q_1.

An Example of Obtaining a CFG from a PDA (Continued)

P contains all productions with one of the following forms:

(0) $S \rightarrow [q_0, Z_0, q]$
(1) $[q_0, Z_0, q] \rightarrow a[q_0, A, p][p, Z_0, q]$
(2) $[q_0, Z_0, q] \rightarrow b[q_0, B, p][p, Z_0, q]$
(3) $[q_0, A, q] \rightarrow a[q_0, A, p][p, A, q]$
(4) $[q_0, A, q] \rightarrow b[q_0, B, p][p, A, q]$
(5) $[q_0, B, q] \rightarrow a[q_0, A, p][p, B, q]$
(6) $[q_0, B, q] \rightarrow b[q_0, B, p][p, B, q]$
(7) $[q_0, Z_0, q] \rightarrow c[q_1, Z_0, q]$
(8) $[q_0, A, q] \rightarrow c[q_1, A, q]$
(9) $[q_0, B, q] \rightarrow c[q_1, B, q]$
(10) $[q_1, A, q_1] \rightarrow a$
(11) $[q_1, B, q_1] \rightarrow b$
(12) $[q_1, Z_0, q_1] \rightarrow \Lambda$

Allowing all combinations of p and q gives 35 productions in all.

The PDA accepts the string *bacab* by the following sequence of moves:

$$
\begin{aligned}
(q_0, bacab, Z_0) &\vdash (q_0, acab, BZ_0) \\
&\vdash (q_0, cab, ABZ_0) \\
&\vdash (q_1, ab, ABZ_0) \\
&\vdash (q_1, b, BZ_0) \\
&\vdash (q_1, \Lambda, Z_0) \\
&\vdash (q_1, \Lambda, \Lambda)
\end{aligned}
$$

The corresponding leftmost derivation in the grammar is

$$
\begin{aligned}
S &\Rightarrow [q_0, Z_0, q] \\
&\Rightarrow b[q_0, B, q_1][q_1, Z_0, q_1] \\
&\Rightarrow ba[q_0, A, q_1][q_1, B, q_1][q_1, Z_0, q_1] \\
&\Rightarrow bac[q_1, A, q_1][q_1, B, q_1][q_1, Z_0, q_1] \\
&\Rightarrow baca[q_1, B, q_1][q_1, Z_0, q_1] \\
&\Rightarrow bacab[q_1, Z_0, q_1] \\
&\Rightarrow bacab
\end{aligned}
$$