3.3 PUSHDOWN AUTOMATA

Not every context-free language can be recognized by a finite automaton, since, as we have already seen, some context-free languages are not regular. What sort of more powerful device could be used for recognizing arbitrary context-free languages? Or, to be a bit more specific, what extra features do we need to add to the finite automata so that they accept any context-free language?

To take a particular example, consider $\{ww^R : w \in \{a,b\}^*\}$. It *is* context-free, since it is generated by the grammar with rules $S \to aSa$, $S \to bSb$, and $S \to e$. It would seem that any device that recognizes the strings in this language by reading them from left to right must "remember" the first half of the input string so that it can check it —in reverse order— against the second half of the input. It is not surprising that this function cannot be performed by a finite automaton. If, however, the machine is capable of accumulating its input string as it is read, appending symbols one at a time to a stored string (see Figure 3-8), then it could nondeterministically guess when the center of the input has been reached and thereafter check the symbols off from its memory one at a time. The storage device need not be a general-purpose one. A sort of "stack" or "pushdown store," allowing read and write access only to the top symbol, would do nicely.

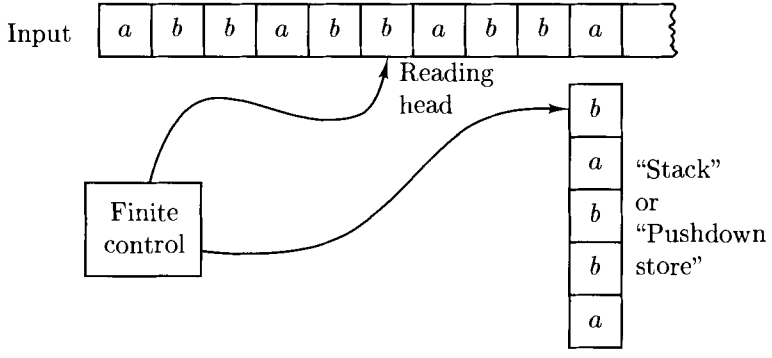

Figure 3-8

To take another example, the set of strings of balanced parentheses (Example 3.1.4) is also nonregular. However, computer programmers are familiar with a simple algorithm for recognizing this language: Start counting at zero, add one for every left parenthesis, and subtract one for every right parenthesis. If the count either goes negative at any time, or ends up different from zero, then the string should be rejected as unbalanced; otherwise it should be accepted. Now,

a counter can be considered as a special case of a stack, on which only one kind of symbol can be written.

To address this question from yet another point of view, rules such as $A \to aB$ are easy to simulate by a finite automaton, as follows: "If in state A reading a, go to state B." But what about a rule whose right-hand side is not a terminal followed by a nonterminal, say the rule $A \to aBb$? Certainly the machine must again go from state A to state B reading a, but what about b? What further action would allow us to remember the presence of b in this rule? A stack would be handy here: By pushing b on the top of a stack, we resolve to remember it and act upon it when it resurfaces again —presumably to be checked against a b in the input.

The idea of an automaton with a stack as auxiliary storage can be formalized as follows.

Definition 3.3.1: Let us define a **pushdown automaton** to be a sextuple $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where

K is a finite set of **states**,
Σ is an alphabet (the **input symbols**),
Γ is an alphabet (the **stack symbols**),
$s \in K$ is the **initial state**,
$F \subseteq K$ is the set of **final states**, and
Δ, the **transition relation**, is a finite subset of $(K \times (\Sigma \cup \{e\}) \times \Gamma^*) \times (K \times \Gamma^*)$.

Intuitively, if $((p, a, \beta), (q, \gamma)) \in \Delta$, then M, whenever it is in state p with β at the top of the stack, may read a from the input tape (if $a = e$, then the input is not consulted), replace β by γ on the top of the stack, and enter state q. Such a pair $((p, a, \beta), (q, \gamma))$ is called a **transition** of M; since several transitions of M may be simultaneously applicable at any point, the machines we are describing are nondeterministic in operation. (We shall later alter this definition to define a more restricted class, the deterministic pushdown automata.)

To **push** a symbol is to add it to the top of the stack; to **pop** a symbol is to remove it from the top of the stack. For example, the transition $((p, u, e), (q, a))$ pushes a, while $((p, u, a), (q, e))$ pops a.

As is the case with finite automata, during a computation the portion of the input already read does not affect the subsequent operation of the machine. Accordingly, a **configuration** of a pushdown automaton is defined to be a member of $K \times \Sigma^* \times \Gamma^*$: The first component is the state of the machine, the second is the portion of the input yet to be read, and the third is the contents of the pushdown store, read top-down. For example, if the configuration were (q, w, abc), the a would be on the top of the stack and the c on the bottom. If (p, x, α) and (q, y, ζ)

are configurations of M, we say that (p, x, α) **yields in one step** (q, y, ζ) (notation: $(p, x, \alpha) \vdash_M (q, y, \zeta)$) if there is a transition $((p, a, \beta), (q, \gamma)) \in \Delta$ such that $x = ay$, $\alpha = \beta\eta$, and $\zeta = \gamma\eta$ for some $\eta \in \Gamma^*$. We denote the reflexive, transitive closure of $\vdash_M$ by $\vdash_M^*$. We say that M **accepts** a string $w \in \Sigma^*$ if and only if $(s, w, e) \vdash_M^* (p, e, e)$ for some state $p \in F$. To put it another way, M accepts a string w if and only if there is a sequence of configurations $C_0, C_1, \ldots, C_n$ $(n > O)$ such that $C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_n$, $C_0 = (s, w, e)$, and $C_n = (p, e, e)$ for some $p \in F$. Any sequence of configurations $C_0, C_1, \ldots, C_n$ such that $C_i \vdash_M C_{i+1}$ for $i = 0, \ldots, n-1$ will be called a **computation** by M; it will be said to have **length** n, or to have n **steps**. The **language accepted** by M, denoted $L(M)$, is the set of all strings accepted by M.

When no confusion can result, we write $\vdash$ and $\vdash^*$ instead of $\vdash_M$ and $\vdash_M^*$.

Example 3.3.1: Let us design a pushdown automaton M to accept the language $L = \{wcw^R : w \in \{a, b\}^*\}$. For example, *ababcbaba* $\in L$, but *abcab* $\notin L$, and *cbc* $\notin L$. We let $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where $K = \{s, f\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{a, b\}$, $F = \{f\}$, and Δ contains the following five transitions.

(1) $((s, a, e), (s, a))$
(2) $((s, b, e), (s, b))$
(3) $((s, c, e), (f, e))$
(4) $((f, a, a), (f, e))$
(5) $((f, b, b), (f, e))$

This automaton operates in the following way. As it reads the first half of its input, it remains in its initial state s and uses transitions 1 and 2 to transfer symbols from the input string onto the pushdown store. Note that these transitions are applicable regardless of the current content of the pushdown store, since the "string" to be matched on the top of the pushdown store is the empty string. When the machine sees a c in the input string, it switches from state s to state f without operating on its stack. Thereafter only transitions 4 and 5 are operative; these permit the removal of the top symbol on the stack, provided that it is the same as the next input symbol. If the input symbol does not match the top symbol on the stack, no further operation is possible. If the automaton reaches in this way the configuration (f, e, e) —final state, end of input, empty stack —then the input was indeed of the form wcw^R, and the automaton accepts. On the other hand, if the automaton detects a mismatch between input and stack symbols, or if the input is exhausted before the stack is emptied, then it does not accept.

To illustrate the operation of M, we describe a sequence of transitions for the input string *abbcbba*.

Example 3.3.2: Now we construct a pushdown automaton to accept $L = \{ww^R : w \in \{a, b\}^*\}$. That is, the strings accepted by this machine are the

State	Unread Input	Stack	Transition Used
s	$abbcbba$	e	-
s	$bbcbba$	a	1
s	$bcbba$	ba	2
s	$cbba$	bba	2
f	bba	bba	3
f	ba	ba	5
f	a	a	5
f	e	e	4

same as those accepted by the machine of the previous example, except that the symbol c that marked the center of the strings is missing. Therefore the machine must "guess" when it has reached the middle of the input string and change from state s to state f in a nondeterministic fashion. Thus $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where $K = \{s, f\}$, $\Sigma = \{a, b\}$, $F = \{f\}$, and Δ is the set of the following five transitions.

(1) $((s, a, e), (s, a))$
(2) $((s, b, e), (s, b))$
(3) $((s, e, e), (f, e))$
(4) $((f, a, a), (f, e))$
(5) $((f, b, b), (f, e))$

Thus this machine is identical to that of the last example, except for transition 3. Whenever the machine is in state s, it can nondeterministically choose either to push the next input symbol onto the stack, or to switch to state f without consuming any input. Therefore even starting from a string of the form ww^R, M has computations that do not lead it to the accepting configuration (f, e, e); but there is some computation that leads M to this configuration if and only if the input string is of this form.$\Diamond$

Example 3.3.3: This pushdown automaton accepts the language $\{w \in \{a, b\}^* :$ w has the same number of a's and b's$\}$. Either a string of a's or a string of b's is kept by M on its stack. A stack of a's indicates the excess of a's over b's thus far read, if in fact M has read more a's than b's; a stack of b's indicates the excess of b's over a's. In either case, M keeps a special symbol c on the bottom of the stack as a marker. Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where $K = \{s, q, f\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, c\}$, $F = \{f\}$, and Δ is listed below.

(1) $((s, e, e), (q, c))$
(2) $((q, a, c), (q, ac))$
(3) $((q, a, a), (q, aa))$

(4) $((q, a, b), (q, e))$
(5) $((q, b, c), (q, bc))$
(6) $((q, b, b), (q, bb))$
(7) $((q, b, a), (q, e))$
(8) $((q, e, c), (f, e))$

Transition 1 puts M in state q while placing a c on the bottom of the stack. In state q, when M reads an a, it either starts up a stack of a's from the bottom, while keeping the bottom marker (transition 2), or pushes an a onto a stack of a's (transition 3), or pops a b from a stack of b's (transition 4). When reading a b from the input, the machine acts analogously, pushing a b onto a stack of b's or a stack consisting of just a bottom marker, and popping an a from a stack of a's (transitions 5, 6, and 7). Finally, when c is the topmost (and therefore the only) symbol on the stack, the machine may remove it and pass to a final state (transition 8). If at this point all the input has been read, then the configuration (f, e, e) has been reached, and the input string is accepted.

The following table contains an illustration of the operation of M.

State	Unread Input	Stack	Transition	Comments
s	$abbbabaa$	e	-	Initial configuration.
q	$abbbabaa$	c	1	Bottom marker.
q	$bbbabaa$	ac	2	Start a stack of a's.
q	$bbabaa$	c	7	Remove one a.
q	$babaa$	bc	5	Start a stack of b's.
q	$abaa$	bbc	6	
q	baa	bc	4	
q	aa	bbc	6	
q	a	bc	4	
q	e	c	4	
f	e	e	8	Accepts.

$\diamondsuit$

Example 3.3.4: Every finite automaton can be trivially viewed as a pushdown automaton that never operates on its stack. To be precise, let $M = (K, \Sigma, \Delta, s, F)$ be a nondeterministic finite automaton, and let M' be the pushdown automaton $(K, \Sigma, \emptyset, \Delta', s, F)$, where $\Delta' = \{((p, u, e), (q, e)) : (p, u, q) \in \Delta\}$. In other words, M' always pushes and pops an empty string on its stack, and otherwise simulates the transitions of M. Then it is immediate that M and M' accept precisely the same language.$\diamondsuit$

Problems for Section 3.3

3.3.1. Consider the pushdown automaton $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where

$$\begin{aligned} K &= \{s, f\}, \\ F &= \{f\}, \\ \Sigma &= \{a, b\}, \\ \Gamma &= \{a\}, \\ \Delta &= \{((s, a, e), (s, a)), ((s, b, e), (s, a)), ((s, a, e), (f, e)), \\ &\quad ((f, a, a), (f, e)), ((f, b, a), (f, e))\}. \end{aligned}$$

(a) Trace all possible sequences of transitions of M on input aba.
(b) Show that $aba, aa, abb \notin L(M)$, but $baa, bab, baaaa \in L(M)$.
(c) Describe $L(M)$ in English.

3.3.2. Construct pushdown automata that accept each of the following.
(a) The language generated by the grammar $G = (V, \Sigma, R, S)$, where

$$\begin{aligned} V &= \{S, (,), [,]\}, \\ \Sigma &= \{(,), [,]\}, \\ R &= \{S \rightarrow e, \\ &\quad S \rightarrow SS, \\ &\quad S \rightarrow [S], \\ &\quad S \rightarrow (S)\}. \end{aligned}$$

(b) The language $\{a^m b^n : m \leq n \leq 2m\}$.
(c) The language $\{w \in \{a, b\}^* : w = w^R\}$.
(d) The language $\{w \in \{a, b\}^* :$ w has twice as many b's as a's$\}$.

3.3.3. Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ be a pushdown automaton. The **language accepted by M by final state** is defined as follows:

$$L_f(M) = \{w \in \Sigma^* : (s, w, e) \vdash_M^* (f, e, \alpha) \text{ for some } f \in F, \alpha \in \Gamma^*\}.$$

a) Show that there is a pushdown automaton M' such that $L(M') = L_f(M)$.
b) Show that there is a pushdown automaton M'' such that $L_f(M'') = L(M)$.

3.3.4. Let $M = (K, , \Sigma, \Gamma, \Delta, s, F)$ be a pushdown automaton. The **language accepted by M by empty store** is defined as follows:

$$L_e(M) = \{w \in \Sigma^* : (s, w, e) \vdash_M^* (q, e, e) \text{ for some } q \in K\}.$$

(a) Show that there is a pushdown automaton M' such that $L_e(M') = L(M)$.
(b) Show that there is a pushdown automaton M'' such that $L(M'') = L_e(M)$.
(c) Show by a counterexample that it is not necessarily the case that $L_e(M) = L(M) \cup \{e\}$.

3.4 PUSHDOWN AUTOMATA AND CONTEXT-FREE GRAMMARS

In this section we show that the pushdown automaton is exactly what is needed to accept arbitrary context-free languages. This fact is of mathematical and practical significance: mathematical, because it knits together two different formal views of the same class of languages; and practical, because it lays the foundation for the study of syntax analyzers for "real" context-free languages such as programming languages (more on this in Section 3.7).

Theorem 3.4.1: *The class of languages accepted by pushdown automata is exactly the class of context-free languages.*

Proof: We break this proof into two parts.

Lemma 3.4.1: *Each context-free language is accepted by some pushdown automaton.*

Proof: Let $G = (V, \Sigma, R, S)$ be a context-free grammar; we must construct a pushdown automaton M such that $L(M) = L(G)$. The machine we construct has only two states, p and q, and remains permanently in state q after its first move. Also, M uses V, the set of terminals and nonterminals, as its stack alphabet. We let $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, where Δ contains the following transitions:

(1) $((p, e, e), (q, S))$
(2) $((q, e, A), (q, x))$ for each rule $A \to x$ in R.
(3) $((q, a, a), (q, e))$ for each $a \in \Sigma$.

The pushdown automaton M begins by pushing S, the start symbol of G, on its initially empty pushdown store, and entering state q (transition 1). On each subsequent step, it either replaces the topmost symbol A on the stack,

provided that it is a nonterminal, by the right-hand side x of some rule $A \to x$ in R (transitions of type 2), or pops the topmost symbol from the stack, provided that it is a terminal symbol that matches the next input symbol (transitions of type 3). The transitions of M are designed so that the pushdown store during an accepting computation mimics a leftmost derivation of the input string; M intermittently carries out a step of such a derivation on the stack, and between such steps it strips away from the top of the stack any terminal symbols and matches them against symbols in the input string. Popping the terminals from the stack has in turn the effect of exposing the leftmost nonterminal, so that the process can continue.

Example 3.4.1: Consider the grammar $G = (V, \Sigma, R, S)$ with $V = \{S, a, b, \mathrm{c}\}$, $\Sigma = \{a, b, c\}$, and $R = \{S \to aSa, S \to bSb, S \to c)$, which generates the language $\{wcw^R : w \in \{a, b\}^*\}$. The corresponding pushdown automaton, according to the construction above, is $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, with

$$\begin{aligned}
\Delta = \{&((p, e, e), (q, S)), && (T1)\\
&((q, e, S), (q, aSa)), && (T2)\\
&((q, e, S), (q, bSb)), && (T3)\\
&((q, e, S), (q, c)), && (T4)\\
&((q, a, a), (q, e)), && (T5)\\
&((q, b, b), (q, e)), && (T6)\\
&((q, \mathrm{c}, \mathrm{c}), (q, e))\} && (T7).
\end{aligned}$$

The string $abbcbba$ is accepted by M through the following sequence of moves.

State	Unread Input	Stack	Transition Used
p	$abbcbba$	e	-
q	$abbcbba$	S	T1
q	$abbcbba$	aSa	T2
q	$bbcbba$	Sa	T5
q	$bbcbba$	$bSba$	T3
q	$bcbba$	Sba	T6
q	$bcbba$	$bSbba$	T3
q	$cbba$	$Sbba$	T6
q	$cbba$	$cbba$	T4
q	bba	bba	T7
q	ba	ba	T6
q	a	a	T6
q	e	e	T5

Compare this to the operation, on the same string, of the pushdown automaton of Example 3.3.1.◇

To continue the proof of the Lemma, in order to establish that $L(M) = L(G)$, we prove the following claim.

Claim. Let $w \in \Sigma^*$ and $\alpha \in (V - \Sigma)V^* \cup \{e\}$. Then $S \overset{L}{\Rightarrow}^* w\alpha$ if and only if $(q, w, S) \vdash_M^* (q, e, \alpha)$.

This claim will suffice to establish Lemma 3.4.1, since it will follow (by taking $\alpha = e$) that $S \overset{L}{\Rightarrow}^* w$ if and only if $(q, e, S) \vdash_M^* (q, e, e)$ —in other words, $w \in L(G)$ if and only if $w \in L(M)$.

(Only if) Suppose that $S \overset{L}{\Rightarrow}^* w\alpha$, where $w \in \Sigma^*$, and $\alpha \in (V - \Sigma)V^* \cup \{e\}$. We shall show by induction on the length of the leftmost derivation of w from S that $(q, w, S) \vdash_M^* (q, e, \alpha)$.

Basis Step. If the derivation is of length 0, then $w = e$, and $\alpha = S$, and hence indeed $(q, w, S) \vdash_M^* (q, e, \alpha)$.

Induction Hypothesis. Assume that if $S \overset{L}{\Rightarrow}^* w\alpha$ by a derivation of length n or less, $n \geq 0$, then $(q, w, S) \vdash_M^* (q, e, \alpha)$.

Induction Step. Let

$$S = u_0 \overset{L}{\Rightarrow} u_1 \overset{L}{\Rightarrow} \cdots \overset{L}{\Rightarrow} u_n \overset{L}{\Rightarrow} u_{n+1} = w\alpha$$

be a leftmost derivation of $w\alpha$ from S. Let A be the leftmost nonterminal of u_n. Then $u_n = xA\beta$, and $u_{n+1} = x\gamma\beta$, where $x \in \Sigma^*$, $\beta, \gamma \in V^*$, and $A \to \gamma$ is a rule in R. Since there is a leftmost derivation of length n of $u_n = xA\beta$ from S, by the induction hypothesis

$$(q, x, S) \vdash_M^* (q, e, A\beta). \tag{2}$$

Since $A \to \gamma$ is a rule in R,

$$(q, e, A\beta) \vdash_M (q, e, \gamma\beta), \tag{3}$$

by a transition of type 2.

Now notice that u_{n+1} is $w\alpha$, but it is also $x\gamma\beta$. Hence, there is a string $y \in \Sigma^*$ such that $w = xy$ and $y\alpha = \gamma\beta$. Thus, we can rewrite (2) and (3) above as

$$(q, w, S) \vdash_M^* (q, y, \gamma\beta). \tag{4}$$

However, Since $y\alpha = \gamma\beta$,

$$(q, y, \gamma\beta) \vdash_M^* (q, e, \alpha), \tag{5}$$

by a sequence of $|y|$ transitions of type 3. Combining (4) and (5) completes the induction step.

(If) Now suppose that $(q, w, S) \vdash_M^* (q, e, \alpha)$ with $w \in \Sigma^*$ and $\alpha \in (V - \Sigma)V^* \cup \{e\}$; we show that $S \overset{L}{\Rightarrow}^* w\alpha$. Again, the proof is by induction, but this time *on the number of transitions of type 2* in the computation by M.

Basis Step. Since the first move in any computation is by a transition of type 2, if $(q, w, S) \vdash_M^* (q, e, \alpha)$ with no type-2 transitions, then $w = e$ and $\alpha = S$, and the result is true.

Induction Hypothesis. If $(q, w, S) \vdash_M^* (q, e, \alpha)$ by a computation with n type 2 steps or fewer, $n \geq 0$, then $S \overset{L}{\Rightarrow}^* w\alpha$.

Induction Step. Suppose that $(q, w, S) \vdash_M^* (q, e, \alpha)$ in $n + 1$ type-2 transitions, and consider the next-to-last such transition, say,

$$(q, w, S) \vdash_M^* (q, y, A\beta) \vdash_M (q, y, \gamma\beta) \vdash_M^* (q, e, \alpha),$$

where $w = xy$ for some $x, y \in \Sigma^*$, and $A \to \gamma$ is a rule of the grammar. By the induction hypothesis we have that $S \overset{L}{\Rightarrow}^* xA\beta$, and thus $S \overset{L}{\Rightarrow}^* x\gamma\beta$. Since however $(q, y, \gamma\beta) \vdash_M^* (q, e, \alpha)$, presumably by transitions of type 3, it follows that $y\alpha = \gamma\beta$, and thus $S \overset{L}{\Rightarrow}^* xy\alpha = w\alpha$. This completes the proof of Lemma 3.4.1, and with it half the proof of Theorem 3.4.1. ■

We now turn to the proof of the other half of Theorem 3.4.1.

Lemma 3.4.2: *If a language is accepted by a pushdown automaton, it is a context-free language.*

Proof: It will be helpful to restrict somewhat the pushdown automata under consideration. Call a pushdown automaton **simple** if the following is true:

> Whenever $((q, a, \beta), (p, \gamma))$ is a transition of the pushdown automaton and q is *not* the start state, then $\beta \in \Gamma$, and $|\gamma| \leq 2$.

In other words, the machine always consults its topmost stack symbol (and no symbols below it), and replaces it either with e, or with a single stack symbol, or with two stack symbols. Now it is easy to see that no interesting pushdown automaton can have only transitions of this kind, because then it would not be able to operate when the stack is empty (for example, it would not be able to *start* the computation, since in the beginning the stack is empty). This is why we do not restrict transitions from the start state.

We claim that if a language is accepted by an unrestricted pushdown automaton, then it is accepted by a simple pushdown automaton. To see this, let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ be any pushdown automaton; we shall construct a simple pushdown automaton $M' = (K', \Sigma, \Gamma \cup \{Z\}, \Delta', s', \{f'\})$ that also accepts $L(M)$;