a) Design an algorithm to construct an NFA that accepts the language of a left-regular grammar.

b) Show that the left-regular grammars generate precisely the regular sets.

22. A context-free grammar $G = (V, \Sigma, P, S)$ is called **left-linear** if each rule is of the form

i) $A \rightarrow u$, or

ii) $A \rightarrow Bu$,

where $A, B \in V$, and $u \in \Sigma^*$. Show that the left-linear grammars generate precisely the regular sets.

23. Give a regular language L such that $\equiv_L$ has exactly three equivalence classes.

24. Give the $\equiv_L$ equivalence classes of the language a^+b^+.

25. Let $[u]_{\equiv_L}$ be a $\equiv_L$ equivalence class of a language L. Show that if $[u]_{\equiv_L}$ contains one string $v \in L$, then every string in $[u]_{\equiv_L}$ is in L.

26. Prove that $\equiv_L$ is right-invariant for any regular language L. That is, if $u \equiv_L v$, then $ux \equiv_L vx$ for any $x \in \Sigma^*$, where Σ is the alphabet of the language L.

27. Use the Myhill-Nerode Theorem to prove that the language $\{a^i \mid i \text{ is a perfect square}\}$ is not regular.

28. Let $u \in [ab]_{\equiv_M}$ and $v \in [aba]_{\equiv_M}$ be strings from the equivalence classes of $(a \cup b)(a \cup b^*)$ defined in Example 6.7.4. Show that u and v are distinguishable.

29. Give the equivalence classes defined by the relation $\equiv_M$ for the DFA in Example 5.3.1.

30. Give the equivalence classes defined by the relation $\equiv_M$ for the DFA in Example 5.3.3.

***31.** Let M_L be the minimal state DFA that accepts a language L defined in Theorems 6.7.4 and 6.7.5. Let M be another DFA that accepts L with the same number of states as M_L. Prove that M_L and M are identical except (possibly) for the names assigned to the states. Two such DFAs are said to be *isomorphic*.

Bibliographic Notes

The equivalence of regular sets and languages accepted by finite automata was established by Kleene [1956]. The proof given in Section 6.2 is modeled after that of McNaughton and Yamada [1960]. Chomsky and Miller [1958] established the equivalence of the languages generated by regular grammars and accepted by finite automata. Closure under homomorphisms (Exercise 19) is from Ginsburg and Rose [1963b]. The closure of regular sets under reversal was noted by Rabin and Scott [1959]. Additional closure results for regular sets can be found in Bar-Hillel, Perles, and Shamir [1961], Ginsburg and Rose [1963b], and Ginsburg [1966]. The pumping lemma for regular languages is from Bar-Hillel, Perles, and Shamir [1961]. The relationship between the number of equivalence classes of a language and regularity was established in Myhill [1957] and Nerode [1958].

CHAPTER 7

Pushdown Automata and Context-Free Languages

Regular languages have been characterized as the languages generated by regular grammars and accepted by finite automata. This chapter presents a class of machines, the pushdown automata, that accepts the context-free languages. A pushdown automaton is a finite-state machine augmented with an external stack memory. The addition of a stack provides the pushdown automaton with a last-in, first-out memory management capability. The combination of stack and states overcomes the memory limitations that prevented the acceptance of the language $\{a^ib^i \mid i \geq 0\}$ by a deterministic finite automaton.

As with regular languages, a pumping lemma for context-free languages ensures the existence of repeatable substrings in strings of a context-free language. The pumping lemma provides a technique for showing that many easily definable languages are not context-free.

7.1 Pushdown Automata

Theorem 6.5.1 established that the language $\{a^ib^i \mid i \geq 0\}$ is not accepted by any finite automaton. To accept this language, a machine needs the ability to record the processing of any finite number of a's. The restriction of having finitely many states does not allow the automaton to "remember" the number of leading a's in an arbitrary input string. A new type of automaton is constructed that augments the state-input transitions of a finite automaton with the ability to utilize unlimited memory.

A pushdown stack, or simply a stack, is added to a finite automaton to construct a new machine known as a pushdown automaton (PDA). Stack operations affect only the top item of the stack; a pop removes the top element from the stack and a push places an element

on the stack top. Definition 7.1.1 formalizes the concept of a pushdown automaton. The components Q, Σ, q_0, and F of a PDA are the same as in a finite automaton.

Definition 7.1.1

A **pushdown automaton** is a sextuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q is a finite set of states, Σ a finite set called the *input alphabet*, Γ a finite set called the *stack alphabet*, q_0 the start state, $F \subseteq Q$ a set of final states, and δ a transition function from $Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ to subsets of $Q \times (\Gamma \cup \{\lambda\})$.

A PDA has two alphabets: an input alphabet Σ from which the input strings are built and a stack alphabet Γ whose elements are stored on the stack. The stack is represented as a string of stack elements; the element on the top of the stack is the leftmost symbol in the string. We will use capital letters to represent stack elements and Greek letters to represent strings of stack elements. The notation $A\alpha$ represents a stack with A as the top element. An empty stack is denoted λ. The computation of a PDA begins with the machine in state q_0, the input on the tape, and the stack empty.

A PDA consults the current state, input symbol, and the symbol on the top of the stack to determine the machine transition. The transition function δ lists all possible transitions for a given state, symbol, and stack top combination. The value of the transition function

$$\delta(q_i, a, A) = \{[q_j, B], [q_k, C]\}$$

indicates that two transitions are possible when the automaton is in state q_i scanning an a with A on the top of the stack. The transition

$$[q_j, B] \in \delta(q_i, a, A)$$

new state / new stack top — current state / current input symbol / current stack top

causes the machine to

i) change the state from q_i to q_j,

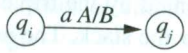
ii) process the symbol a (advance the tape head),

iii) remove A from the top of the stack (pop the stack), and

iv) push B onto the stack.

Since multiple transitions may be specified for a machine configuration, PDAs are nondeterministic machines.

A pushdown automaton can also be depicted by a state diagram. The labels on the arcs indicate both the input and the stack operation. The transition $\delta(q_i, a, A) = \{[q_j, B]\}$ is depicted by



The symbol / indicates replacement: A/B indicates that A is replaced on the top of the stack by B.

The domain of the transition function is $Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$, which indicates that λ may occur in either the input or stack top positions of a transition. A λ argument specifies that the value of the component should be neither consulted nor acted upon by the transition; the applicability of the transition is completely determined by the positions that do not contain λ.

When λ occurs as an argument in the stack position of the transition function, the transition is applicable whenever the current state and input symbol match those in the transition regardless of the status of the stack. The stack top may contain any symbol or the stack may be empty. The transition $[q_j, B] \in \delta(q_i, a, \lambda)$ is applicable whenever a machine is in state q_i scanning an a; the application of the transition will cause the machine to enter q_j and add B to the top of the stack.

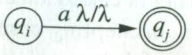
The symbol λ may also occur in the new stack position of a transition, $[q_j, \lambda] \in \delta(q_i, a, A)$. The execution such a transition does not push a symbol onto the stack. We will now look at several examples of the effect of λ in PDA transitions.

If the input position is λ, the transition does not process an input symbol. Thus, transition (i) pops and (ii) pushes the stack symbol A without altering the state or the input.

i) $[q_i, \lambda] \in \delta(q_i, \lambda, A)$

ii) $[q_i, A] \in \delta(q_i, \lambda, \lambda)$

iii) $[q_j, \lambda] \in \delta(q_i, a, \lambda)$



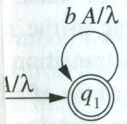
If the action specified by a transition has λ in the new stack top position, $[q_j, \lambda]$, no symbol is pushed onto the stack. Transition (iii) is the PDA equivalent of a finite automaton transition. The applicability is determined only by the state and input symbol; the transition does not consult nor does it alter the stack.

y the triple $[q_i, w, \alpha]$, where q_i is the machine
ack. The notation

$\vdash_M [q_j, v, \beta]$

obtained from $[q_i, w, \alpha]$ by a single transition
result of a sequence of transitions. When there
ot M is omitted. A computation of a PDA is
he machine in the initial state with an empty

M to accept the language $\{a^i b^i \mid i \geq 0\}$. The
and an empty stack. Processing input symbol
essing b pops the stack, matching the number
generated by the input string $aabb$ illustrates



$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
\vdash\ &[q_0, abb, A] \\
\vdash\ &[q_0, bb, AA] \\
\vdash\ &[q_1, b, A] \\
\vdash\ &[q_1, \lambda, \lambda]
\end{aligned}
$$

processes the entire string and halts in an
nditions become our criteria for acceptance.

string $w \in \Sigma^*$ is **accepted** by M if there is a

$\vdash [q_i, \lambda, \lambda]$

M), is the set of strings accepted by M.

led *successful*. A computation that processes
ing configuration is said to be unsuccessful.
ansition function, there may be computations
ut string. Computations of this form are also

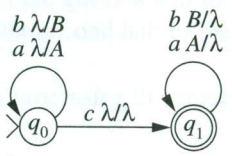
l pattern for nondeterministic machines; one
nd halts in a final state is sufficient for the

string to be in the language. The existence of additional unsuccessful computations does not affect the acceptance of the string.

Example 7.1.1

The PDA M accepts the language $\{wcw^R \mid w \in \{a, b\}^*\}$. The stack is used to record the string w as it is processed. Stack symbols A and B represent input a and b, respectively.

$$
\begin{aligned}
\text{M: } Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{A, B\} \\
F &= \{q_1\}
\end{aligned}
\qquad
\begin{aligned}
\delta(q_0, a, \lambda) &= \{[q_0, A]\} \\
\delta(q_0, b, \lambda) &= \{[q_0, B]\} \\
\delta(q_0, c, \lambda) &= \{[q_1, \lambda]\} \\
\delta(q_1, a, A) &= \{[q_1, \lambda]\} \\
\delta(q_1, b, B) &= \{[q_1, \lambda]\}
\end{aligned}
$$



A successful computation records the string w on the stack as it is processed. Once the c is encountered, the accepting state q_1 is entered and the stack contains a string representing w^R. The computation is completed by matching the remaining input with the elements on the stack. The computation of M with input $abcba$ is

$$
\begin{aligned}
&[q_0, abcba, \lambda] \\
\vdash\ &[q_0, bcba, A] \\
\vdash\ &[q_0, cba, BA] \\
\vdash\ &[q_1, ba, BA] \\
\vdash\ &[q_1, a, A] \\
\vdash\ &[q_1, \lambda, \lambda]
\end{aligned}
$$

□

A PDA is *deterministic* if there is at most one transition that is applicable for each combination of state, input symbol, and stack top. Two transitions $[q_j, C] \in \delta(q_i, u, A)$ and $[q_k, D] \in \delta(q_i, v, B)$ are called *compatible* if any of the following conditions are satisfied:

i) $u = v$ and $A = B$.
ii) $u = v$ and $A = \lambda$ or $B = \lambda$.
iii) $A = B$ and $u = \lambda$ or $v = \lambda$.
iv) $u = \lambda$ or $v = \lambda$ and $A = \lambda$ or $B = \lambda$.

Compatible transitions can be applied to the same machine configurations. A PDA is deterministic if it does not contain distinct compatible transitions. Both the PDA in Example 7.1.1 and the machine constructed to accept $\{a^i b^i \mid i \geq 0\}$ are deterministic.

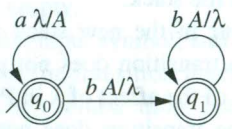
A PDA configuration is represented by the triple $[q_i, w, \alpha]$, where q_i is the machine state, w the unprocessed input, and α the stack. The notation

$$[q_i, w, \alpha] \vdash_M [q_j, v, \beta]$$

indicates that configuration $[q_j, v, \beta]$ can be obtained from $[q_i, w, \alpha]$ by a single transition of the PDA M. As before, $\vdash_M^*$ represents the result of a sequence of transitions. When there is no possibility of confusion, the subscript M is omitted. A computation of a PDA is a sequence of transitions beginning with the machine in the initial state with an empty stack.

We are now ready to construct a PDA M to accept the language $\{a^i b^i \mid i \geq 0\}$. The computation begins with the input string w and an empty stack. Processing input symbol a causes A to be pushed onto the stack. Processing b pops the stack, matching the number of b's to the number of a's. The computation generated by the input string $aabb$ illustrates the actions of M.

M: $Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$\Gamma = \{A\}$
$F = \{q_0, q_1\}$
$\delta(q_0, a, \lambda) = \{[q_0, A]\}$
$\delta(q_0, b, A) = \{[q_1, \lambda]\}$
$\delta(q_1, b, A) = \{[q_1, \lambda]\}$



$[q_0, aabb, \lambda]$
$\vdash [q_0, abb, A]$
$\vdash [q_0, bb, AA]$
$\vdash [q_1, b, A]$
$\vdash [q_1, \lambda, \lambda]$

The computation of M with input $a^i b^i$ processes the entire string and halts in an accepting state with an empty stack. These conditions become our criteria for acceptance.

Definition 7.1.2

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. A string $w \in \Sigma^*$ is **accepted** by M if there is a computation

$$[q_0, w, \lambda] \vdash^* [q_i, \lambda, \lambda]$$

where $q_i \in F$. The **language** of M, denoted L(M), is the set of strings accepted by M.

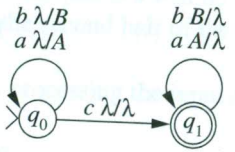
A computation that accepts a string is called *successful*. A computation that processes the entire input string and halts in a nonaccepting configuration is said to be unsuccessful. Because of the nondeterministic nature of the transition function, there may be computations that cannot complete the processing of the input string. Computations of this form are also considered unsuccessful.

Acceptance by a PDA follows the standard pattern for nondeterministic machines; one computation that processes the entire string and halts in a final state is sufficient for the string to be in the language. The existence of additional unsuccessful computations does not affect the acceptance of the string.

Example 7.1.1

The PDA M accepts the language $\{wcw^R \mid w \in \{a, b\}^*\}$. The stack is used to record the string w as it is processed. Stack symbols A and B represent input a and b, respectively.

M: $Q = \{q_0, q_1\}$
$\Sigma = \{a, b, c\}$
$\Gamma = \{A, B\}$
$F = \{q_1\}$
$\delta(q_0, a, \lambda) = \{[q_0, A]\}$
$\delta(q_0, b, \lambda) = \{[q_0, B]\}$
$\delta(q_0, c, \lambda) = \{[q_1, \lambda]\}$
$\delta(q_1, a, A) = \{[q_1, \lambda]\}$
$\delta(q_1, b, B) = \{[q_1, \lambda]\}$



A successful computation records the string w on the stack as it is processed. Once the c is encountered, the accepting state q_1 is entered and the stack contains a string representing w^R. The computation is completed by matching the remaining input with the elements on the stack. The computation of M with input $abcba$ is

$[q_0, abcba, \lambda]$
$\vdash [q_0, bcba, A]$
$\vdash [q_0, cba, BA]$
$\vdash [q_1, ba, BA]$
$\vdash [q_1, a, A]$
$\vdash [q_1, \lambda, \lambda]$

□

A PDA is *deterministic* if there is at most one transition that is applicable for each combination of state, input symbol, and stack top. Two transitions $[q_j, C] \in \delta(q_i, u, A)$ and $[q_k, D] \in \delta(q_i, v, B)$ are called *compatible* if any of the following conditions are satisfied:

i) $u = v$ and $A = B$.
ii) $u = v$ and $A = \lambda$ or $B = \lambda$.
iii) $A = B$ and $u = \lambda$ or $v = \lambda$.
iv) $u = \lambda$ or $v = \lambda$ and $A = \lambda$ or $B = \lambda$.

Compatible transitions can be applied to the same machine configurations. A PDA is deterministic if it does not contain distinct compatible transitions. Both the PDA in Example 7.1.1 and the machine constructed to accept $\{a^i b^i \mid i \geq 0\}$ are deterministic.

Clearly, every atomic PDA is a PDA in the sense of Definition 7.1.1. Theorem 7.2.1 shows that the languages accepted by atomic PDAs are the same as those accepted by PDAs. Moreover, it outlines a method to construct an equivalent atomic PDA from an arbitrary PDA.

Theorem 7.2.1

Let M be a PDA. Then there is an atomic PDA M′ with $L(M') = L(M)$.

Proof. To construct M′, the nonatomic transitions of M are replaced by a sequence of atomic transitions. Let $[q_j, B] \in \delta(q_i, a, A)$ be a transition of M. The atomic equivalent requires two new states, p_1 and p_2, and the transitions

$$[p_1, \lambda] \in \delta(q_i, a, \lambda)$$
$$\delta(p_1, \lambda, A) = \{[p_2, \lambda]\}$$
$$\delta(p_2, \lambda, \lambda) = \{[q_j, B]\}$$

to accomplish the same result as the nonatomic single transition.

In a similar manner, a transition that consists of changing the state and performing two additional actions can be replaced with a sequence of two atomic transitions. Replacing all nonatomic transitions with a sequence of atomic transitions produces an equivalent atomic PDA. ∎

An extended transition is an operation on a PDA that pushes a string of elements, rather than just a single element, onto the stack. The transition $[q_j, BCD] \in \delta(q_i, a, A)$ pushes BCD onto the stack with B becoming the new stack top. A PDA containing extended transitions is called an **extended PDA**. The apparent generalization does not increase the set of languages accepted by pushdown automata. Each extended PDA can be converted into an equivalent PDA in the sense of Definition 7.1.1.

To construct a PDA from an extended PDA, extended transitions are transformed into a sequence of transitions each of which pushes a single stack element. To achieve the result of an extended transition that pushes k elements requires $k - 1$ additional states. The sequence of transitions

$$[p_1, D] \in \delta(q_i, a, A)$$
$$\delta(p_1, \lambda, \lambda) = \{[p_2, C]\}$$
$$\delta(p_2, \lambda, \lambda) = \{[q_j, B]\}$$

pushes the string BCD onto the stack and leaves the machine in state q_j. The sequential execution of these three transitions produces the same result as the single extended transition $[q_j, BCD] \in \delta(q_i, a, A)$. The preceding argument can be generalized to yield Theorem 7.2.2.

Theorem 7.2.2

Let M be an extended PDA. Then there is a PDA M′ such that $L(M') = L(M)$.

Example 7.2.1

Let $L = \{a^i b^{2i} \mid i \geq 1\}$. A standard PDA, an atomic PDA, and an extended PDA are constructed to accept L. The input alphabet $\{a, b\}$, stack alphabet $\{A\}$, and accepting state q_1 are the same for each automaton. The states and transitions are

PDA	Atomic PDA	Extended PDA
$Q = \{q_0, q_1, q_2\}$	$Q = \{q_0, q_1, q_2, q_3, q_4\}$	$Q = \{q_0, q_1\}$
$\delta(q_0, a, \lambda) = \{[q_2, A]\}$	$\delta(q_0, a, \lambda) = \{[q_3, \lambda]\}$	$\delta(q_0, a, \lambda) = \{[q_0, AA]\}$
$\delta(q_2, \lambda, \lambda) = \{[q_0, A]\}$	$\delta(q_3, \lambda, \lambda) = \{[q_2, A]\}$	$\delta(q_0, b, A) = \{[q_1, \lambda]\}$
$\delta(q_0, b, A) = \{[q_1, \lambda]\}$	$\delta(q_2, \lambda, \lambda) = \{[q_0, A]\}$	$\delta(q_1, b, A) = \{[q_1, \lambda]\}$
$\delta(q_1, b, A) = \{[q_1, \lambda]\}$	$\delta(q_0, b, \lambda) = \{[q_4, \lambda]\}$	
	$\delta(q_4, \lambda, A) = \{[q_1, \lambda]\}$	
	$\delta(q_1, b, \lambda) = \{[q_4, \lambda]\}$	

As might be expected, the atomic PDA requires more transitions and the extended PDA fewer transitions than the equivalent standard PDA. The stack symbol A is used to count the number of matching b's required to accept the string. The extended transition $\delta(q_0, a, \lambda) = \{[q_0, AA]\}$ pushes both counters on the stack with a single transition. The standard PDA requires two transitions and the atomic PDA three to accomplish the same result. □

By Definition 7.1.2, an input string is accepted if there is a computation that processes the entire string and terminates in an accepting state with an empty stack. This type of acceptance is referred to as acceptance by *final state and empty stack*. Defining acceptance in terms of the final state or the configuration of the stack alone does not change the set of languages recognized by pushdown automaton.

A string w is accepted by *final state* if there is a computation $[q_0, w, \lambda] \vdash^* [q_i, \lambda, \alpha]$, where q_i is an accepting state and $\alpha \in \Gamma^*$, that is, a computation that processes the input and terminates in an accepting state. The contents of the stack at termination are irrelevant with acceptance by final state. A language accepted by final state is denoted L_F.

Lemma 7.2.3

Let L be a language accepted by a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ with acceptance defined by final state. Then there is a PDA that accepts L by final state and empty stack.

Proof. A PDA $M' = (Q \cup \{q_f\}, \Sigma, \Gamma, \delta', q_0, \{q_f\})$ is constructed from M by adding a state q_f and transitions for q_f. Intuitively, a computation in M′ that accepts a string should be identical to one in M except for the addition of transitions that empty the stack. The transition function δ' is constructed by augmenting δ with the transitions

$$\delta'(q_i, \lambda, \lambda) = \{[q_f, \lambda]\} \quad \text{for all } q_i \in F$$
$$\delta'(q_f, \lambda, A) = \{[q_f, \lambda]\} \quad \text{for all } A \in \Gamma.$$

Let $[q_0, w, \lambda] \overset{*}{\underset{M}{\vdash}} [q_i, \lambda, \alpha]$ be a computation of M accepting w by final state. In M′, this computation is completed by entering the accepting state q_f and emptying the stack

$$[q_0, w, \lambda]$$
$$\overset{*}{\underset{M}{\vdash}} [q_i, \lambda, \alpha]$$
$$\underset{M'}{\vdash} [q_f, \lambda, \alpha]$$
$$\overset{*}{\underset{M'}{\vdash}} [q_f, \lambda, \lambda]$$

showing that w is accepted in M′.

We must also guarantee that the new transitions do not cause M′ to accept strings that are not in L(M). The sole accepting state of M′ is q_f, which can be entered only on a transition from any accepting state of M. Since the transitions for q_f do not process input, entering q_f with unprocessed input results in an unsuccessful computation. Consequently, a string w is accepted by M′ only if there is computation in M that processes all of w and halts in an accepting state of M. That is, $w \in L(M')$ only when $w \in L(M)$ as desired. ■

A string w is said to be accepted by *empty stack* if there is a computation $[q_0, w, \lambda] \overset{*}{\vdash} [q_i, \lambda, \lambda]$. No restriction is placed on the halting state q_i. When acceptance is defined by empty stack, it is necessary to require at least one transition to permit the acceptance of languages that do not contain the null string. The language accepted by empty stack is denoted $L_E(M)$.

Lemma 7.2.4

Let L be a language accepted by a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0)$ with acceptance defined by empty stack. Then there is a PDA that accepts L by final state and empty stack.

Proof. Let $M' = (Q \cup \{q_0'\}, \Sigma, \Gamma, \delta', q_0', Q)$, where $\delta'(q_i, x, A) = \delta(q_i, x, A)$ and $\delta'(q_0', x, A) = \delta(q_0, x, A)$ for every $q_i \in Q$, $x \in \Sigma \cup \{\lambda\}$, and $A \in \Gamma \cup \{\lambda\}$. Every state of the original machine M is an accepting state of M′.

The computations of M and M′ are identical except that those of M begin in state q_0 and M′ in state q_0'. A computation of length one or more in M′ that halts with an empty stack also halts in a final state. Since q_0' is not accepting, the null string is accepted by M′ only if it is accepted by M. Thus, $L(M') = L_E(M)$. ■

Lemmas 7.2.3 and 7.2.4 show that a language accepted by either final state or empty stack alone is also accepted by final state and empty stack. Exercises 8 and 9 establish that any language accepted by final state and empty stack is accepted by a pushdown automaton using the less restrictive forms of acceptance. These observations yield the following theorem.

Theorem 7.2.5

The following three conditions are equivalent:

i) The language L is accepted by some PDA.

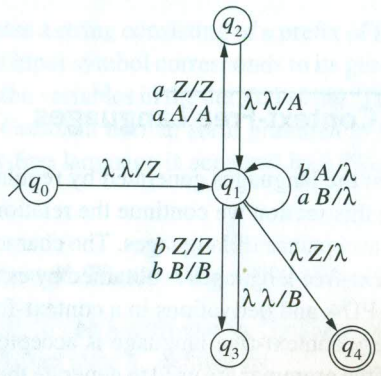
ii) There is a PDA M_1 with $L_F(M_1) = L$.

iii) There is a PDA M_2 with $L_E(M_2) = L$.

We have considered alternatives to the standard PDA model obtained by changing the acceptance criteria and the form of the transitions. Another common modification is to assume that there is a distinguished element that marks the bottom of the stack. A bottom marker can be read but not popped from the stack. Reading the bottom marker allows the machine to recognize an empty stack and act accordingly. The following example illustrates the role of a bottom marker and shows how it can be simulated in a standard PDA.

Example 7.2.2

The pushdown automaton M defined by the transitions



accepts strings that have the same number of a's and b's. The stack symbol Z plays the role of a bottom marker; it is placed on the stack with the first transition and remains throughout the computation.

The stack records the difference in the number of a's and b's that have been read. The stack will contain n A's if the automaton has processed n more a's than b's. Similarly, the number of B's on the stack indicates the number of b's in excess of the number of a's that have been processed. The bottom marker Z is read when the same number of a's and b's have been processed. The computation

$$[q_0, abba, \lambda]$$
$$\vdash [q_1, abba, Z]$$
$$\vdash [q_2, bba, Z]$$
$$\vdash [q_1, bba, AZ]$$
$$\vdash [q_1, ba, Z]$$
$$\vdash [q_3, a, Z]$$
$$\vdash [q_1, a, BZ]$$
$$\vdash [q_1, \lambda, Z]$$
$$\vdash [q_4, \lambda, \lambda]$$

exhibits the acceptance of $abba$. When an a is read with an A or Z on the top of the stack, an A is added to the stack by the transitions to q_2 and back to q_1. If the stack top is a B, the stack is popped in q_1 since reading the a decreases the difference between the number of b's and a's that have been processed. A similar strategy is employed when a b is read.

The lone accepting state of the automaton is q_4. If the input string has the same number of a's and b's, the transition to q_4 pops the Z and terminates the computation. □

The variations of pushdown automata that accept the same family of languages illustrate the robustness of acceptance using a stack memory. In the next section we show that the languages accepted by pushdown automata are precisely those generated by context-free grammars.

7.3 Acceptance of Context-Free Languages

In Chapter 6 we showed that the languages generated by regular grammars were precisely those accepted by DFAs. In this section we continue the relationship between grammatical generation and mechanical acceptance of languages. The characterization of pushdown automata as acceptors of context-free languages is obtained by establishing a correspondence between computations of a PDA and derivations in a context-free grammar.

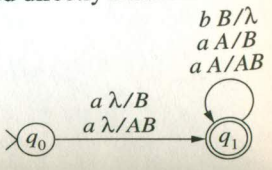
First we prove that every context-free language is accepted by an extended PDA. To accomplish this, the rules of the grammar are used to generate the transitions of an equivalent PDA. Let L be a context-free language and G a grammar in Greibach normal form with $L(G) = L$. The rules of G, except for $S \to \lambda$, have the form $A \to aA_1A_2 \ldots A_n$. In a leftmost derivation, the variables A_i must be processed in a left-to-right manner. Pushing $A_1A_2 \ldots A_n$ onto the stack stores the variables in the order required by the derivation. The PDA has two states: a start state q_0 and an accepting state q_1. An S rule of the form $S \to aA_1A_2 \ldots A_n$ generates a transition that processes the terminal symbol a, pushes the variables $A_1A_2 \ldots A_n$ onto the stack, and enters state q_1. The remainder of the computation uses the input symbol and the stack top to determine the appropriate transition.

The Greibach normal form grammar G that accepts $\{a^ib^i \mid i > 0\}$ is used to illustrate the construction of an equivalent PDA.

$$\begin{aligned} \text{G: } S &\to aAB \mid aB \\ A &\to aAB \mid aB \\ B &\to b \end{aligned}$$

The transition function of the equivalent PDA is defined directly from the rules of G.

$$\begin{aligned} \delta(q_0, a, \lambda) &= \{[q_1, AB],\ [q_1, B]\} \\ \delta(q_1, a, A) &= \{[q_1, AB],\ [q_1, B]\} \\ \delta(q_1, b, B) &= \{[q_1, \lambda]\} \end{aligned}$$



The computation obtained by processing $aaabbb$ exhibits the correspondence between derivations in the Greibach normal form grammar and computations in the associated PDA.

$$\begin{aligned} S &\Rightarrow aAB && [q_0, aaabbb, \lambda] \vdash [q_1, aabbb, AB] \\ &\Rightarrow aaABB && \vdash [q_1, abbb, ABB] \\ &\Rightarrow aaaBBB && \vdash [q_1, bbb, BBB] \\ &\Rightarrow aaabBB && \vdash [q_1, bb, BB] \\ &\Rightarrow aaabbB && \vdash [q_1, b, B] \\ &\Rightarrow aaabbb && \vdash [q_1, \lambda, \lambda] \end{aligned}$$

The derivation generates a string consisting of a prefix of terminals followed by a suffix of variables. Processing an input symbol corresponds to its generation in the derivation. The stack of the PDA contains the variables in the derived string. This strategy for the generation of a PDA equivalent to a Greibach normal form grammar is formalized in Theorem 7.3.1 to show that every context-free language is accepted by a PDA.

Theorem 7.3.1

Let L be a context-free language. Then there is a PDA that accepts L.

Proof. Let $G = (V, \Sigma, P, S)$ be a grammar in Greibach normal form that generates L. The extended PDA M with start state q_0 defined by

$$\begin{aligned} Q_M &= \{q_0, q_1\} \\ \Sigma_M &= \Sigma \\ \Gamma_M &= V - \{S\} \\ F_M &= \{q_1\} \end{aligned}$$

and transitions

$$\begin{aligned} &\delta(q_0, a, \lambda) = \{[q_1, w] \mid S \to aw \in P\} \\ &\delta(q_1, a, A) = \{[q_1, w] \mid A \to aw \in P \text{ and } A \in V - \{S\}\} \\ &\delta(q_0, \lambda, \lambda) = \{[q_1, \lambda]\} \text{ if } S \to \lambda \in P \end{aligned}$$

accepts L.

We first show that $L \subseteq L(M)$. Let $S \overset{*}{\Rightarrow} uw$ be a derivation with $u \in \Sigma^+$ and $w \in V^*$. We will prove that there is a computation

$$[q_0, u, \lambda] \overset{*}{\vdash} [q_1, \lambda, w]$$

in M. The proof is by induction on the length of the derivation and utilizes the correspondence between derivations in G and computations of M.

The basis consists of derivations $S \Rightarrow aw$ of length one. The transition generated by the rule $S \to aw$ yields the desired computation. Assume that for all strings uw generated by derivations $S \overset{n}{\Rightarrow} uw$ there is a computation

$$[q_0, u, \lambda] \overset{*}{\vdash} [q_1, \lambda, w]$$

in M.

Now let $S \overset{n+1}{\Longrightarrow} uw$ be a derivation with $u = va \in \Sigma^+$ and $w \in V^*$. This derivation can be written

$$S \overset{n}{\Rightarrow} vAw_2 \Rightarrow uw,$$

where $w = w_1w_2$ and $A \to aw_1$ is a rule in P. The inductive hypothesis and the transition $[q_1, w_1] \in \delta(q_1, a, A)$ combine to produce the computation

$$[q_0, va, \lambda] \overset{*}{\vdash} [q_1, a, Aw_2]$$
$$\vdash [q_1, \lambda, w_1w_2].$$

For every string u in L of positive length, the acceptance of u is exhibited by the computation in M corresponding to the derivation $S \overset{*}{\Rightarrow} u$. If $\lambda \in L$, then $S \to \lambda$ is a rule of G and the computation $[q_0, \lambda, \lambda] \vdash [q_1, \lambda, \lambda]$ accepts the null string.

The opposite inclusion, $L(M) \subseteq L$, is established by showing that for every computation $[q_0, u, \lambda] \overset{*}{\vdash} [q_1, \lambda, w]$ there is a corresponding derivation $S \overset{*}{\Rightarrow} uw$ in G. The proof is by induction on the number of transitions in a computation and is left as an exercise. ■

To complete the characterization of context-free languages as precisely those accepted by pushdown automata, we must show that every language accepted by a PDA is context-free. The rules of a context-free grammar are constructed from the transitions of the automaton so that the application of a rule corresponds to a transition in the computation in the PDA. To simplify the proof, we divide the presentation into four stages:

1. The addition of transitions to the PDA so that each string in the language is accepted by a computation in which every transition both pops and pushes the stack;
2. The construction of the rules of a grammar from the modified PDA;
3. The presentation of an example that illustrates the correspondence between computations of the PDA and derivations of the grammar;
4. Finally, the formal proof that the language of the grammar and the PDA are the same.

The first two steps are constructive—adding transitions and building rules. The final step is accomplished by Lemmas 7.3.3 and 7.3.4, which show that the rules generate exactly the strings accepted by the PDA. We start with an arbitrary PDA M and show that L(M) is context-free. The proof begins by modifying M so that the transitions can be converted to rules.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. An extended PDA M′ with transition function δ' is obtained from M by augmenting δ with the transitions

i) If $[q_j, \lambda] \in \delta(q_i, u, \lambda)$, then $[q_j, A] \in \delta'(q_i, u, A)$ for every $A \in \Gamma$.

ii) If $[q_j, B] \in \delta(q_i, u, \lambda)$, then $[q_j, BA] \in \delta'(q_i, u, A)$ for every $A \in \Gamma$.

The interpretation of these transitions is that a transition of M that does not remove an element from the stack can be considered to initially pop the stack and later replace the same symbol on the top of the stack. Any string accepted by a computation that utilizes a new transition can also be obtained by applying the original transition; hence, $L(M) = L(M')$.

A grammar $G = (V, \Sigma, P, S)$ is constructed from the transitions of M′. The alphabet of G is the input alphabet of M′. The variables of G consist of a start symbol S and objects of the form $\langle q_i, A, q_j \rangle$ where the q's are states of M′ and $A \in \Gamma \cup \{\lambda\}$. The variable $\langle q_i, A, q_j \rangle$ represents a computation that begins in state q_i, ends in q_j, and removes the symbol A from the stack. The rules of G are constructed as follows:

1. $S \to \langle q_0, \lambda, q_j \rangle$ for each $q_j \in F$.
2. Each transition $[q_j, B] \in \delta'(q_i, x, A)$, where $A \in \Gamma \cup \{\lambda\}$, generates the set of rules
$$\{\langle q_i, A, q_k \rangle \to x \langle q_j, B, q_k \rangle \mid q_k \in Q\}.$$
3. Each transition $[q_j, BA] \in \delta'(q_i, x, A)$, where $A \in \Gamma$, generates the set of rules
$$\{\langle q_i, A, q_k \rangle \to x \langle q_j, B, q_n \rangle \langle q_n, A, q_k \rangle \mid q_k, q_n \in Q\}.$$
4. For each state $q_k \in Q$,
$$\langle q_k, \lambda, q_k \rangle \to \lambda.$$

A derivation begins with a rule of type 1 whose right-hand side represents a computation that begins in state q_0, ends in a final state, and terminates with an empty stack, in other words, a successful computation in M′. Rules of types 2 and 3 trace the action of the machine. Rules of type 3 correspond to the extended transitions of M′. In a computation, these transitions increase the size of the stack. The effect of the corresponding rule is to introduce an additional variable into the derivation.

Rules of type 4 are used to terminate derivations. The rule $\langle q_k, \lambda, q_k \rangle \to \lambda$ represents a computation from a state q_k to itself that does not alter the stack, that is, the null computation.

Example 7.3.1

A grammar G is constructed from the PDA M. The language of M is the set $\{a^n c b^n \mid n \geq 0\}$.

M: $Q = \{q_0, q_1\}$ $\quad \delta(q_0, a, \lambda) = \{[q_0, A]\}$
$\Sigma = \{a, b, c\}$ $\quad \delta(q_0, c, \lambda) = \{[q_1, \lambda]\}$
$\Gamma = \{A\}$ $\quad \delta(q_1, b, A) = \{[q_1, \lambda]\}$
$F = \{q_1\}$

The transitions $\delta'(q_0, a, A) = \{[q_0, AA]\}$ and $\delta'(q_0, c, A) = \{[q_1, A]\}$ are added to M to construct M′. The rules of the equivalent grammar G and the transition from which they were constructed are

Transition	Rule
	$S \to \langle q_0, \lambda, q_1\rangle$
$\delta(q_0, a, \lambda) = \{[q_0, A]\}$	$\langle q_0, \lambda, q_0\rangle \to a\langle q_0, A, q_0\rangle$
	$\langle q_0, \lambda, q_1\rangle \to a\langle q_0, A, q_1\rangle$
$\delta(q_0, a, A) = \{[q_0, AA]\}$	$\langle q_0, A, q_0\rangle \to a\langle q_0, A, q_0\rangle\langle q_0, A, q_0\rangle$
	$\langle q_0, A, q_1\rangle \to a\langle q_0, A, q_0\rangle\langle q_0, A, q_1\rangle$
	$\langle q_0, A, q_0\rangle \to a\langle q_0, A, q_1\rangle\langle q_1, A, q_0\rangle$
	$\langle q_0, A, q_1\rangle \to a\langle q_0, A, q_1\rangle\langle q_1, A, q_1\rangle$
$\delta(q_0, c, \lambda) = \{[q_1, \lambda]\}$	$\langle q_0, \lambda, q_0\rangle \to c\langle q_1, \lambda, q_0\rangle$
	$\langle q_0, \lambda, q_1\rangle \to c\langle q_1, \lambda, q_1\rangle$
$\delta(q_0, c, A) = \{[q_1, A]\}$	$\langle q_0, A, q_0\rangle \to c\langle q_1, A, q_0\rangle$
	$\langle q_0, A, q_1\rangle \to c\langle q_1, A, q_1\rangle$
$\delta(q_1, b, A) = \{[q_1, \lambda]\}$	$\langle q_1, A, q_0\rangle \to b\langle q_1, \lambda, q_0\rangle$
	$\langle q_1, A, q_1\rangle \to b\langle q_1, \lambda, q_1\rangle$
	$\langle q_0, \lambda, q_0\rangle \to \lambda$
	$\langle q_1, \lambda, q_1\rangle \to \lambda$

□

The relationship between computations in a PDA and derivations in the associated grammar are demonstrated using the grammar and PDA of Example 7.3.1. The derivation begins with the application of an S rule; the remaining steps correspond to the processing of an input symbol in M′. The first component of the leftmost variable contains the state of the computation. The third component of the rightmost variable contains the accepting state in which the computation will terminate. The stack can be obtained by concatenating the second components of the variables.

$$
\begin{aligned}
[q_0, aacbb, \lambda] \quad & S \Rightarrow \langle q_0, \lambda, q_1\rangle \\
\vdash [q_0, acbb, A] \quad & \Rightarrow a\langle q_0, A, q_1\rangle \\
\vdash [q_0, cbb, AA] \quad & \Rightarrow aa\langle q_0, A, q_1\rangle\langle q_1, A, q_1\rangle \\
\vdash [q_1, bb, AA] \quad & \Rightarrow aac\langle q_1, A, q_1\rangle\langle q_1, A, q_1\rangle \\
\vdash [q_1, b, A] \quad & \Rightarrow aacb\langle q_1, \lambda, q_1\rangle\langle q_1, A, q_1\rangle \\
& \Rightarrow aacb\langle q_1, A, q_1\rangle \\
\vdash [q_1, \lambda, \lambda] \quad & \Rightarrow aacbb\langle q_1, \lambda, q_1\rangle \\
& \Rightarrow aacbb
\end{aligned}
$$

The variable $\langle q_0, \lambda, q_1\rangle$, obtained by the application of the S rule, indicates that a computation from state q_0 to state q_1 that does not alter the stack is required. The result of subsequent rule application signals the need for a computation from q_0 to q_1 that removes an A from the top of the stack. The fourth rule application demonstrates the necessity for augmenting the transitions of M when δ contains transitions that do not remove a symbol from the stack. The application of the rule $\langle q_0, A, q_1\rangle \to c\,\langle q_1, A, q_1\rangle$ represents a computation that processes c without removing the A from the top of the stack.

We are now ready to prove that a language accepted by a PDA is context-free. This result combines with Theorem 7.3.1 to establish the equivalence of string generation using context-free rules and string acceptance by pushdown automata.

Theorem 7.3.2

Let M be a PDA. Then there is a context-free grammar G with L(G) = L(M).

The grammar G is constructed as outlined from the extended PDA M′ that is equivalent to M. We must show that there is a derivation $S \overset{*}{\Rightarrow} w$ if, and only if, $[q_0, w, \lambda] \overset{*}{\vdash} [q_j, \lambda, \lambda]$ for some $q_j \in F$. This follows from Lemmas 7.3.3 and 7.3.4, which establish the correspondence of derivations in G to computations in M′.

Lemma 7.3.3

If $\langle q_i, A, q_j\rangle \overset{*}{\Rightarrow} w$ where $w \in \Sigma^*$ and $A \in \Gamma \cup \{\lambda\}$, then $[q_i,\ w,\ A] \overset{*}{\vdash} [q_j,\ \lambda,\ \lambda]$.

Proof. The proof is by induction on the length of derivations of terminal strings from variables of the form $\langle q_i, A, q_j\rangle$. The basis consists of derivations of strings consisting of a single rule application. The null string is the only terminal string derivable with one rule application. The derivation has the form $\langle q_i, \lambda, q_i\rangle \Rightarrow \lambda$ utilizing a rule of type 4. The null computation in state q_i yields $[q_i, \lambda, \lambda] \overset{*}{\vdash} [q_i, \lambda, \lambda]$ as desired.

Assume that there is a computation $[q_i, v, A] \overset{*}{\vdash} [q_j, \lambda, \lambda]$ whenever $\langle q_i, A, q_j\rangle \overset{n}{\Rightarrow} v$. Let w be a terminal string derivable from $\langle q_i, A, q_j\rangle$ by a derivation of length $n + 1$. The first step of the derivation consists of the application of a rule of type 2 or 3. A derivation initiated by a rule of type 2 can be written

$$
\begin{aligned}
\langle q_i, A, q_j\rangle &\Rightarrow u\,\langle q_k, B, q_j\rangle \\
&\overset{n}{\Rightarrow} uv = w,
\end{aligned}
$$

where $\langle q_i, A, q_j\rangle \to u\,\langle q_k, B, q_j\rangle$ is a rule of G. By the inductive hypothesis, there is a computation $[q_k, v, B] \overset{*}{\vdash} [q_j, \lambda, \lambda]$ corresponding to the derivation $\langle q_k, B, q_j\rangle \overset{n}{\Rightarrow} v$.

The rule $\langle q_i, A, q_j\rangle \to u\,\langle q_k, B, q_j\rangle$ in G is generated by a transition $[q_k, B] \in \delta(q_i, u, A)$. Combining this transition with the computation established by the inductive hypothesis yields

$$
\begin{aligned}
[q_i, uv, A] &\vdash [q_k, v, B] \\
&\overset{*}{\vdash} [q_j, \lambda, \lambda].
\end{aligned}
$$

If the first step of the derivation is a rule of type 3, the derivation can be written

$$\langle q_i, A, q_j\rangle \Rightarrow u\,\langle q_k, B, q_m\rangle\langle q_m, A, q_j\rangle \overset{n}{\Rightarrow} w.$$

The corresponding computation is constructed from the transition $[q_k, BA] \in \delta(q_i, u, A)$ and two invocations of the inductive hypothesis. ■

Lemma 7.3.4

If $[q_i, w, A] \overset{*}{\vdash} [q_j, \lambda, \lambda]$ where $A \in \Gamma \cup \{\lambda\}$, then there is a derivation $\langle q_i, A, q_j\rangle \overset{*}{\Rightarrow} w$.

Proof. The null computation from configuration $[q_i, \lambda, \lambda]$ is the only computation of M that uses no transitions. The corresponding derivation consists of a single application of the rule $\langle q_i, \lambda, q_i\rangle \to \lambda$.

Assume that every computation $[q_i, v, A] \overset{n}{\vdash} [q_j, \lambda, \lambda]$ has a corresponding derivation $\langle q_i, A, q_j\rangle \overset{*}{\Rightarrow} v$ in G. Consider a computation of length $n + 1$. A computation of the prescribed form beginning with a nonextended transition can be written

$$\begin{aligned}&[q_i, w, A]\\ &\vdash [q_k, v, B]\\ &\overset{n}{\vdash} [q_j, \lambda, \lambda],\end{aligned}$$

where $w = uv$ and $[q_k, B] \in \delta(q_i, u, A)$. By the inductive hypothesis, there is a derivation $\langle q_k, B, q_j\rangle \overset{*}{\Rightarrow} v$. The first transition generates the rule $\langle q_i, A, q_j\rangle \to u\,\langle q_k, B, q_j\rangle$ in G. Hence a derivation of w from $\langle q_i, A, q_j\rangle$ can be obtained by

$$\begin{aligned}\langle q_i, A, q_j\rangle &\Rightarrow u\,\langle q_k, B, q_j\rangle\\ &\overset{*}{\Rightarrow} uv.\end{aligned}$$

A computation in M′ beginning with an extended transition $[q_j, BA] \in \delta(q_i, u, A)$ has the form

$$\begin{aligned}&[q_i, w, A]\\ &\vdash [q_k, v, BA]\\ &\overset{*}{\vdash} [q_m, y, A]\\ &\overset{*}{\vdash} [q_j, \lambda, \lambda],\end{aligned}$$

where $w = uv$ and $v = xy$. The rule $\langle q_i, A, q_j\rangle \to u\,\langle q_k, B, q_m\rangle\langle q_m, A, q_j\rangle$ is generated by the first transition of the computation. By the inductive hypothesis, G contains derivations

$$\begin{aligned}\langle q_k, B, q_m\rangle &\overset{*}{\Rightarrow} x\\ \langle q_m, A, q_j\rangle &\overset{*}{\Rightarrow} y.\end{aligned}$$

Combining these derivations with the preceding rule produces a derivation of w from $\langle q_i, A, q_j\rangle$. ■

Proof of Theorem 7.3.2. Let w be any string in L(G) with derivation $S \Rightarrow \langle q_0, \lambda, q_j\rangle \overset{*}{\Rightarrow} w$. By Lemma 7.3.3, there is a computation $[q_0, w, \lambda] \overset{*}{\underset{M}{\vdash}} [q_j, \lambda, \lambda]$ exhibiting the acceptance of w by M′.

Conversely, if $w \in L(M) = L(M')$, then there is a computation $[q_0, w, \lambda] \overset{*}{\vdash} [q_j, \lambda, \lambda]$ that accepts w. Lemma 7.3.4 establishes the existence of a corresponding derivation $\langle q_0, \lambda, q_j\rangle \overset{*}{\Rightarrow} w$ in G. Since q_j is an accepting state, G contains a rule $S \to \langle q_0, \lambda, q_j\rangle$. Initiating the previous derivation with this rule generates w in the grammar G. ■

7.4 The Pumping Lemma for Context-Free Languages

The pumping lemma for regular languages, Theorem 6.6.3, showed that sufficiently long strings in a regular language have a substring that can be repeated any number of times with the resulting string remaining in the language. In this section we establish a pumping lemma for context-free languages. For context-free languages, however, pumping refers to simultaneously repeating two substrings. The ability to generate any context-free language with a Chomsky normal form grammar provides the structure needed to prove the pumping lemma.

There are two milestones in the proof of the pumping lemma. Using the properties of derivation trees built using the rules of Chomsky normal form grammars, we obtain a number k such that

1. the derivation of any string of length k or more must have a recursive subderivation $A \overset{*}{\Rightarrow} vAx$, with $v, x \in \Sigma^*$, and
2. the strings v and x can be simultaneously pumped in z with the resulting string remaining in the language.

The relationship between the number of leaves and depth of a binary tree is used to achieve the first milestone, and the repetition of the recursive subderivation establishes the latter. The relationship between string length and depth of a derivation tree for Chomsky normal form grammars is obtained in Lemma 7.4.1 and restated in Corollary 7.4.2.

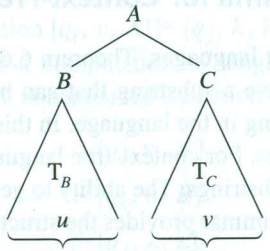
Lemma 7.4.1

Let G be a context-free grammar in Chomsky normal form and $A \overset{*}{\Rightarrow} w$ a derivation of $w \in \Sigma^*$ with derivation tree T. If the depth of T is n, then $length(w) \leq 2^{n-1}$.

Proof. The proof is by induction on the depth of the derivation trees that generate terminal strings. Since G is in Chomsky normal form, a derivation tree of depth 1 that represents the generation of a terminal string must have one of the following two forms.

In either case, the length of the derived string is less than or equal to $2^0 = 1$ as required.

Assume that the property holds for all derivation trees of depth n or less. Let $A \stackrel{*}{\Rightarrow} w$ be a derivation with tree T of depth $n+1$. Since the grammar is in Chomsky normal form, the derivation can be written $A \Rightarrow BC \stackrel{*}{\Rightarrow} uv$ where $B \stackrel{*}{\Rightarrow} u$, $C \stackrel{*}{\Rightarrow} v$, and $w = uv$. The derivation tree of $A \stackrel{*}{\Rightarrow} w$ is constructed from T_B and T_C, the derivation trees of $B \stackrel{*}{\Rightarrow} u$ and $C \stackrel{*}{\Rightarrow} v$.



The trees T_B and T_C both have depth n or less. By the inductive hypothesis, $length(u) \leq 2^{n-1}$ and $length(v) \leq 2^{n-1}$. Therefore, $length(w) = length(uv) \leq 2^n$. ■

Corollary 7.4.2

Let $G = (V, \Sigma, P, S)$ be a context-free grammar in Chomsky normal form and $S \stackrel{*}{\Rightarrow} w$ a derivation of $w \in L(G)$. If $length(w) \geq 2^n$, then the derivation tree has depth at least $n+1$.

Theorem 7.4.3 (Pumping Lemma for Context-Free Languages)

Let L be a context-free language. There is a number k, depending on L, such that any string $z \in L$ with $length(z) > k$ can be written $z = uvwxy$ where

i) $length(vwx) \leq k$

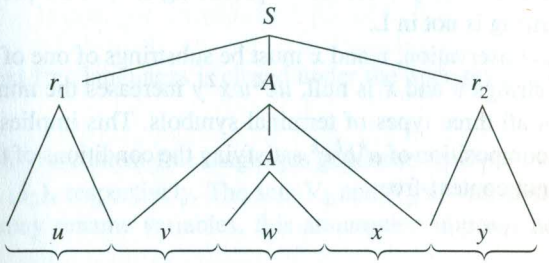
ii) $length(v) + length(x) > 0$

iii) $uv^iwx^iy \in L$, for $i \geq 0$.

Proof. Let $G = (V, \Sigma, P, S)$ be a Chomsky normal form grammar that generates L and let $k = 2^n$ with $n = card(V)$. We show that all strings in L with length k or greater can be decomposed to satisfy the conditions of the pumping lemma. Let $z \in L(G)$ be such a string and let $S \stackrel{*}{\Rightarrow} z$ be a derivation in G. By Corollary 7.4.2, there is a path of length at least $n+1 = card(V)+1$ in the derivation tree of $S \stackrel{*}{\Rightarrow} z$.

Let $\mathbf{p}$ be a path of maximal length from the root S to a leaf of the derivation tree. Then $\mathbf{p}$ must contain at least $n+2$ nodes, all of which are labeled by variables except the leaf node, which is labeled by a terminal symbol. The pigeonhole principle guarantees that some variable A must occur twice in the final $n+2$ nodes of this path. Although A may appear more than twice in the path, we will be concerned only with its last and next to last occurrence in $\mathbf{p}$.

Translating the properties of a path in the derivation tree to subderivations, the derivation of z can be depicted



where $z = uvwxy$. The derivation $S \stackrel{*}{\Rightarrow} r_1Ar_2$ produces the next to last occurrence of the variable A. The subderivation $A \stackrel{+}{\Rightarrow} vAx$ may be omitted or repeated any number of times before applying $A \stackrel{*}{\Rightarrow} w$ to halt the recursion. The resulting derivations generate the strings $uv^iwx^iy \in L(G) = L$.

We now show that conditions (i) and (ii) in the pumping lemma are satisfied by this decomposition. The subderivation $A \stackrel{+}{\Rightarrow} vAx$ must begin with a rule of the form $A \rightarrow BC$. The second occurrence of the variable A is derived from either B or C. If it is derived from B, the derivation can be written

$$\begin{aligned} A &\Rightarrow BC \\ &\stackrel{*}{\Rightarrow} vAsC \\ &\stackrel{*}{\Rightarrow} vAst \\ &= vAx. \end{aligned}$$

The string t is nonnull since it is obtained by a derivation from a variable in a Chomsky normal form grammar that is not the start symbol of the grammar. It follows that x is also nonnull. If the second occurrence of A is derived from the variable C, a similar argument shows that v must be nonnull.

The subpath between the final two occurrences of A in the path $\mathbf{p}$ must be of length at most $n+2$. The derivation tree generated by the derivation $A \stackrel{*}{\Rightarrow} vwx$ has depth of at most $n+1$. It follows from Lemma 7.4.1 that the string vwx obtained from this derivation has length $k = 2^n$ or less. ■

Like its counterpart for regular languages, the pumping lemma provides a tool for demonstrating that languages are not context-free. By the pumping lemma, every sufficiently long string in a context-free grammar must have pumpable substrings. Thus we can show that a language is not context-free by finding a string that has no decomposition $uvwxy$ that satisfies the requirement of Theorem 7.4.3.

Example 7.4.1

The language $L = \{a^i b^i c^i \mid i \geq 0\}$ is not context-free. Assume L is context-free. By Theorem 7.4.1, the string $z = a^k b^k c^k$, where k is the number specified by the pumping lemma, can be decomposed into substrings $uvwxy$ that satisfy the repetition properties. Consider the possibilities for the substrings v and x. If either of these contains more than one type of terminal symbol, then uv^2wx^2y contains a b preceding an a or a c preceding a b. In either case, the resulting string is not in L.

By the previous observation, v and x must be substrings of one of a^k, b^k, or c^k. Since at most one of the strings v and x is null, uv^2wx^2y increases the number of at least one, maybe two, but not all three types of terminal symbols. This implies that $uv^2wx^2y \notin L$. Thus there is no decomposition of $a^k b^k c^k$ satisfying the conditions of the pumping lemma; consequently, L is not context-free. □

Example 7.4.2

The language $L = \{a^i b^j a^i b^j \mid i, j \geq 0\}$ is not context-free. Let k be the number specified by the pumping lemma and $z = a^k b^k a^k b^k$. Assume there is a decomposition $uvwxy$ of z that satisfies the conditions of the pumping lemma. Condition (ii) requires the length of vwx to be at most k. This implies that vwx is a string containing only one type of terminal or the concatenation of two such strings. That is,

i) $vwx \in a^*$ or $vwx \in b^*$, or

ii) $vwx \in a^*b^*$ or $vwx \in b^*a^*$.

By an argument similar to that in Example 7.4.1, the substrings v and x must contain only one type of terminal. Pumping v and x increases the number of a's or b's in only one of the substrings in z. Since there is no decomposition of z satisfying the conditions of the pumping lemma, we conclude that L is not context-free. □

Example 7.4.3

The language $L = \{w \in a^* \mid length(w) \text{ is prime}\}$ is not context-free. Assume L is context-free and n a prime greater than k, the constant of Theorem 7.4.3. The string a^n must have a decomposition $uvwxy$ that satisfies the conditions of the pumping lemma. Let $m = length(u) + length(w) + length(y)$. The length of any string uv^iwx^iy is $m + i(n - m)$.

In particular, $length(wv^{n+1}wx^{n+1}y) = m + (n+1)(n-m) = n(n-m+1)$. Both of the terms in the preceding product are natural numbers greater than 1. Consequently, the length of $uv^{n+1}wx^{n+1}y$ is not prime and the string is not in L. Thus, L is not context-free. □

7.5 Closure Properties of Context-Free Languages

The flexibility of the rules of context-free grammars is used to establish closure results for the set of context-free languages. Operations that preserve context-free languages provide another tool for proving that languages are context-free. These operations, combined with the pumping lemma, can also be used to show that certain languages are not context-free.

Theorem 7.5.1

The family of context-free languages is closed under the operations union, concatenation, and Kleene star.

Proof. Let L_1 and L_2 be context-free languages generated by $G_1 = (V_1, \Sigma_1, P_1, S_1)$ and $G_2 = (V_2, \Sigma_2, P_2, S_2)$, respectively. The sets V_1 and V_2 of variables are assumed to be disjoint. Since we may rename variables, this assumption imposes no restriction on the grammars.

A context-free grammar will be constructed from G_1 and G_2 that establishes the desired closure property.

Union: Define $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S)$. A string w is in L(G) if, and only if, there is a derivation $S \Rightarrow S_i \overset{*}{\underset{G_i}{\Rightarrow}} w$ for $i = 1$ or 2. Thus w is in L_1 or L_2. On the other hand, any derivation $S_i \overset{*}{\underset{G_i}{\Rightarrow}} w$ can be initialized with the rule $S \to S_i$ to generate w in G.

Concatenation: Define $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1S_2\}, S)$. The start symbol initiates derivations in both G_1 and G_2. A leftmost derivation of a terminal string in G has the form $S \Rightarrow S_1S_2 \overset{*}{\Rightarrow} uS_2 \overset{*}{\Rightarrow} uv$, where $u \in L_1$ and $v \in L_2$. The derivation of u uses only rules from P_1 and v rules from P_2. Hence $L(G) \subseteq L_1L_2$. The opposite inclusion is established by observing that every string w in L_1L_2 can be written uv with $u \in L_1$ and $v \in L_2$. The derivations $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} u$ and $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} v$, along with the S rule of G, generate w in G.

Kleene star: Define $G = (V_1, \Sigma_1, P_1 \cup \{S \to S_1S \mid \lambda\}, S)$. The S rule of G generates any number of copies of S_1. Each of these, in turn, initiates the derivation of a string in L_1. The concatenation of any number of strings from L_1 yields L_1^*. ■

Theorem 7.5.1 presented positive closure results for the set of context-free languages. A simple example is given to show that the context-free languages are not closed under intersection. Finally, we combine the closure properties of union and intersection to obtain a similar negative result for complementation.

Theorem 7.5.2

The set of context-free languages is not closed under intersection or complementation.

Proof.

Intersection: Let $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$ and $L_2 = \{a^j b^i c^i \mid i, j \geq 0\}$. L_1 and L_2 are both context-free since they are generated by G_1 and G_2, respectively.

$$G_1\colon\ S \to BC \qquad G_2\colon\ S \to AB$$
$$B \to aBb \mid \lambda \qquad A \to aA \mid \lambda$$
$$C \to cC \mid \lambda \qquad B \to bBc \mid \lambda$$

The intersection of L_1 and L_2 is the set $\{a^i b^i c^i \mid i \geq 0\}$, which is not context-free by Example 7.4.1.

Complementation: Let L_1 and L_2 be any two context-free languages. If the context-free languages are closed under complementation, then by Theorem 7.5.1, the language

$$L = \overline{\overline{L_1} \cup \overline{L_2}}$$

is context-free. By DeMorgan's Law, $L = L_1 \cap L_2$. This implies that the context-free languages are closed under intersection, contradicting the result of part (i). ∎

Exercise 9 of Chapter 6 showed that the intersection of a regular and context-free language need not be regular. The correspondence between languages and pushdown automata is used to establish a positive closure property for the intersection of regular and context-free languages.

Let R be a regular language accepted by a DFA N and L a context-free language accepted by PDA M. We show that $R \cap L$ is context-free by constructing a single PDA that simulates the operation of both N and M. The states of this composite machine are ordered pairs consisting of a state from M and one from N.

Theorem 7.5.3

Let R be a regular language and L a context-free language. Then the language $R \cap L$ is context-free.

Proof. Let $N = (Q_N, \Sigma_N, \delta_N, q_0, F_N)$ be a DFA that accepts R and let $M = (Q_M, \Sigma_M, \Gamma, \delta_M, p_0, F_M)$ be a PDA that accepts L. The machines N and M are combined to construct a PDA

$$M' = (Q_M \times Q_N, \Sigma_M \cup \Sigma_N, \Gamma, \delta, [p_0, q_0], F_M \times F_N)$$

that accepts $R \cap L$. The transition function of M' is defined to "run the machines M and N in parallel." The first component of the ordered pair traces the sequence of states entered by the machine M and the second component by N. The transition function of M' is defined by

i) $\delta([p, q], a, A) = \{[[p', q'], B] \mid [p', B] \in \delta_M(p, a, A) \text{ and } \delta_N(q, a) = q'\}$

ii) $\delta([p, q], \lambda, A) = \{[[p', q], B] \mid [p', B] \in \delta_M(p, \lambda, A)\}$.

Every transition of a DFA processes an input symbol, whereas a PDA may contain transitions that do not process input. The transitions introduced by condition (ii) simulate the action of a PDA transition that does not process an input symbol.

A string w is accepted by M' if there is a computation

$$[[p_0, q_0], w, \lambda] \vdash^* [[p_i, q_j], \lambda, \lambda],$$

where p_i and q_j are final states of M and N, respectively.

The inclusion $L(N) \cap L(M) \subseteq L(M')$ is established by showing that there is a computation

$$[[p_0, q_0], w, \lambda] \overset{*}{\underset{M}{\vdash}} [[p_i, q_j], u, \alpha]$$

whenever

$$[p_0, w, \lambda] \overset{*}{\underset{M}{\vdash}} [p_i, u, \alpha] \quad \text{and} \quad [q_0, w] \overset{*}{\underset{N}{\vdash}} [q_j, u]$$

are computations in M and N. The proof is by induction on the number of transitions in the PDA M.

The basis consists of the null computation in M. This computation terminates with $p_i = p_0$, $u = w$, and M containing an empty stack. The only computation in N that terminates with the original string is the null computation; thus, $q_j = q_0$. The corresponding computation in the composite machine is the null computation in M'.

Assume the result holds for all computations of M having length n. Let

$$[p_0, w, \lambda] \overset{n+1}{\underset{M}{\vdash}} [p_i, u, \alpha] \qquad \text{and} \qquad [q_0, w] \overset{*}{\underset{N}{\vdash}} [q_j, u]$$

be computations in the PDA and DFA, respectively. The computation in M can be written

$$[p_0, w, \lambda]$$
$$\overset{n}{\vdash} [p_k, v, \beta]$$
$$\vdash [p_i, u, \alpha],$$

where either $v = u$ or $v = au$. To show that there is a computation $[[p_0, q_0], w, \lambda] \overset{*}{\underset{M}{\vdash}} [[p_i, q_j], u, \alpha]$, we consider each of the possibilities for v separately.

Case 1: $v = u$. In this case, the final transition of the computation in M does not process an input symbol. The computation in M is completed by a transition of the form $[p_i, B] \in \delta_M(p_k, \lambda, A)$. This transition generates $[[p_i, q_j], B] \in \delta([p_k, q_j], \lambda, A)$ in M'. The computation

$$[[p_0, q_0], w, \lambda] \overset{n}{\underset{M}{\vdash}} [[p_k, q_j], v, \beta]$$
$$\underset{M}{\vdash} [[p_i, q_j], v, \alpha]$$

is obtained from the inductive hypothesis and the preceding transition of M'.

Case 2: $v = au$. The computation in N that reduces w to u can be written

$$[q_0, w]$$
$$\overset{*}{\underset{N}{\vdash}} [q_m, v]$$
$$\underset{N}{\vdash} [q_j, u],$$

where the final step utilizes a transition $\delta_N(q_m, a) = q_j$. The DFA and PDA transitions for input symbol a combine to generate the transition $[[p_i, q_j], B] \in \delta([p_k, q_m], a, A)$ in

M′. Applying this transition to the result of the computation established by the inductive hypothesis produces

$$[[p_0, q_0], w, \lambda] \overset{n}{\underset{M}{\vdash}} [[p_k, q_m], v, \beta] \underset{M}{\vdash} [[p_i, q_j], u, \alpha].$$

The opposite inclusion, $L(M') \subseteq L(N) \cap L(M)$, is proved using induction on the length of computations in M′. The proof is left as an exercise. ■

Theorem 7.5.2 used DeMorgan's Law to show that the family of context-free languages is not closed under complementation. The next example gives a grammar that explicitly demonstrates this property.

Example 7.5.1

The language $L = \{ww \mid w \in \{a, b\}^*\}$ is not context-free, but $\overline{L}$ is. First we show that L is not context-free using a proof by contradiction. Assume L is context-free. Then, by Theorem 7.5.3,

$$L \cap a^*b^*a^*b^* = \{a^i b^j a^i b^j \mid i, j \geq 0\}$$

is context-free. However, this language was shown not to be context-free in Example 7.4.2, contradicting our assumption.

To show that $\overline{L}$ is context-free, we construct two context-free grammars G_1 and G_2 with $L(G_1) \cup L(G_2) = \overline{L}$.

G_1: $S \to aA \mid bA \mid a \mid b$
$A \to aS \mid bS$

G_2: $S \to AB \mid BA$
$A \to ZAZ \mid a$
$B \to ZBZ \mid b$
$Z \to a \mid b$

The grammar G_1 generates the strings of odd length over $\{a, b\}$, all of which are in $\overline{L}$. G_2 generates the set of even length string in $\overline{L}$. Such a string may be written $u_1xv_1u_2yv_2$, where $x, y \in \Sigma$ and $x \neq y$; $u_1, u_2, v_1, v_2 \in \Sigma^*$ with $length(u_1) = length(u_2)$ and $length(v_1) = length(v_2)$. That is, x and y are different symbols that occur in the same position in the substrings that make up the first half and the second half of $u_1xv_1u_2yv_2$. Since the u's and v's are arbitrary strings in Σ^*, this characterization can be rewritten u_1xpqyv_2, where $length(p) = length(u_1)$ and $length(q) = length(v_2)$. The recursive variables of G_2 generate precisely this set of strings. □

Exercises

1. Let M be the PDA defined by

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{A\}$
$F = \{q_1, q_2\}$

$\delta(q_0, a, \lambda) = \{[q_0, A]\}$
$\delta(q_0, \lambda, \lambda) = \{[q_1, \lambda]\}$
$\delta(q_0, b, A) = \{[q_2, \lambda]\}$
$\delta(q_1, \lambda, A) = \{[q_1, \lambda]\}$
$\delta(q_2, b, A) = \{[q_2, \lambda]\}$
$\delta(q_2, \lambda, A) = \{[q_2, \lambda]\}$.

   a) Describe the language accepted by M.
   b) Give the state diagram of M.
   c) Trace all computations of the strings *aab*, *abb*, *aba* in M.
   d) Show that *aabb*, *aaab* ∈ L(M).

2. Let M be the PDA in Example 7.1.3.
   a) Give the transition table of M.
   b) Trace all computations of the strings *ab*, *abb*, *abbb* in M.
   c) Show that *aaaa*, *baab* ∈ L(M).
   d) Show that *aaa*, *ab* ∉ L(M).

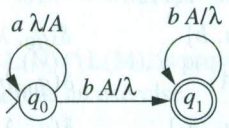
3. Construct PDAs that accept each of the following languages.
   a) $\{a^i b^j \mid 0 \leq i \leq j\}$
   b) $\{a^i c^j b^i \mid i, j \geq 0\}$
   c) $\{a^i b^j c^k \mid i + k = j\}$
   d) $\{w \mid w \in \{a, b\}^*$ and w has twice as many a's as b's$\}$
   e) $\{a^i b^i \mid i \geq 0\} \cup a^* \cup b^*$
   f) $\{a^i b^j c^k \mid i = j$ or $j = k\}$
   g) $\{a^i b^j \mid i \neq j\}$
   h) $\{a^i b^j \mid 0 \leq i \leq j \leq 2i\}$
   i) $\{a^{i+j} b^i c^j \mid i, j > 0\}$
   j) The set of palindromes over $\{a, b\}$

4. Construct a PDA with only two stack elements that accepts the language

$$\{wdw^R \mid w \in \{a, b, c\}^*\}.$$

5. Give the state diagram of a PDA M that accepts $\{a^{2i}b^{i+j} \mid 0 \le j \le i\}$ with acceptance by empty stack. Explain the role of the stack symbols in the computation of M. Trace the computations of M with input *aabb* and *aaaabb*.

6. The machine M



accepts the language $L = \{a^i b^i \mid i > 0\}$ by final state and empty stack.

a) Give the state diagram of a PDA that accepts L by empty stack.

b) Give the state diagram of a PDA that accepts L by final state.

7. Let L be the language $\{w \in \{a, b\}^* \mid w$ has a prefix containing more b's than a's$\}$. For example, *baa*, *abba*, *abbaaa* $\in$ L, but *aab*, *aabbab* $\notin$ L.

a) Construct a PDA that accepts L by final state.

b) Construct a PDA that accepts L by empty stack.

8. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA that accepts L by final state and empty stack. Prove that there is a PDA that accepts L by final state alone.

9. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA that accepts L by final state and empty stack. Prove that there is a PDA that accepts L by empty stack alone.

10. Let $L = \{a^{2i}b^i \mid i \ge 0\}$.

a) Construct a PDA M_1 with $L(M_1) = L$.

b) Construct an atomic PDA M_2 with $L(M_2) = L$.

c) Construct an extended PDA M_3 with $L(M_3) = L$ that has fewer transitions than M_1.

d) Trace the computation that accepts the string *aab* in each of the automata constructed in parts (a), (b), and (c).

11. Let $L = \{a^{2i}b^{3i} \mid i \ge 0\}$.

a) Construct a PDA M_1 with $L(M_1) = L$.

b) Construct an atomic PDA M_2 with $L(M_2) = L$.

c) Construct an extended PDA M_3 with $L(M_3) = L$ that has fewer transitions than M_1.

d) Trace the computation that accepts the string *aabbb* in each of the automata constructed in parts (a), (b), and (c).

12. Use the technique of Theorem 7.3.1 to construct a PDA that accepts the language of the Greibach normal form grammar

$$S \to aABA \mid aBB$$
$$A \to bA \mid b$$
$$B \to cB \mid c.$$

13. Let G be a grammar in Greibach normal form and M the PDA constructed from G. Prove that if $[q_0, u, \lambda] \vdash^* [q_1, \lambda, w]$ in M, then there is a derivation $S \overset{*}{\Rightarrow} uw$ in G. This completes the proof of Theorem 7.3.1.

14. Let M be the PDA

$$Q = \{q_0, q_1, q_2\} \qquad \delta(q_0, a, \lambda) = \{[q_0, A]\}$$
$$\Sigma = \{a, b\} \qquad \delta(q_0, b, A) = \{[q_1, \lambda]\}$$
$$\Gamma = \{A\} \qquad \delta(q_1, b, \lambda) = \{[q_2, \lambda]\}$$
$$F = \{q_2\} \qquad \delta(q_2, b, A) = \{[q_1, \lambda]\}.$$

a) Give the state diagram of M.

b) Give a set-theoretic definition of L(M).

c) Using the technique from Theorem 7.3.2, build a context-free grammar G that generates L(M).

d) Trace the computation of *aabbbb* in M.

e) Give the derivation of *aabbbb* in G.

15. Let M be the PDA in Example 7.1.1.

a) Trace the computation in M that accepts *bbcbb*.

b) Use the technique from Theorem 7.3.2 to construct a grammar G that accepts L(M).

c) Give the derivation of *bbcbb* in G.

*16. Theorem 7.3.2 presented a technique for constructing a grammar that generates the language accepted by an extended PDA. The transitions of the PDA pushed at most two variables onto the stack. Generalize this construction to build grammars from arbitrary extended PDAs. Prove that the resulting grammar generates the language of the PDA.

17. Use the pumping lemma to prove that each of the following languages is not context-free.

a) $\{a^k \mid k$ is a perfect square$\}$.

b) $\{a^i b^j c^i d^j \mid i, j \ge 0\}$

c) $\{a^i b^{2i} a^i \mid i \ge 0\}$

d) $\{a^i b^j c^k \mid 0 < i < j < k < zi\}$

e) $\{ww^R w \mid w \in \{a, b\}^*\}$

f) The set of finite-length prefixes of the infinite string

$$abaabaaabaaaab \ldots ba^n ba^{n+1} b \ldots .$$

18. a) Prove that the language $L_1 = \{a^i b^{2i} c^j \mid i, j \ge 0\}$ is context-free.

b) Prove that the language $L_2 = \{a^j b^i c^{2i} \mid i, j \ge 0\}$ is context-free.

c) Prove that $L_1 \cap L_2$ is not context-free.

19. a) Prove that the language $L_1 = \{a^i b^i c^j d^j \mid i, j \geq 0\}$ is context-free.
    b) Prove that the language $L_2 = \{a^j b^i c^i d^k \mid i, j, k \geq 0\}$ is context-free.
    c) Prove that $L_1 \cap L_2$ is not context-free.

20. Let L be the language consisting of all strings over $\{a, b\}$ with the same number of a's and b's. Show that the pumping lemma is satisfied for L. That is, show that every string z of length k or more has a decomposition that satisfies the conditions of the pumping lemma.

21. Let M be a PDA. Prove that there is a decision procedure to determine whether
    a) L(M) is empty.
    b) L(M) is finite.
    c) L(M) is infinite.

*22. A grammar $G = (V, \Sigma, P, S)$ is called **linear** if every rule has the form

$$A \to u$$
$$A \to uBv$$

where $u, v \in \Sigma^*$ and $A, B \in V$. A language is called linear if it is generated by a linear grammar. Prove the following pumping lemma for linear languages.

Let L be a linear language. Then there is a constant k such that for all $z \in L$ with $length(z) \geq k$, z can be written $z = uvwxy$ with

i) $length(uvxy) \leq k$,
ii) $length(vx) > 0$, and
iii) $uv^i wx^i y \in L$, for $i \geq 0$.

23. a) Construct a DFA N that accepts all strings in $\{a, b\}^*$ with an odd number of a's.
    b) Construct a PDA M that accepts $\{a^{3i} b^i \mid i \geq 0\}$.
    c) Use the technique from Theorem 7.5.3 to construct a PDA M′ that accepts $L(N) \cap L(M)$.
    d) Trace the computations that accept $aaab$ in N, M, and M′.

24. Let $G = (V, \Sigma, P, S)$ be a context-free grammar. Define an extended PDA M as follows:

$$Q = \{q_0\} \qquad \delta(q_0, \lambda, \lambda) = \{[q_0, S]\}$$
$$\Sigma = \Sigma_G \qquad \delta(q_0, \lambda, A) = \{[q_0, w] \mid A \to w \in P\}$$
$$\Gamma = \Sigma_G \cup V \qquad \delta(q_0, a, a) = \{[q_0, \lambda] \mid a \in \Sigma\}.$$
$$F = \{q_0\}$$

Prove that $L(M) = L(G)$.

25. Complete the proof of Theorem 7.5.3.

26. Prove that the set of context-free languages is closed under reversal.

*27. Let L be a context-free language over Σ and $a \in \Sigma$. Define $er_a(L)$ to be the set obtained by removing all occurrences of a from strings of L. The language $er_a(L)$ is the language L with a erased. For example, if $abab$, $bacb$, $aa \in L$, then bb, bcb, and $\lambda \in er_a(L)$. Prove that $er_a(L)$ is context-free. *Hint:* Convert the grammar that generates L to one that generates $er_a(L)$.

*28. The notion of a string homomorphism was introduced in Exercise 6.19. Let L be a context-free language over Σ and let $h : \Sigma^* \to \Sigma^*$ be a homomorphism.
    a) Prove that $h(L) = \{h(w) \mid w \in L\}$ is context-free, that is, that the context-free languages are closed under homomorphisms.
    b) Use the result of part (a) to show that $er_a(L)$ is context-free.
    c) Give an example to show that the homomorphic image of a noncontext-free language may be context-free.

29. Let $h : \Sigma^* \to \Sigma^*$ be a homomorphism and L a context-free language over Σ. Prove that $\{w \mid h(w) \in L\}$ is context-free. In other words, the family of context-free languages is closed under inverse homomorphic images.

30. Use closure under homomorphic images and inverse images to show that the following languages are not context-free.
    a) $\{a^i b^j c^i d^j \mid i, j \geq 0\}$
    b) $\{a^i b^{2i} c^{3i} \mid i \geq 0\}$
    c) $\{(ab)^i (bc)^i (ca)^i \mid i \geq 0\}$

Bibliographic Notes

Pushdown automata were introduced in Oettinger [1961]. Deterministic pushdown automata were studied in Fischer [1963] and Schutzenberger [1963] and their acceptance of the languages generated by LR(k) grammars is from Knuth [1965]. The relationship between context-free languages and pushdown automata was discovered by Chomsky [1962], Evey [1963], and Schutzenberger [1963]. The closure properties for context-free languages presented in Section 7.5 are from Bar-Hillel, Perles, and Shamir [1961] and Scheinberg [1960]. A solution to Exercises 28 and 29 can be found in Ginsburg and Rose [1963b].

The pumping lemma for context-free languages is from Bar-Hillel, Perles, and Shamir [1961]. A stronger version of the pumping lemma is given in Ogden [1968]. Parikh's Theorem [1966] provides another tool for establishing that languages are not context-free.