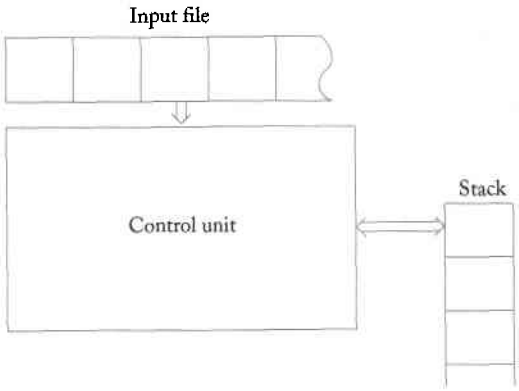
Chapter 7

Pushdown Automata

The description of context-free languages by means of context-free grammars is convenient, as illustrated by the use of BNF in programming language definition. The next question is whether there is a class of automata that can be associated with context-free languages. As we have seen, finite automata cannot recognize all context-free languages. Intuitively, we understand that this is because finite automata have strictly finite memories, whereas the recognition of a context-free language may require storing an unbounded amount of information. For example, when scanning a string from the language $L = \{a^n b^n : n \geq 0\}$, we must not only check that all a's precede the first b, we must also count the number of a's. Since n is unbounded, this counting cannot be done with a finite memory. We want a machine that can count without limit. But as we see from other examples, such as $\{ww^R\}$, we need more than unlimited counting ability: we need the ability to store and match a sequence of symbols in reverse order. This suggests that we might try a stack as a storage mechanism, allowing unbounded storage that is restricted to operating like a stack. This gives us a class of machines called **pushdown automata (pda).**

Figure 7.1



In this chapter, we explore the connection between pushdown automata and context-free languages. We first show that if we allow pushdown automata to act nondeterministically, we get a class of automata that accepts exactly the family of context-free languages. But we will also see that here there is no longer an equivalence between the deterministic and nondeterministic versions. The class of deterministic pushdown automata defines a new family of languages, the deterministic context-free languages, forming a proper subset of the context-free languages. Since this is an important family for the treatment of programming languages, we conclude the chapter with a brief introduction to the grammars associated with deterministic context-free languages.

7.1 Nondeterministic Pushdown Automata

A schematic representation of a pushdown automaton is given in Figure 7.1. Each move of the control unit reads a symbol from the input file, while at the same time changing the contents of the stack through the usual stack operations. Each move of the control unit is determined by the current input symbol as well as by the symbol currently on top of the stack. The result of the move is a new state of the control unit and a change in the top of the stack. In our discussion, we will restrict ourselves to pushdown automata acting as accepters.

Definition of a Pushdown Automaton

Formalizing this intuitive notion gives us a precise definition of a pushdown automaton.

Definition 7.1

A **nondeterministic pushdown accepter (npda)** is defined by the septuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F),$$

where

Q is a finite set of internal states of the control unit,
Σ is the input alphabet,
Γ is a finite set of symbols called the **stack alphabet,**
$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$ is the transition function,
$q_0 \in Q$ is the initial state of the control unit,
$z \in \Gamma$ is the **stack start symbol,**
$F \subseteq Q$ is the set of final states.

The complicated formal appearance of the domain and range of δ merits a closer examination. The arguments of δ are the current state of the control unit, the current input symbol, and the current symbol on top of the stack. The result is a set of pairs (q, x), where q is the next state of the control unit and x is a string which is put on top of the stack in place of the single symbol there before. Note that the second argument of δ may be λ, indicating that a move that does not consume an input symbol is possible. We will call such a move a λ-transition. Note also that δ is defined so that it needs a stack symbol; no move is possible if the stack is empty. Finally, the requirement that the range of δ be a finite subset is necessary because $Q \times \Gamma^*$ is an infinite set and therefore has infinite subsets. While an npda may have several choices for its moves, this choice must be restricted to a finite set of possibilities.

Example 7.1 Suppose the set of transition rules of an npda contains

$$\delta(q_1, a, b) = \{(q_2, cd), q_3, \lambda\}.$$

If at any time the control unit is in state q_1, the input symbol read is a, and the symbol on top of the stack is b, then one of two things can happen: (1) the control unit goes into state q_2 and the string cd replaces b on top of the stack, or (2) the control unit goes into state q_3 with the symbol b removed from the top of the stack. In our notation we assume that the insertion of a string into a stack is done symbol by symbol, starting at the right end of the string.

■

Example 7.2 Consider an npda with

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2, q_3\}, \\
\Sigma &= \{a, b\}, \\
\Gamma &= \{0, 1\}, \\
z &= 0, \\
F &= \{q_3\},
\end{aligned}
$$

and

$$
\begin{aligned}
\delta(q_0, a, 0) &= \{(q_1, 10), (q_3, \lambda)\}, \\
\delta(q_0, \lambda, 0) &= \{(q_3, \lambda)\}, \\
\delta(q_1, a, 1) &= \{(q_1, 11)\}, \\
\delta(q_1, b, 1) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, 1) &= \{(q_2, \lambda)\}, \\
\delta(q_2, \lambda, 0) &= \{(q_3, \lambda)\}.
\end{aligned}
$$

What can we say about the action of this automaton?

First, notice that transitions are not specified for all possible combinations of input and stack symbols. For instance, there is no entry given for $\delta(q_0, b, 0)$. The interpretation of this is the same that we used for nondeterministic finite automata: an unspecified transition is to the null set and represents a dead configuration for the npda.

The crucial transitions are

$$\delta(q_1, a, 1) = \{(q_1, 11)\},$$

which adds a 1 to the stack when an a is read, and

$$\delta(q_2, b, 1) = \{(q_2, \lambda)\},$$

which removes a 1 when a b is encountered. These two steps count the number of a's and match that count against the number of b's. The control unit is in state q_1 until the first b is encountered at which time it goes into state q_2. This assures that no b precedes the last a. After analyzing the remaining transitions, we see that the npda will end in the final state q_3 if and only if the input string is in the language

$$L = \{a^n b^n : n \geq 0\} \cup \{a\}.$$

As an analogy with finite automata, we might say that the npda accepts the above language. Of course, before making such a claim, we must define what we mean by an npda accepting a language. ■

To simplify the discussion, we introduce a convenient notation for describing the successive configurations of an npda during the processing of a string. The relevant factors at any time are the current state of the control unit, the unread part of the input string, and the current contents of the stack. Together these completely determine all the possible ways in which the npda can proceed. The triplet

where q is the state of the control unit, w is the unread part of the input string, and u is the stack contents (with the leftmost symbol indicating the top of the stack) is called an **instantaneous description** of a pushdown automaton. A move from one instantaneous description to another will be denoted by the symbol $\vdash$; thus

$$(q_1, aw, bx) \vdash (q_2, w, yx)$$

is possible if and only if

$$(q_2, y) \in \delta(q_1, a, b).$$

Moves involving an arbitrary number of steps will be denoted by $\overset{*}{\vdash}$. On occasions where several automata are under consideration we will use $\vdash_M$ to emphasize that the move is made by the particular automaton M.

The Language Accepted by a Pushdown Automaton

Definition 7.2

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ be a nondeterministic pushdown automaton. The language accepted by M is the set

$$L(M) = \left\{ w \in \Sigma^* : (q_0, w, z) \overset{*}{\vdash}_M (p, \lambda, u), p \in F, u \in \Gamma^* \right\}.$$

In words, the language accepted by M is the set of all strings that can put M into a final state at the end of the string. The final stack content u is irrelevant to this definition of acceptance.

Example 7.3 Construct an npda for the language

$$L = \{ w \in \{a, b\}^* : n_a(w) = n_b(w) \}.$$

As in Example 7.2, the solution to this problem involves counting the number of a's and b's, which is easily done with a stack. Here we need not even

worry about the order of the a's and b's. We can insert a counter symbol, say 0, into the stack whenever an a is read, then pop one counter symbol from the stack when a b is found. The only difficulty with this is that if there is a prefix of w with more b's than a's, we will not find a 0 to use. But this is easy to fix; we can use a negative counter symbol, say 1, for counting the b's that are to be matched against a's later. The complete solution is an npda $M = (\{q_0, q_f\}, \{a, b\}, \{0, 1, z\}, \delta, q_0, z, \{q_f\})$, with δ given as

$$\begin{aligned}
\delta(q_0, \lambda, z) &= \{(q_f, z)\}, \\
\delta(q_0, a, z) &= \{(q_0, 0z)\}, \\
\delta(q_0, b, z) &= \{(q_0, 1z)\}, \\
\delta(q_0, a, 0) &= \{(q_0, 00)\}, \\
\delta(q_0, b, 0) &= \{(q_0, \lambda)\}, \\
\delta(q_0, a, 1) &= \{(q_0, \lambda)\}, \\
\delta(q_0, b, 1) &= \{(q_0, 11)\}.
\end{aligned}$$

In processing the string $baab$, the npda makes the moves

$$\begin{aligned}
(q_0, baab, z) &\vdash (q_0, aab, 1z) \vdash (q_0, ab, z) \\
&\vdash (q_0, b, 0z) \vdash (q_0, \lambda, z) \vdash (q_f, \lambda, z)
\end{aligned}$$

and hence the string is accepted. ■

Example 7.4 To construct an npda for accepting the language

$$L = \left\{ ww^R : w \in \{a, b\}^+ \right\},$$

we use the fact that the symbols are retrieved from a stack in the reverse order of their insertion. When reading the first part of the string, we push consecutive symbols on the stack. For the second part, we compare the current input symbol with the top of the stack, continuing as long as the two match. Since symbols are retrieved from the stack in reverse of the order in which they were inserted, a complete match will be achieved if and only if the input is of the form ww^R.

An apparent difficulty with this suggestion is that we do not know the middle of the string, that is, where w ends and w^R starts. But the nondeterministic nature of the automaton helps us with this; the npda correctly guesses where the middle is and switches states at that point. A solution to the problem is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$, where

$$\begin{aligned}
Q &= \{q_0, q_1, q_2\}, \\
\Sigma &= \{a, b\}, \\
\Gamma &= \{a, b, z\}, \\
F &= \{q_2\}.
\end{aligned}$$

The transition function can be visualized as having several parts: a set to push w on the stack,

$$\left.\begin{aligned}
&\delta(q_0, a, a) = \{(q_0, aa)\}, \\
&\delta(q_0, b, a) = \{(q_0, ba)\}, \\
&\delta(q_0, a, b) = \{(q_0, ab)\}, \\
&\delta(q_0, b, b) = \{(q_0, bb)\}, \\
&\delta(q_0, a, z) = \{(q_0, az)\}, \\
&\delta(q_0, b, z) = \{(q_0, bz)\},
\end{aligned}\right\} \text{Push}$$

a set to guess the middle of the string, where the npda switches from state q_0 to q_1,

$$\begin{aligned}
&\delta(q_0, \lambda, a) = \{(q_1, a)\}, \\
&\delta(q_0, \lambda, b) = \{(q_1, b)\},
\end{aligned}$$

a set to match w^R against the contents of the stack,

$$\begin{aligned}
&\delta(q_1, a, a) = \{(q_1, \lambda)\}, \\
&\delta(q_1, b, b) = \{(q_1, \lambda)\},
\end{aligned}$$

and finally

$$\delta(q_1, \lambda, z) = \{(q_2, z)\},$$

to recognize a successful match.

The sequence of moves in accepting $abba$ is

$$\begin{aligned}
(q_0, abba, z) &\vdash (q_0, baa, az) \vdash (q_0, ba, baz) \\
&\vdash (q_1, ba, baz) \vdash (q_1, a, az) \vdash (q_1, \lambda, z) \vdash (q_2, z).
\end{aligned}$$

The nondeterministic alternative for locating the middle of the string is taken at the third move. At that stage, the pda has the instantaneous descriptions (q_0, ba, baz) and has two choices for its next move. One is to use $\delta(q_0, b, b) = \{(q_0, bb)\}$ and make the move

$$(q_0, baa, baz) \vdash (q_0, a, bbaz),$$

the second is the one used above, namely $\delta(q_0, \lambda, b) = \{(q_1, b)\}$. Only the latter leads to acceptance of the input.

EXERCISES

1. Find a pda with fewer than four states that accepts the same language as the pda in Example 7.2.
2. Prove that the pda in Example 7.4 does not accept any string not in $\{ww^R\}$.
3. Construct npda's that accept the following regular languages.
   (a) $L_1 = L(aaa^*b)$
   (b) $L_1 = L(aab^*aba^*)$
   (c) the union of L_1 and L_2
   (d) $L_1 - L_2$
4. Construct npda's that accept the following languages on $\Sigma = \{a, b, c\}$.
   (a) $L = \{a^n b^{2n} : n \geq 0\}$
   (b) $L = \{wcw^R : w \in \{a, b\}^*\}$
   (c) $L = \{a^n b^m c^{n+m} : n \geq 0, m \geq 0\}$
   (d) $L = \{a^n b^{n+m} c^m : n \geq 0, m \geq 1\}$
   (e) $L = \{a^3 b^n c^n : n \geq 0\}$
   (f) $L = \{a^n b^m : n \leq m \leq 3n\}$
   (g) $L = \{w : n_a(w) = n_b(w) + 1\}$
   (h) $L = \{w : n_a(w) = 2n_b(w)\}$
   (i) $L = \{w : n_a(w) + n_b(w) = n_c(w)\}$
   (j) $L = \{w : 2n_a(w) \leq n_b(w) \leq 3n_a(w)\}$
   (k) $L = \{w : n_a(w) < n_b(w)\}$
5. Construct an npda that accepts the language $L = \{a^n b^m : n \geq 0, n \neq m\}$.
6. Find an npda on $\Sigma = \{a, b, c\}$ that accepts the language

$$L = \left\{w_1 c w_2 : w_1, w_2 \in \{a, b\}^*, w_1 \neq w_2^R\right\}.$$

7. Find an npda for the concatenation of $L(a^*)$ and the language in Exercise 6.
8. Find an npda for the language $L = \{ab(ab)^n b(ba)^n : n \geq 0\}$.
9. Is it possible to find a dfa that accepts the same language as the pda

$$M = (\{q_0, q_1\}, \{a, b\}, \{z\}, q_0, \{q_1\}),$$

with

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_1, z)\},\\ \delta(q_0, b, z) &= \{(q_0, z)\},\\ \delta(q_1, a, z) &= \{(q_1, z)\},\\ \delta(q_1, b, z) &= \{(q_0, z)\}?\end{aligned}$$

10. What language is accepted by the pda

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \{0, 1, a\}, q_0, \{q_5\}),$$

with

$$\begin{aligned}\delta(q_0, b, z) &= \{(q_1, 1z)\},\\ \delta(q_1, b, 1) &= \{(q_1, 11)\},\\ \delta(q_2, a, 1) &= \{(q_3, \lambda)\},\\ \delta(q_3, a, 1) &= \{(q_4, \lambda)\},\\ \delta(q_4, a, z) &= \{(q_4, z), (q_5, z)\}?\end{aligned}$$

11. What language is accepted by the npda $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, z\}, \delta, q_0, z, \{q_2\})$ with transitions

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_1, a), (q_2, \lambda)\},\\ \delta(q_1, b, a) &= \{(q_1, b)\},\\ \delta(q_1, b, b) &= \{(q_1, b)\},\\ \delta(q_1, a, b) &= \{(q_2, \lambda)\}.\end{aligned}$$

12. What language is accepted by the npda in Example 7.3 if we use $F = \{q_0, q_f\}$?

13. What language is accepted by the npda in Exercise 11 above if we use $F = \{q_0, q_1, q_2\}$?

14. Find an npda with no more than two internal states that accepts the language $L(aa^*ba^*)$.

15. Suppose that in Example 7.2 we replace the given value of $\delta(q_2, \lambda, 0)$ with

$$\delta(q_2, \lambda, 0) = \{(q_0, \lambda)\}.$$

What is the language accepted by this new pda?

16. We can define a restricted npda as one that can increase the length of the stack by at most one symbol in each move, changing Definition 7.1 so that

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{Q \times (\Gamma\Gamma \cup \Gamma \cup \{\lambda\})}.$$

The interpretation of this is that the range of δ consists of sets of pairs of the form $(q_i, ab), (q_i, a)$, or (q_i, λ). Show that for every npda M there exists such a restricted npda $\widehat{M}$ such that $L(M) = L\left(\widehat{M}\right)$.

17. An alternative to Definition 7.2 for language acceptance is to require the stack to be empty when the end of the input string is reached. Formally, an npda M is said to accept the language $N(M)$ by empty stack if

$$N(M) = \left\{ w \in \Sigma^* : (q_0, w, z) \overset{*}{\vdash}_M (p, \lambda, \lambda) \right\},$$

where p is any element in Q. Show that this notion is effectively equivalent to Definition 7.2, in the sense that for any npda M there exists an npda $\widehat{M}$ such that $L(M) = N\left(\widehat{M}\right)$, and vice versa.

7.2 Pushdown Automata and Context-Free Languages

In the examples of the previous section, we saw that pushdown automata exist for some of the familiar context-free languages. This is no accident. There is a general relation between context-free languages and nondeterministic pushdown accepters that is established in the next two major results. We will show that for every context-free language there is an npda that accepts it, and conversely, that the language accepted by any npda is context-free.

Pushdown Automata for Context-Free Languages

We first show that for every context-free language there is an npda that accepts it. The underlying idea is to construct an npda that can, in some way, carry out a leftmost derivation of any string in the language. To simplify the argument a little, we assume that the language is generated by a grammar in Greibach normal form.

The pda we are about to construct will represent the derivation by keeping the variables in the right part of the sentential form on its stack, while the left part, consisting entirely of terminals, is identical with the input read. We begin by putting the start symbol on the stack. After that, to simulate the application of a production $A \to ax$, we must have the variable A on top of the stack and the terminal a as the input symbol. The variable on the stack is removed and replaced by the variable string x. What δ should be to achieve this is easy to see. Before we present the general argument, let us look at a simple example.

Example 7.5 Construct a pda that accepts the language generated by grammar with productions

$$S \to aSbb|a.$$

We first transform the grammar into Greibach normal form, changing the productions to

$$S \to aSA|a,$$
$$A \to bB,$$
$$B \to b.$$

The corresponding automaton will have three states $\{q_0, q_1, q_2\}$, with initial state q_0 and final state q_2. First, the start symbol S is put on the stack by

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}.$$

The production $S \to aSA$ will be simulated in the pda by removing S from the stack and replacing it with SA, while reading a from the input. Similarly, the rule $S \to a$ should cause the pda to read an a while simply removing S. Thus, the two productions are represented in the pda by

$$\delta(q_1, a, S) = \{(q_1, SA), (q_1, \lambda)\}.$$

In an analogous manner, the other productions give

$$\delta(q_1, b, A) = \{(q_1, B)\},$$
$$\delta(q_1, b, B) = \{(q_1, \lambda)\}.$$

The appearance of the stack start symbol on top of the stack signals the completion of the derivation and the pda is put into its final state by

$$\delta(q_1, \lambda, z) = \{(q_2, \lambda)\}.$$

The construction of this example can be adapted to other cases, leading to a general result.

■

Theorem 7.1 For any context-free language L, there exists an npda M such that

$$L = L(M).$$

Proof: If L is a λ-free context-free language, there exists a context-free grammar in Greibach normal form for it. Let $G = (V, T, S, P)$ be such a grammar. We then construct an npda which simulates leftmost derivations in this grammar. As suggested, the simulation will be done so that the unprocessed part of the sentential form is in the stack, while the terminal prefix of any sentential form matches the corresponding prefix of the input string.

Specifically, the npda will be

$$M = (\{q_0, q_1, q_f\}, T, V \cup \{z\}, \delta, q_0, z, \{q_f\}),$$

where $z \notin V$. Note that the input alphabet of M is identical with the set of terminals of G and that the stack alphabet contains the set of variables of the grammar.

The transition function will include

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}, \tag{7.1}$$

so that after the first move of M, the stack contains the start symbol S of the derivation. (The stack start symbol z is a marker to allow us to detect the end of the derivation.) In addition, the set of transition rules is such that

$$(q_1, u) \in \delta(q_1, a, A), \tag{7.2}$$

whenever

$$A \to au$$

is in P. This reads input a and removes the variable A from the stack, replacing it with u. In this way it generates the transitions that allow the pda to simulate all derivations. Finally, we have

$$\delta(q_1, \lambda, z) = \{(q_f, z)\}, \tag{7.3}$$

to get M into a final state.

To show that M accepts any $w \in L(G)$, consider the partial leftmost derivation

$$\begin{aligned} S &\overset{*}{\Rightarrow} a_1 a_2 \cdots a_n A_1 A_2 \cdots A_m \\ &\Rightarrow a_1 a_2 \cdots a_n b B_1 \cdots B_k A_2 \cdots A_m. \end{aligned}$$

If M is to simulate this derivation, then after reading $a_1 a_2 \cdots a_n$, the stack must contain $A_1 A_2 \cdots A_m$. To take the next step in the derivation, G must have a production

$$A_1 \to b B_1 \cdots B_k.$$

But the construction is such that then M has a transition rule in which

$$(q_1, B_1 \cdots B_k) \in \delta(q_1, b, A_1),$$

so that the stack now contains $B_1 \cdots B_k A_2 \cdots A_m$ after having read $a_1 a_2 \cdots a_n b$.

A simple induction argument on the number of steps in the derivation then shows that if

$$S \overset{*}{\Rightarrow} w,$$

then

$$(q_1, w, Sz) \overset{*}{\vdash} (q_1, \lambda, z).$$

Using (7.1) and (7.3) we have

$$(q_0, w, z) \vdash (q_1, w, Sz) \overset{*}{\vdash} (q_1, \lambda, z) \vdash (q_f, \lambda, z),$$

so that $L(G) \subseteq L(M)$.

To prove that $L(M) \subseteq L(G)$, let $w \in L(M)$. Then by definition

$$(q_0, w, z) \overset{*}{\vdash} (q_f, \lambda, u).$$

But there is only one way to get from q_0 to q_1 and only one way from q_1 to q_f. Therefore, we must have

$$(q_1, w, Sz) \overset{*}{\vdash} (q_1, \lambda, z).$$

Now let us write $w = a_1a_2a_3 \cdots a_n$. Then the first step in

$$(q_1, a_1a_2a_3 \cdots a_n, Sz) \overset{*}{\vdash} (q_1, \lambda, z) \tag{7.4}$$

must be a rule of the form (7.2) to get

$$(q_1, a_1a_2a_3 \cdots a_n, Sz) \vdash (q_1, a_2a_3 \cdots a_n, u_1z).$$

But then the grammar has a rule of the form $S \to a_1u_1$, so that

$$S \Rightarrow a_1u_1.$$

Repeating this, writing $u_1 = Au_2$, we have

$$(q_1, a_2a_3 \cdots a_n, Au_2z) \vdash (q_1, a_3 \cdots a_n, u_3u_2z),$$

implying that $A \to a_2u_3$ is in the grammar and that

$$S \overset{*}{\Rightarrow} a_1a_2u_3u_2.$$

This makes it quite clear at any point the stack contents (excluding z) are identical with the unmatched part of the sentential form, so that (7.4) implies

$$S \overset{*}{\Rightarrow} a_1a_2 \cdots a_n.$$

In consequence, $L(M) \subseteq L(G)$, completing the proof if the language does not contain λ.

If $\lambda \in L$, we add to the constructed npda the transition

$$\delta(q_0, \lambda, z) = \{(q_f, z)\}$$

so that the empty string is also accepted. ■

Example 7.6 Consider the grammar

$$S \to aA,$$
$$A \to aABC \,|bB|\, a,$$
$$B \to b,$$
$$C \to c.$$

Since the grammar is already in Greibach normal form, we can use the construction in the previous theorem immediately. In addition to rules

$$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$$

and

$$\delta(q_1, \lambda, z) = \{(q_f, z)\},$$

the pda will also have transition rules

$$\delta(q_1, a, S) = \{(q_1, A)\},$$
$$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\},$$
$$\delta(q_1, b, A) = \{(q_1, B)\},$$
$$\delta(q_1, b, B) = \{(q_1, \lambda)\},$$
$$\delta(q_1, c, C) = \{(q_1, \lambda)\}.$$

The sequence of moves made by M in processing $aaabc$ is

$$\begin{aligned}(q_0, aaabc, z) &\vdash (q_1, aaabc, Sz)\\ &\vdash (q_1, aabc, Az)\\ &\vdash (q_1, abc, ABCz)\\ &\vdash (q_1, bc, BCz)\\ &\vdash (q_1, c, Cz)\\ &\vdash (q_1, \lambda, z)\\ &\vdash (q_f, \lambda, z).\end{aligned}$$

This corresponds to the derivation

$$S \Rightarrow aA \Rightarrow aaABC \Rightarrow aaaBC \Rightarrow aaabC \Rightarrow aaabc.$$

■

In order to simplify the arguments, the proof in Theorem 7.1 assumed that the grammar was in Greibach normal form. It is not necessary to do this; we can make a similar and only slightly more complicated construction

from a general context-free grammar. For example, for productions of the form

$$A \to Bx,$$

we remove A from the stack and replace it with Bx, but consume no input symbol. For productions of the form

$$A \to abCx,$$

we must first match the ab in the input against a similar string in the stack and then replace A with Cx. We leave the details of the construction and the associated proof as an exercise.

Context-Free Grammars for Pushdown Automata

The converse of Theorem 7.1 is also true. The construction involved readily suggests itself: reverse the process in Theorem 7.1 so that the grammar simulates the moves of the pda. This means that the content of the stack should be reflected in the variable part of the sentential form, while the processed input is the terminal prefix of the sentential form. Quite a few details are needed to make this work.

To keep the discussion as simple as possible, we will assume that the npda in question meets the following requirements:

1. It has a single final state q_f that is entered if and only if the stack is empty;
2. All transitions must have the form $\delta(q_i, a, A) = \{c_1, c_2, ..., c_n\}$, where

$$c_i = (q_j, \lambda), \tag{7.5}$$

or

$$c_i = (q_j, BC). \tag{7.6}$$

That is, each move either increases or decreases the stack content by a single symbol.

These restrictions may appear to be very severe, but they are not. It can be shown that for any npda there exists an equivalent one having properties 1 and 2. This equivalence was explored partially in Exercises 16 and 17 in Section 7.1. Here we need to explore it further, but again we will leave the arguments as an exercise (see Exercise 16 at the end of this section). Taking this as given, we now construct a context-free grammar for the language accepted by the npda.

As stated, we want the sentential form to represent the content of the stack. But the configuration of the npda also involves an internal state, and

this has to be remembered in the sentential form as well. It is hard to see how this can be done, and the construction we give here is a little tricky.

Suppose for the moment that we can find a grammar whose variables are of the form $(q_i A q_j)$ and whose productions are such that

$$(q_i A q_j) \stackrel{*}{\Rightarrow} v,$$

if and only if the npda erases A from the stack while reading v and going from state q_i to state q_j. "Erasing" here means that A and its effects (i.e., all the successive strings by which it is replaced) are removed from the stack, bringing the symbol originally below A to the top. If we can find such a grammar, and if we choose $(q_0 z q_f)$ as its start symbol, then

$$(q_0 z q_f) \stackrel{*}{\Rightarrow} w,$$

if and only if the npda removes z (creating an empty stack) while reading w and going from q_0 to q_f. But this is exactly how the npda accepts w. Therefore, the language generated by the grammar will be identical to the language accepted by the npda.

To construct a grammar that satisfies these conditions, we examine the different types of transitions that can be made by the npda. Since (7.5) involves an immediate erasure of A, the grammar will have a corresponding production

$$(q_i A q_j) \rightarrow a.$$

Productions of type (7.6) generate the set of rules

$$(q_i A q_k) \rightarrow a\,(q_j B q_l)\,(q_l C q_k)\,,$$

where q_k and q_l take on all possible values in Q. This is due to the fact that to erase A we first replace it with BC, while reading an a and going from state q_i to q_j. Subsequently, we go from q_j to q_l, erasing B, then from q_l to q_k, erasing C.

In the last step, it may seem that we have added too much, as there may be some states q_l that cannot be reached from q_j while erasing B. This is true, but this does not affect the grammar. The resulting variables $(q_j B q_l)$ are useless variables and do not affect the language accepted by the grammar.

Finally, as start variable we take $(q_0 z q_f)$, where q_f is the single final state of the npda.

Example 7.7 Consider the npda with transitions

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_0, Az)\},\\ \delta(q_0, a, A) &= \{(q_0, A)\},\\ \delta(q_0, b, A) &= \{(q_1, \lambda)\},\\ \delta(q_1, \lambda, z) &= \{(q_2, \lambda)\}.\end{aligned}$$

Using q_0 as initial state and q_2 as the final state, the npda satisfies condition 1 above, but not 2. To satisfy the latter, we introduce a new state q_3 and an intermediate step in which we first remove the A from the stack, then replace it in the next move. The new set of transition rule is

$$\begin{aligned}\delta(q_0, a, z) &= \{(q_0, Az)\},\\ \delta(q_3, \lambda, z) &= \{(q_0, Az)\},\\ \delta(q_0, a, A) &= \{(q_3, \lambda)\},\\ \delta(q_0, b, A) &= \{(q_1, \lambda)\},\\ \delta(q_1, \lambda, z) &= \{(q_2, \lambda)\}.\end{aligned}$$

The last three transitions are of the form (7.5) so that they yield the corresponding productions

$$(q_0Aq_3) \to a, \quad (q_0Aq_1) \to b, \quad (q_1zq_2) \to \lambda.$$

From the first two transitions we get the set of productions

$$\begin{aligned}(q_0zq_0) &\to a\,(q_0Aq_0)\,(q_0zq_0)\,|a\,(q_0Aq_1)\,(q_1zq_0)|\\ &\quad\ a\,(q_0Aq_2)\,(q_2zq_0)\,|a\,(q_0Aq_3)\,(q_3zq_0),\\ (q_0zq_1) &\to a\,(q_0Aq_0)\,(q_0zq_1)\,|a\,(q_0Aq_1)\,(q_1zq_1)|\\ &\quad\ a\,(q_0Aq_2)\,(q_2zq_1)\,|a\,(q_0Aq_3)\,(q_3zq_1),\\ (q_0zq_2) &\to a\,(q_0Aq_0)\,(q_0zq_2)\,|a\,(q_0Aq_1)\,(q_1zq_2)|\\ &\quad\ a\,(q_0Aq_2)\,(q_2zq_2)\,|a\,(q_0Aq_3)\,(q_3zq_2),\\ (q_0zq_3) &\to a\,(q_0Aq_0)\,(q_0zq_3)\,|a\,(q_0Aq_1)\,(q_1zq_3)|\\ &\quad\ a\,(q_0Aq_2)\,(q_2zq_3)\,|a\,(q_0Aq_3)\,(q_3zq_3),\end{aligned}$$

$$\begin{aligned}&(q_3zq_0) \to (q_0Aq_0)\,(q_0zq_0)\,|(q_0Aq_1)\,(q_1zq_0)|\,(q_0Aq_2)\,(q_2zq_0)\,|\,(q_0Aq_3)\,(q_3zq_0),\\ &(q_3zq_1) \to (q_0Aq_0)\,(q_0zq_1)\,|(q_0Aq_1)\,(q_1zq_1)|\,(q_0Aq_2)\,(q_2zq_1)\,|\,(q_0Aq_3)\,(q_3zq_1),\\ &(q_3zq_2) \to (q_0Aq_0)\,(q_0zq_2)\,|(q_0Aq_1)\,(q_1zq_2)|\,(q_0Aq_2)\,(q_2zq_2)\,|\,(q_0Aq_3)\,(q_3zq_2),\\ &(q_3zq_3) \to (q_0Aq_0)\,(q_0zq_3)\,|(q_0Aq_1)\,(q_1zq_3)|\,(q_0Aq_2)\,(q_2zq_3)\,|\,(q_0Aq_3)\,(q_3zq_3).\end{aligned}$$

The start variable will be $(q_0 z q_2)$. The string aab is accepted by the pda, with successive configurations

$$\begin{aligned}(q_0, aab, z) &\vdash (q_0, ab, Az) \\ &\vdash (q_3, b, z) \\ &\vdash (q_0, b, Az) \\ &\vdash (q_1, \lambda, z) \\ &\vdash (q_2, \lambda, \lambda)\,.\end{aligned}$$

The corresponding derivation with G is

$$\begin{aligned}(q_0 z q_2) &\Rightarrow a\,(q_0 A q_3)\,(q_3 z q_2) \\ &\Rightarrow aa\,(q_3 z q_2) \\ &\Rightarrow aa\,(q_0 A q_1)\,(q_1 z q_2) \\ &\Rightarrow aab\,(q_1 z q_2) \\ &\Rightarrow aab.\end{aligned}$$

The steps in the proof of the following theorem will be easier to understand if you notice the correspondence between the successive instantaneous descriptions of the pda and the sentential forms in the derivation. The first q_i in the leftmost variable of every sentential form is the current state of the pda, while the sequence of middle symbols is the same as the stack content.

Although the construction yields a rather complicated grammar, it can be applied to any pda whose transition rules satisfy the given conditions. This forms the basis for the proof of the general result. ■

Theorem 7.2 If $L = L(M)$ for some npda M, then L is a context-free language.

Proof: Assume that $M = (Q, \Sigma, \Gamma, \delta, q_0, z, \{q_f\})$ satisfies conditions 1 and 2 above. We use the suggested construction to get the grammar $G = (V, T, S, P)$, with $T = \Sigma$ and V consisting of elements of the form $(q_i c q_j)$. We will show that the grammar so obtained is such that for all $q_i, q_j, \in Q, A \in \Gamma, X \in \Gamma^*, u, v \in \Sigma^*$,

$$(q_i, uv, AX) \overset{*}{\vdash} (q_j, v, X) \tag{7.7}$$

implies that

$$(q_i A q_j) \overset{*}{\Rightarrow} u,$$

and vice versa.

The first part is to show that, whenever the npda is such that the symbol A and its effects can be removed from the stack while reading u and

going from state q_i to q_j, then the variable (q_iAq_j) can derive u. This is not hard to see since the grammar was explicitly constructed to do this. We only need an induction on the number of moves to make this precise.

For the converse, consider a single step in the derivation such as

$$(q_iAq_k) \Rightarrow a\,(q_jBq_l)\,(q_lCq_k)\,.$$

Using the corresponding transition for the npda

$$\delta\,(q_i, a, A) = \{(q_j, BC)\,, ...\}\,, \tag{7.8}$$

we see that the A can be removed from the stack, BC put on, reading a, with the control unit going from state q_i to q_j. Similarly, if

$$(q_iAq_j) \Rightarrow a, \tag{7.9}$$

then there must be a corresponding transition

$$\delta\,(q_i, a, A) = \{(q_j, \lambda)\} \tag{7.10}$$

whereby the A can be popped off the stack. We see from this that the sentential forms derived from (q_iAq_j) define a sequence of possible configurations of the npda by which (7.7) can be achieved.

Notice that $(q_iAq_j) \Rightarrow a\,(q_jBq_l)\,(q_lCq_k)$ might be possible for some $(q_jBq_l)\,(q_lCq_k)$ for which there is no corresponding transition of the form (7.8) or (7.10). But, in that case, at least one of the variables on the right will be useless. For all sentential forms leading to a terminal string, the argument given holds.

If we now apply the conclusion to

$$(q_0, w, z) \overset{*}{\vdash} (q_f, \lambda, \lambda)\,,$$

we see that this can be so if and only if

$$(q_0zq_f) \overset{*}{\Rightarrow} w.$$

Consequently $L\,(M) = L\,(G)$. ■

EXERCISES

1. Show that the pda constructed in Example 7.5 accepts the string *aaabbbb* that is in the language generated by the given grammar.
2. Prove that the pda in Example 7.5 accepts the language $L = \{a^{n+1}b^{2n} : n \geq 0\}$.

3. Construct an npda that accepts the language generated by the grammar

$$S \to aSbb|aab.$$

4. Construct an npda that accepts the language generated by the grammar $S \to aSSS|ab$.

5. Construct an npda corresponding to the grammar

$$S \to aABB|aAA,$$
$$A \to aBB|a,$$
$$B \to bBB|A.$$

6. Construct an npda that will accept the language generated by the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions $S \to AA|a, A \to SA|b$.

7. Show that Theorems 7.1 and 7.2 imply the following. For every npda M, there exists an npda $\widehat{M}$ with at most three states, such that $L(M) = L\left(\widehat{M}\right)$.

8. Show how the number of states of $\widehat{M}$ in the above exercise can be reduced to two.

9. Find an npda with two states for the language $L = \left\{a^n b^{n+1} : n \geq 0\right\}$.

10. Find an npda with two states that accepts $L = \left\{a^n b^{2n} : n \geq 1\right\}$.

11. Show that the npda in Example 7.7 accepts $L(aa^*b)$.

12. Show that the grammar in Example 7.7 generates the language $L(aa^*b)$.

13. In Example 7.7, show that the variables (q_0Bq_0) and (q_0zq_1) are useless.

14. Use the construction in Theorem 7.1 to find an npda for the language Example 7.5, Section 7.1.

15. Find a context-free grammar that generates the language accepted by the npda $M = (\{q_0, q_1\}, \{a, b\}, \{A, z\}, \delta, q_0, z, \{q_1\})$, with transitions

$$\delta(q_0, a, z) = \{(q_0, Az)\},$$
$$\delta(q_0, b, A) = \{(q_0, AA)\},$$
$$\delta(q_0, a, A) = \{(q_1, \lambda)\}.$$

16. Show that for every npda there exists an equivalent one satisfying conditions 1 and 2 in the preamble to Theorem 7.2.

17. Give full details of the proof of Theorem 7.2.

18. Give a construction by which an arbitrary context-free grammar can be used in the proof of Theorem 7.1.

7.3 Deterministic Pushdown Automata and Deterministic Context-Free Languages

A **deterministic pushdown accepter** (**dpda**) is a pushdown automaton that never has a choice in its move. This can be achieved by a modification of Definition 7.1.

Definition 7.3

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ is said to be deterministic if it is an automaton as defined in Definition 7.1, subject to the restrictions that, for every $q \in Q, a \in \Sigma \cup \{\lambda\}$ and $b \in \Gamma$,

1. $\delta(q, a, b)$ contains at most one element,
2. if $\delta(q, \lambda, b)$ is not empty, then $\delta(q, c, b)$ must be empty for every $c \in \Sigma$.

The first of these conditions simply requires that for any given input symbol and any stack top, at most one move can be made. The second condition is that when a λ-move is possible for some configuration, no input-consuming alternative is available.

It is interesting to note the difference between this definition and the corresponding definition of a deterministic finite automaton. The domain of the transition function is still as in Definition 7.1 rather than $Q \times \Sigma \times \Gamma$ because we want to retain λ-transitions. Since the top of the stack plays a role in determining the next move, the presence of λ-transitions does not automatically imply nondeterminism. Also, some transitions of a dpda may be to the empty set, that is, undefined, so there may be dead configurations. This does not affect the definition; the only criterion for determinism is that at all times at most one possible move exists.

Definition 7.4

A language L is said to be a **deterministic context-free language** if and only if there exists a dpda M such that $L = L(M)$.

Example 7.8 The language

$$L = \{a^n b^n : n \geq 0\}$$

is a deterministic context-free language. The pda $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{0, 1\}, \delta, q_0, 0, \{q_0\})$ with

$$\begin{aligned}
\delta(q_0, a, 0) &= \{(q_1, 10)\}, \\
\delta(q_1, a, 1) &= \{(q_1, 11)\}, \\
\delta(q_1, b, 1) &= \{(q_2, \lambda)\}, \\
\delta(q_2, b, 1) &= \{(q_2, \lambda)\}, \\
\delta(q_2, \lambda, 0) &= \{(q_0, \lambda)\},
\end{aligned}$$

accepts the given language. It satisfies the conditions of Definition 7.4 and is therefore deterministic. ■

Look now at Example 7.4. The npda there is not deterministic because

$$\delta(q_0, a, a) = \{(q_0, aa)\}$$

and

$$\delta(q_0, \lambda, a) = \{(q_1, a)\}$$

violate condition 2 of Definition 7.3. This, of course, does not imply that the language $\{ww^R\}$ itself is nondeterministic, since there is the possibility of an equivalent dpda. But it is known that the language is indeed not deterministic. From this and the next example we see that, in contrast to finite automata, deterministic and nondeterministic pushdown automata are not equivalent. There are context-free languages that are not deterministic.

Example 7.9 Let

$$L_1 = \{a^n b^n : n \geq 0\}$$

and

$$L_2 = \{a^n b^{2n} : n \geq 0\}.$$

An obvious modification of the argument that L_1 is a context-free language shows that L_2 is also context-free. The language

$$L = L_1 \cup L_2$$

is context-free as well. This will follow from a general theorem to be presented in the next chapter, but can easily be made plausible at this point. Let $G_1 = (V_1, T, S_1, P_1)$ and $G_2 = (V_2, T, S_2, P_2)$ be context-free grammars such that $L_1 = L(G_1)$ and $L_2 = L(G_2)$. If we assume that V_1 and V_2 are disjoint and that $S \notin V_1 \cup V_2$, then, combining the two, grammar $G = (V_1 \cup V_2 \cup \{S\}, T, S, P)$, where

$$P = P_1 \cup P_2 \cup \{S \to S_1 | S_2\},$$

generates $L_1 \cup L_2$. This should be fairly clear at this point, but the details of the argument will be deferred until Chapter 8. Accepting this, we see that L is context-free. But L is not a deterministic context-free language. This seems reasonable, since the pda has either to match one b or two against each a, and so has to make an initial choice whether the input is in L_1 or in L_2. There is no information available at the beginning of the string by which the choice can be made deterministically. Of course, this sort of argument is based on a particular algorithm we have in mind; it may lead us to the correct conjecture, but does not prove anything. There is always the possibility of a completely different approach that avoids an initial choice. But it turns out that there is not, and L is indeed nondeterministic. To see this we first establish the following claim. If L were a deterministic context-free language, then

$$\widehat{L} = L \cup \{a^n b^n c^n : n \geq 0\}$$

would be a context-free language. We show the latter by constructing an npda $\widehat{M}$ for $\widehat{L}$, given a dpda M for L.

The idea behind the construction is to add to the control unit of M a similar part in which transitions caused by the input symbol b are replaced with similar ones for input c. This new part of the control unit may be entered after M has read $a^n b^n$. Since the second part responds to c^n in the same way as the first part does to b^n, the process that recognizes $a^n b^{2n}$ now also accepts $a^n b^n c^n$. Figure 7.2 describes the construction graphically; a formal argument follows.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ with

$$Q = \{q_0, q_1, ..., q_n\}.$$

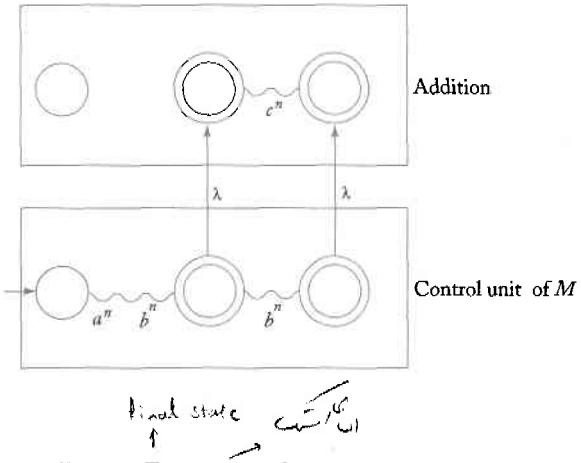
Then consider $\widehat{M} = \left(\widehat{Q}, \Sigma, \Gamma, \delta \cup \widehat{\delta}, z, \widehat{F}\right)$ with

$$\widehat{Q} = Q \cup \{\widehat{q}_0, \widehat{q}_1, ..., \widehat{q}_n\},$$
$$\widehat{F} = F \cup \{\widehat{q}_i : q_i \in F\},$$

and $\widehat{\delta}$ constructed from δ by including

$$\widehat{\delta}(q_f, \lambda, s) = \{(\widehat{q}_f, s)\},$$

Figure 7.2



for all $q_f \in F, s \in \Gamma$, and

$$\widehat{\delta}\left(\widehat{q}_i, c, s\right) = \left\{\left(\widehat{q}_j, u\right)\right\},$$

for all

$$\delta\left(q_i, b, s\right) = \left\{\left(q_j, u\right)\right\},$$

$q_i \in Q, s \in \Gamma, u \in \Gamma^*$. For M to accept $a^n b^n$ we must have

$$\left(q_0, a^n b^n, z\right) \overset{*}{\vdash}_M \left(q_i, \lambda, u\right),$$

with $q_i \in F$. Because M is deterministic, it must also be true that

$$\left(q_0, a^n b^{2n}, z\right) \overset{*}{\vdash}_M \left(q_i, b^n, u\right),$$

so that for it to accept $a^n b^{2n}$ we must further have

$$\left(q_i, b^n, u\right) \overset{*}{\vdash}_M \left(q_j, \lambda, u_1\right),$$

for some $q_j \in F$. But then, by construction

$$\left(\widehat{q}_i, c^n, u\right) \overset{*}{\vdash}_{\widehat{M}} \left(\widehat{q}_j, \lambda, u_1\right),$$

so that $\widehat{M}$ will accept $a^n b^n c^n$. It remains to be shown that no strings other than those in $\widehat{L}$ are accepted by $\widehat{M}$; this is considered in several exercises at the end of this section. The conclusion is that $\widehat{L} = L\left(\widehat{M}\right)$, so that $\widehat{L}$ is context-free. But we will show in the next chapter (Example 8.1) that $\widehat{L}$ is not context-free. Therefore, our assumption that L is a deterministic context-free language must be false.

■

EXERCISES

1. Show that $L = \{a^n b^{2n} : n \geq 0\}$ is a deterministic context-free language.
2. Show that $L = \{a^n b^m : m \geq n + 2\}$ is deterministic.
3. Is the language $L = \{a^n b^n : n \geq 1\} \cup \{b\}$ deterministic?
4. Is the language $L = \{a^n b^n : n \geq 1\} \cup \{a\}$ in Example 7.2 deterministic?
5. Show that the pushdown automaton in Example 7.3 is not deterministic, but that the language in the example is nevertheless deterministic.
6. For the language L in Exercise 1, show that L^* is a deterministic context-free language.
7. Give reasons why one might conjecture that the following language is not deterministic.

$$L = \{a^n b^m c^k : n = m \text{ or } m = k\}$$

8. Is the language $L = \{a^n b^m : n = m \text{ or } n = m + 2\}$ deterministic?
9. Is the language $\{wcw^R : w \in \{a, b\}^*\}$ deterministic?
10. While the language in Exercise 9 is deterministic, the closely related language $L = \{ww^R : w \in \{a, b\}^*\}$ is known to be nondeterministic. Give arguments that make this statement plausible.
11. Show that $L = \{w \in \{a, b\}^* : n_a(w) \neq n_b(w)\}$ is a deterministic context-free language.
12. Show that $\widehat{M}$ in Example 7.9 does not accept $a^n b^n c^k$ for $k \neq n$.
13. Show that $\widehat{M}$ in Example 7.9 does not accept any string not in $L(a^* b^* c^*)$.
14. Show that $\widehat{M}$ in Example 7.9 does not accept $a^n b^{2n} c^k$ with $k > 0$. Show also that it does not accept $a^n b^m c^k$ unless $m = n$ or $m = 2n$.
15. Show that every regular language is a deterministic context-free language.
16. Show that if L_1 is deterministic context-free and L_2 is regular, then the language $L_1 \cup L_2$ is deterministic context-free.
17. Show that under the conditions of Exercise 16, $L_1 \cap L_2$ is a deterministic context-free language.
18. Give an example of a deterministic context-free language whose reverse is not deterministic.

7.4 Grammars for Deterministic Context-Free Languages*

The importance of deterministic context-free languages lies in the fact that they can be parsed efficiently. We can see this intuitively by viewing the pushdown automaton as a parsing device. Since there is no backtracking involved, we can easily write a computer program for it, and we may expect that it will work efficiently. Since there may be λ-transitions involved, we cannot immediately claim that this will yield a linear-time parser, but it puts us on the right track nevertheless. To pursue this, let us see what grammars might be suitable for the description of deterministic context-free languages. Here we enter a topic important in the study of compilers, but somewhat peripheral to our interests. We will provide only a brief introduction to some important results, referring the reader to books on compilers for a more thorough treatment.

Suppose we are parsing top-down, attempting to find the leftmost derivation of a particular sentence. For the sake of discussion, we use the approach illustrated in Figure 7.3. We scan the input w from left to right, while developing a sentential form whose terminal prefix matches the prefix of w up to the currently scanned symbol. To proceed in matching consecutive symbols, we would like to know exactly which production rule is to be applied at each step. This would avoid backtracking and give us an efficient parser. The question then is whether there are grammars that allow us to do this. For a general context-free grammar, this is not the case, but if the form of the grammar is restricted, we can achieve our goal.

As first case, take the s-grammars introduced in Definition 5.4. From the discussion there, it is clear that at every stage in the parsing we know exactly which production has to be applied. Suppose that $w = w_1 w_2$ and that we have developed the sentential form $w_1 Ax$. To get the next symbol of the sentential form matched against the next symbol in w, we simply look at the leftmost symbol of w_2, say a. If there is no rule $A \rightarrow ay$ in the grammar, the string w does not belong to the language. If there is such a rule, the parsing can proceed. But in this case there is only one such rule, so there is no choice to be made.

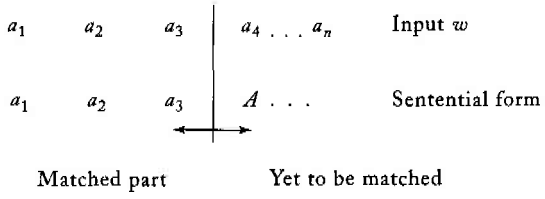


Figure 7.3

Although s-grammars are useful, they are too restrictive to capture all aspects of the syntax of programming languages. We need to generalize the idea so that it becomes more powerful without losing its essential property for parsing. One type of grammar is called an **LL grammar.** In an LL grammar we still have the property that we can, by looking at a limited part of the input (consisting of the scanned symbol plus a finite number of symbols following it), predict exactly which production rule must be used. The term LL is standard usage in books on compilers; the first L stands for the fact that the input is scanned from left to right; the second L indicates that leftmost derivations are constructed. Every s-grammar is an LL grammar, but the concept is more general.

Example 7.10 The grammar

$$S \rightarrow aSb|ab$$

is not an s-grammar, but it is an LL grammar. In order to determine which production is to be applied, we look at two consecutive symbols of the input string. If the first is an a and the second a b, we must apply the production $S \rightarrow ab$. Otherwise, the rule $S \rightarrow aSb$ must be used. ■

We say that a grammar is an $LL(k)$ grammar if we can uniquely identify the correct production, given the currently scanned symbol and a "look-ahead" of the next $k-1$ symbols. The above is an example of an $LL(2)$ grammar.

Example 7.11 The grammar

$$S \rightarrow SS\,|aSb|\,ab$$

generates the positive closure of the language in Example 7.10. As remarked in Example 5.4, this is the language of properly nested parenthesis structures. The grammar is not an $LL(k)$ grammar for any k.

To see why this is so, look at the derivation of strings of length greater than two. To start, we have available two possible productions $S \rightarrow SS$ and $S \rightarrow aSb$. The scanned symbol does not tell us which is the right one. Suppose we now use a look-ahead and consider the first two symbols, finding that they are aa. Does this allow us to make the right decision? The answer is still no, since what we have seen could be a prefix of a number of strings, including both $aabb$ or $aabbab$. In the first case, we must start with $S \rightarrow aSb$, while in the second it is necessary to use $S \rightarrow SS$. The grammar is therefore not an $LL(2)$ grammar. In a similar fashion, we can see that

no matter how many look-ahead symbols we have, there are always some situations that cannot be resolved.

This observation about the grammar does not imply that the language is not deterministic or that no LL grammar for it exists. We can find an LL grammar for the language if we analyze the reason for the failure of the original grammar. The difficulty lies in the fact that we cannot predict how many repetitions of the basic pattern $a^n b^n$ there are until we get to the end of the string, yet the grammar requires an immediate decision. Rewriting the grammar avoids this difficulty. The grammar

$$S \rightarrow aSbS|\lambda$$

is an LL-grammar nearly equivalent to the original grammar.

To see this, consider the leftmost derivation of $w = abab$. Then

$$S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab.$$

We see that we never have any choice. When the input symbol examined is a, we must use $S \rightarrow aSbS$, when the symbol is b or if we are at the end of the string, we must use $S \rightarrow \lambda$.

But the problem is not yet completely solved because the new grammar can generate the empty string. We fix this by introducing a new start variable S_0 and a production to ensure that some nonempty string is generated. The final result

$$S_0 \rightarrow aSbS$$
$$S \rightarrow aSbS|\lambda$$

is then an LL-grammar equivalent to the original grammar. ■

While this informal description of LL grammars is adequate for understanding simple examples, we need a more precise definition if any rigorous results are to be developed. We conclude our discussion with such a definition.

Definition 7.5

Let $G = (V, T, S, P)$ be a context-free grammar. If for every pair of left-most derivations

$$S \overset{*}{\Rightarrow} w_1 A x_1 \Rightarrow w_1 y_1 x_1 \overset{*}{\Rightarrow} w_1 w_2,$$
$$S \overset{*}{\Rightarrow} w_1 A x_2 \Rightarrow w_1 y_2 x_2 \overset{*}{\Rightarrow} w_1 w_3,$$

with $w_1, w_2, w_3 \in T^*$, the equality of the k leftmost symbols of w_2 and w_3 implies $y_1 = y_2$, then G is said to be an $LL(k)$ grammar. (If $|w_2|$ or $|w_3|$ is less than k, then k is replaced by the smaller of these.)

The definition makes precise what has already been indicated. If at any stage in the leftmost derivation $(w_1 Ax)$ we know the next k symbols of the input, the next step in the derivation is uniquely determined (as expressed by $y_1 = y_2$).

The topic of LL grammars is an important one in the study of compilers. A number of programming languages can be defined by LL grammars, and many compilers have been written using LL parsers. But LL grammars are not sufficiently general to deal with all deterministic context-free languages. Consequently, there is interest in other, more general deterministic grammars. Particularly important are the so-called LR grammars, which also allow efficient parsing, but can be viewed as constructing the derivation tree from the bottom up. There is a great deal of material on this subject that can be found in books on compilers (e.g., Hunter 1981) or books specifically devoted to parsing methods for formal languages (such as Aho and Ullman 1972).

EXERCISES

1. Show that the second grammar in Example 7.11 is an LL grammar and that it is equivalent to the original grammar.
2. Show that the grammar for $L = \{w : n_a(w) = n_b(w)\}$ given in Example 1.13 is not an LL grammar.
3. Find an LL grammar for the language in Exercise 2.
4. Construct an LL grammar for the language $L(a^*ba) \cup L(abbb^*)$.
5. Show that any LL grammar is unambiguous.
6. Show that if G is an $LL(k)$ grammar, then $L(G)$ is a deterministic context-free language.
7. Show that a deterministic context-free language is never inherently ambiguous.
8. Let G be a context-free grammar in Greibach normal form. Describe an algorithm which, for any given k, determines whether or not G is an $LL(k)$ grammar.
9. Give LL grammars for the following languages, assuming $\Sigma = \{a, b, c\}$.
   (a) $L = \{a^n b^m c^{n+m} : n \geq 0, m \geq 0\}$
   (b) $L = \{a^{n+2} b^m c^{n+m} : n \geq 0, m \geq 0\}$
   (c) $L = \{a^n b^{n+2} c^m : n \geq 0, m > 1\}$
   (d) $L = \{w : n_a(w) < n_b(w)\}$
   (e) $L = \{w : n_a(w) + n_b(w) \neq n_c(w)\}$