CHAPTER 10

The Chomsky Hierarchy

In Chapter 3, regular and context-free grammars were introduced as rule-based systems for generating the strings of language. A rule defines a string transformation, and a sentence of the language is obtained by a sequence of permissible transformations. The regular and context-free grammars are subsets of the more general class of phrase-structure grammars. Phrase-structure grammars were proposed as syntactic models of natural language by Noam Chomksy. In this chapter we will consider two additional families of phrase-structure grammars, unrestricted grammars and context-sensitive grammars. The four families of grammars, regular, context-free, context-sensitive, and unrestricted, make up the Chomsky hierarchy of phrase-structure grammars, with each successive family in the hierarchy permitting additional flexibility in the definition of a rule.

Automata were designed to mechanically recognize regular and context-free languages; deterministic finite automata accept the languages generated by regular grammars and pushdown automata accept the languages generated by context-free grammars. The relationship between grammatical generation and mechanical acceptance is extended to the new families of grammars. Turing machines are shown to accept the languages generated by unrestricted grammars. A class of machines obtained by limiting the memory available to a Turing machine accepts the languages generated by context-sensitive grammars.

10.1 Unrestricted Grammars

Phrase-structure grammars were designed to provide formal models of the syntax of natural language. The name, phrase-structure, is based on the proposition that the sentences of

language may have several different syntactic patterns. The sentences themselves are made up of phrases: noun phrases, verb phrases, and the like, that are arranged as specified by one of the sentence patterns. The rules of the grammar define the structure of both the sentences and the phrases.

The components of a phrase-structure grammar are the same as those of the regular and context-free grammars studied in Chapter 3. A phrase-structure grammar consists of a finite set V of variables, an alphabet Σ, a start variable, and a set of rules. A rule has the form $u \to v$, where u and v can be any combination of variables and terminals, and defines a permissible string transformation. The application of a rule to a string z is a two-step process that consists of

i) matching the left-hand side of the rule to a substring of z, and
ii) replacing the left-hand side with the right-hand side.

The application of the rule $u \to v$ to the string xuy, written $xuy \Rightarrow xvy$, produces the string xvy. A string q is derivable from p, $p \overset{*}{\Rightarrow} q$, if there is a sequence of rule applications that transforms p to q. The **language** of G, denoted L(G), is the set of terminal strings derivable from the start symbol S. Symbolically, $\mathrm{L(G)} = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$.

A family of grammars is defined by the restrictions placed on the form of the rules. A context-free grammar is a phrase-structure grammar in which the left-hand side of every rule is a single variable. The right-hand side can be any combination of variables and terminals. Each rule of a regular grammar is required to have one of the following forms:

i) $A \to aB$,
ii) $A \to a$, or
iii) $A \to \lambda$,

where $A, B \in V$, and $a \in \Sigma$.

The unrestricted grammars are the largest class of phrase-structure grammars. There are no constraints on a rule other than requiring that the left-hand side must not be null.

Definition 10.1.1

An **unrestricted grammar** is a quadruple (V, Σ, P, S), where V is a finite set of variables; Σ (the alphabet) is a finite set of terminal symbols; P is a set of rules; and S is a distinguished element of V. A production of an unrestricted grammar has the form $u \to v$, where $u \in (\mathrm{V} \cup \Sigma)^+$ and $v \in (\mathrm{V} \cup \Sigma)^*$. The sets V and Σ are assumed to be disjoint.

Two examples are given that illustrate the generative power of unrestricted grammars. Example 10.1.1 shows that the language $\{a^i b^i c^i \mid i \geq 0\}$, which we know is not derivable by any context-free grammar, can be generated by an unrestricted grammar with six rules. The second example shows how unrestricted rules can be used to generate copies of a string.

Example 10.1.1

The unrestricted grammar

$$\begin{aligned} \mathrm{V} &= \{S, A, C\} & S &\to aAbc \mid \lambda \\ \Sigma &= \{a, b, c\} & A &\to aAbC \mid \lambda \\ & & Cb &\to bC \\ & & Cc &\to cc \end{aligned}$$

with start symbol S generates the language $\{a^i b^i c^i \mid i \geq 0\}$. The string $a^i b^i c^i$, $i > 0$, is generated by a derivation that begins

$$\begin{aligned} S &\Longrightarrow aAbc \\ &\overset{i-1}{\Longrightarrow} a^i A(bC)^{i-1}bc \\ &\Longrightarrow a^i (bC)^{i-1}bc, \end{aligned}$$

using the rule $A \to aABC$ to generate the i leading a's. The rule $Cb \to bC$ allows the final C to pass through the b's that separate it from the c's at the end of the string. Upon reaching the leftmost c, the variable C is replaced with c. This process is repeated until each occurrence of the variable C is moved to the right of all the b's and transformed into a c. □

Example 10.1.2

The unrestricted grammar with terminal alphabet $\{a, b, [,]\}$ defined by the productions

$$\begin{aligned} S &\to aT[a] \mid bT[b] \mid [\,] \\ T[&\to aT[A \mid bT[B \mid [\\ Aa &\to aA \\ Ab &\to bA \\ Ba &\to aB \\ Bb &\to bB \\ A] &\to a] \\ B] &\to b] \end{aligned}$$

generates the language $\{u[u] \mid u \in \{a, b\}^*\}$.

The addition of an a or b to the left of the variable T is accompanied by the generation of the variable A or B after $T[$. Using the rules that interchange the position of a variable and a terminal, the derivation progresses by passing the variable through the copy of the string enclosed in the brackets. When the variable is adjacent to the symbol], the appropriate terminal is added to the second string. The entire process may be repeated to generate

additional terminal symbols or be terminated by the application of the rule $T[\rightarrow [$. The derivation

$$\begin{aligned} S &\Rightarrow aT[a] \\ &\Rightarrow aaT[Aa] \\ &\Rightarrow aaT[aA] \\ &\Rightarrow aaT[aa] \\ &\Rightarrow aabT[Baa] \\ &\Rightarrow aabT[aBa] \\ &\Rightarrow aabT[aaB] \\ &\Rightarrow aabT[aab] \\ &\Rightarrow aab[aab] \end{aligned}$$

exhibits the roles of the variables in a derivation. □

In the grammars in the two preceding examples, the left-hand side of each rule contained a variable. This is not required by the definition of unrestricted grammar. However, the imposition of the restriction that the left-hand side of a rule contain a variable does not reduce the set of languages that can be generated (Exercise 3).

Throughout our study of formal languages, we have demonstrated a correspondence between the generation of a language by a grammar and its acceptance by a finite-state machine. Regular languages are accepted by finite automata and context-free languages by pushdown automata. Unrestricted grammars provide the most flexible type of string transformation; there are no conditions on the matching substring, nor on the replacement. It would seem reasonable that generation by a unrestricted grammar corresponds to acceptance by the most powerful type of abstract machine. This is indeed the case. The next two theorems show that a language is generated by an unrestricted grammar if, and only if, it is accepted by a Turing machine.

Theorem 10.1.2

Let G = (V, Σ, P, S) be an unrestricted grammar. Then L(G) is a recursively enumerable language.

Proof. We will sketch the design of a three-tape nondeterministic Turing machine M that accepts L(G). We will design M so that its computations simulate derivations of the grammar G. Tape 1 holds an input string p from Σ^*. A representation of the rules of G is written on tape 2. A rule $u \rightarrow v$ is represented by the string $u\#v$, where # is a tape symbol reserved for this purpose. Rules are separated by two consecutive #'s. The derivations of G are simulated on tape 3.

A computation of the machine M that accepts L(G) consists of the following actions:

1. S is written on position one of tape 3.
2. The rules of G are written on tape 2.

3. A rule $u\#v$ is chosen from tape 2.
4. An instance of the string u is chosen on tape 3, if one exists. Otherwise, the computation halts in a rejecting state.
5. The string u is replaced by v on tape 3.
6. If the strings on tape 3 and tape 1 match, the computation halts in an accepting state.
7. The computation continues with step 3 to simulate another rule application.

Since the length of u and v may differ, the simulation of a rule application $xuy \Rightarrow xvy$ may require shifting the position of the string y.

For any string $p \in$ L(G), there is a sequence of rule applications that derives p. This derivation will be examined by one of the nondeterministic computations of the machine M, and M will accept p. Conversely, the actions of M on tape 3 generate precisely the strings derivable from S in G. The only strings accepted by M are terminal strings in L(G). Thus, L(M) = L(G). ■

Example 10.1.3

The language L = $\{a^i b^i c^i \mid i \geq 0\}$ is generated by the rules

$$\begin{aligned} S &\rightarrow aAbc \mid \lambda \\ A &\rightarrow aAbC \mid \lambda \\ Cb &\rightarrow bC \\ Cc &\rightarrow cc. \end{aligned}$$

Computations of the machine that accepts L simulate derivations of the grammar. The rules of the grammar are represented on tape 2 by

$$BS\#aAbc\#\#S\#\#\#A\#aAbC\#\#A\#\#\#Cb\#bC\#\#Cc\#ccB.$$

The rule $S \rightarrow \lambda$ is represented by the string $S\#\#\#$. The first # separates the left-hand side of the rule from the right-hand side. The right-hand side of the rule, the null string in this case, is followed by the string ##. □

Theorem 10.1.3

Let L be a recursively enumerable language. Then there is an unrestricted grammar G with L(G) = L.

Proof. Since L is recursively enumerable, it is accepted by a deterministic Turing machine M = (Q, Σ, Γ, δ, q_0, F). An unrestricted grammar G = (V, Σ, P, S) is designed whose derivations simulate the computations of M. Using the representation of a Turing machine configuration as a string, the effect of a Turing machine transition $\delta(q_i, x) = [q_j, y, R]$ on the configuration uq_ixvB can be represented by the string transformation $uq_ixvB \Rightarrow uyq_jvB$.

The derivation of a terminal string in G consists of three distinct subderivations:

i) the generation of a string $u[q_0Bu]$ where $u \in \Sigma^*$,

ii) the simulation of a computation of M on the string $[q_0Bu]$, and

iii) if M accepts u, the removal of the simulation substring.

The grammar G contains a variable A_i for each terminal symbol $a_i \in \Sigma$. These variables, along with S, T, [, and], are used in the generation of the string $u[q_0Bu]$. The simulation of a computation uses variables corresponding to the states of M. The variables E_R and E_L are used in the third phase of a derivation. The terminal symbols of the grammar are the elements of the input alphabet of M. Thus the alphabets of G are

$$\Sigma = \{a_1, a_2, \ldots, a_n\}$$
$$V = \{S, T, E_R, E_L, [,], A_1, A_2, \ldots, A_n\} \cup Q.$$

The rules for each of the three parts of a derivation are given separately. A derivation begins by generating $u[q_0Bu]$, where u is an arbitrary string in Σ^*. The strategy used for generating strings of this form was presented in Example 10.1.2.

1. $S \rightarrow a_iT[a_i] \mid [q_0B]$ for $1 \leq i \leq n$
2. $T[\rightarrow a_iT[A_i \mid [q_0B$ for $i \leq i \leq n$
3. $A_ia_j \rightarrow a_jA_i$ for $1 \leq i, j \leq n$
4. $A_i] \rightarrow a_i]$ for $1 \leq i \leq n$

The computation of the Turing machine with input u is simulated on the string $[q_0Bu]$. The rules are obtained by rewriting the transitions of M as string transformations.

5. $q_ixy \rightarrow zq_jy$ whenever $\delta(q_i, x) = [q_j, z, R]$ and $y \in \Gamma$
6. $q_ix] \rightarrow zq_jB]$ whenever $\delta(q_i, x) = [q_j, z, R]$
7. $yq_ix \rightarrow q_jyz$ whenever $\delta(q_i, x) = [q_j, z, L]$ and $y \in \Gamma$

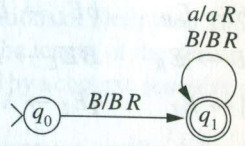
If the computation of M halts in an accepting state, the derivation erases the string within the brackets. The variable E_R erases the string to the right of the halting position of the tape head. Upon reaching the endmarker], the variable E_L (erase left) is generated.

8. $q_ix \rightarrow E_R$ whenever $\delta(q_i, x)$ is undefined and $q_i \in F$
9. $E_Rx \rightarrow E_R$ for $x \in \Gamma$
10. $E_R] \rightarrow E_L$
11. $xE_L \rightarrow E_L$ for $x \in \Gamma$
12. $[E_L \rightarrow \lambda$

The derivation that begins by generating $u[q_0Bu]$ terminates with u whenever $u \in L(M)$. If $u \notin L(M)$, the brackets enclosing the simulation of the computation are never erased and the derivation does not produce a terminal string. ■

Example 10.1.4

The construction of a grammar that generates the language accepted by a Turing machine is demonstrated using the machine M



that accepts $a^*b(a \cup b)^*$. When the first b is encountered, M halts and accepts in state q_1.

The variables and terminals of G are

$$\Sigma = \{a, b\}$$
$$V = \{S, T, E_R, E_L, [,], A, X\} \cup \{q_0, q_1\}.$$

The rules are given in three sets.

Input-generating rules:

$$S \rightarrow aT[a] \mid bT[b] \mid [q_0B]$$
$$T[\rightarrow aT[A \mid bT[X \mid [q_0B$$
$$Aa \rightarrow aA$$
$$Ab \rightarrow bA$$
$$A] \rightarrow a]$$
$$Xa \rightarrow aX$$
$$Xb \rightarrow bX$$
$$X] \rightarrow b]$$

Simulation rules:

Transition	Rules
$\delta(q_0, B) = [q_1, B, R]$	$q_0Ba \rightarrow Bq_1a$
	$q_0Bb \rightarrow Bq_1b$
	$q_0BB \rightarrow Bq_1B$
	$q_0B] \rightarrow Bq_1B]$
$\delta(q_1, a) = [q_1, a, R]$	$q_1aa \rightarrow aq_1a$
	$q_1ab \rightarrow aq_1b$
	$q_1aB \rightarrow aq_1B$
	$q_1a] \rightarrow aq_1B]$
$\delta(q_1, B) = [q_1, B, R]$	$q_1Ba \rightarrow Bq_1a$
	$q_1Bb \rightarrow Bq_1b$
	$q_1BB \rightarrow Bq_1B$
	$q_1B] \rightarrow Bq_1B]$

Erasure rules:

$$q_1 b \to E_R$$
$$E_R a \to E_R \qquad aE_L \to E_L$$
$$E_R b \to E_R \qquad bE_L \to E_L$$
$$E_R B \to E_R \qquad BE_L \to E_L$$
$$E_R] \to E_L \qquad [E_L \to \lambda$$

The computation that accepts the string ab in M and the corresponding derivation in the grammar G that accepts ab are

$$\begin{aligned} & q_0 BabB && S \Rightarrow aT[a] \\ \vdash\ & Bq_1 abB && \Rightarrow abT[Xa] \\ \vdash\ & Baq_1 bB && \Rightarrow ab[q_0 BXa] \\ & && \Rightarrow ab[q_0 BaX] \\ & && \Rightarrow ab[q_0 Bab] \\ & && \Rightarrow ab[Bq_1 ab] \\ & && \Rightarrow ab[Baq_1 b] \\ & && \Rightarrow ab[BaE_R] \\ & && \Rightarrow ab[BaE_L \\ & && \Rightarrow ab[BE_L \\ & && \Rightarrow ab[E_L \\ & && \Rightarrow ab. \end{aligned}$$

□

Properties of unrestricted grammars can be used to establish closure results for recursively enumerable languages. The proofs, similar to those presented in Theorem 7.5.1 for context-free languages, are left as exercises.

Theorem 10.1.4

The set of recursively enumerable languages is closed under union, concatenation, and Kleene star.

10.2 Context-Sensitive Grammars

The context-sensitive grammars represent an intermediate step between the context-free and the unrestricted grammars. No restrictions are placed on the left-hand side of a production, but the length of the right-hand side is required to be at least that of the left.

Definition 10.2.1

A phrase-structure grammar $G = (V, \Sigma, P, S)$ is called **context-sensitive** if each rule has the form $u \to v$, where $u \in (V \cup \Sigma)^+$, $v \in (V \cup \Sigma)^+$, and $length(u) \leq length(v)$.

A rule that satisfies the conditions of Definition 10.2.1 is called *monotonic*. With each application of a monotonic rule, the length of the derived string either remains the same or increases. The language generated by a context-sensitive grammar is called, not surprisingly, a context-sensitive language.

Context-sensitive grammars were originally defined as phrase-structure grammars in which each rule has the form $uAv \to uwv$, where $A \in V$, $w \in (V \cup \Sigma)^+$, and $u, v \in (V \cup \Sigma)^*$. The preceding rule indicates that the variable A can be replaced by w only when it appears in the context of being preceded by u and followed by v. Clearly, every rule defined in this manner is monotonic. On the other hand, a transformation defined by a monotonic rule can be generated by a set of rules of the form $uAv \to uwv$ (Exercises 10 and 11).

The monotonic property of the rules guarantees that the null string is not an element of a context-sensitive language. Removing the rule $S \to \lambda$ from the grammar in Example 10.1.1, we obtain the unrestricted grammar

$$S \to aAbc$$
$$A \to aAbC \mid \lambda$$
$$Cb \to bC$$
$$Cc \to cc$$

that generates the language $\{a^i b^i c^i \mid i > 0\}$. The λ-rule violates the monotonicity property of context-sensitive rules. Replacing the S and A rules with

$$S \to aAbc \mid abc$$
$$A \to aAbC \mid abC$$

produces an equivalent context-sensitive grammar.

A nondeterministic Turing machine, similar to the machine in Theorem 10.1.2, is designed to accept a context-sensitive language. The noncontracting nature of the rules permits the length of the input string to be used to terminate the simulation of an unsuccessful derivation. When the length of the derived string surpasses that of the input, the computation halts and rejects the string.

Theorem 10.2.2

Every context-sensitive language is recursive.

Proof. Following the approach developed in Theorem 10.1.2, derivations of the context-sensitive grammar are simulated on a three-tape nondeterministic Turing machine M. The entire derivation, rather than just the result, is recorded on tape 3. When a rule $u \to v$ is

applied to the string xuy on tape 3, the string xvy is written on the tape following $xuy\#$. The symbol # is used to separate the derived strings.

A computation of M with input string p performs the following sequence of actions:

1. $S\#$ is written beginning at position one of tape 3.
2. The rules of G are written on tape 2.
3. A rule $u\#v$ is chosen from tape 2.
4. Let $q\#$ be the most recent string written on tape 3:
   a) An instance of the string u in q is chosen, if one exists. In this case, q can be written xuy.
   b) Otherwise, the computation halts in a nonaccepting state.
5. $xvy\#$ is written on tape 3 immediately following $q\#$.
6. a) If $xvy = p$, the computation halts in an accepting state.
   b) If xvy occurs at another position on tape 3, the computation halts in a nonaccepting state.
   c) If $length(xvy) > length(p)$, the computation halts in a nonaccepting state.
7. The computation continues with step 3 to simulate another rule application.

There are only a finite number of strings in $(V \cup \Sigma)^*$ with length less than or equal to $length(p)$. This implies that every derivation eventually halts, enters a cycle, or derives a string of length greater than $length(p)$. A computation halts at step 4 when the rule that has been selected cannot be applied to the current string. Cyclic derivations, $S \overset{*}{\Rightarrow} w \overset{+}{\Rightarrow} w$, are terminated in step 6(b). The length bound is used in step 6(c) to terminate all other unsuccessful derivations.

Every string in L(G) is generated by a noncyclic derivation. The simulation of such a derivation causes M to accept the string. Since every computation of M halts, L(G) is recursive (Exercise 8.23). ■

10.3 Linear-Bounded Automata

We have examined several modifications of the standard Turing machine that do not alter the set of languages accepted by the machines. Restricting the amount of the tape decreases the capabilities of a Turing machine computation. A linear-bounded automaton is a Turing machine in which the amount of available tape is determined by the length of the input string. The input alphabet contains two symbols, ⟨ and ⟩, that designate the left and right boundaries of the tape.

Definition 10.3.1

A **linear-bounded automaton** (LBA) is a structure $M = (Q, \Sigma, \Gamma, \delta, q_0, \langle, \rangle, F)$, where Q, Σ, Γ, δ, q_0, and F are the same as for a nondeterministic Turing machine. The symbols ⟨ and ⟩ are distinguished elements of Σ.

The initial configuration of a computation is $q_0\langle w\rangle$, requiring $length(w) + 2$ tape positions. The endmarkers ⟨ and ⟩ are written on the tape but not considered part of the input. A computation remains within the boundaries specified by ⟨ and ⟩. The endmarkers may be read by the machine but cannot be erased. Transitions scanning ⟨ must designate a move to the right and those reading ⟩ a move to the left. A string $w \in (\Sigma - \{\langle, \rangle\})^*$ is accepted by an LBA if a computation with input $\langle w\rangle$ halts in an accepting state.

We will show that every context-sensitive language is accepted by a linear-bounded automaton. An LBA is constructed to simulate the derivations of the context-sensitive grammar. The Turing machine constructed to simulate the derivations of an unrestricted grammar begins by writing the rules of the grammar on one of the tapes. The restriction on the amount of tape available to an LBA prohibits this approach. Instead, states and transitions of the LBA are used to encode the rules.

The diagram in Figure 10.1 shows how transitions can simulate the application of the rule $Sa \rightarrow aAS$. The application of the rule generates a string transformation $uSav \Rightarrow uaASv$. The first two transitions in the diagram verify that the string on the tape beginning at the position of the tape head matches Sa. Before Sa is replaced with aAS, the string v is traversed to determine whether the derived string fits on the segment of the tape available to the computation. If the ⟩ is read, the computation terminates. Otherwise, the string v is shifted one position to the right and Sa is replaced by aAS.

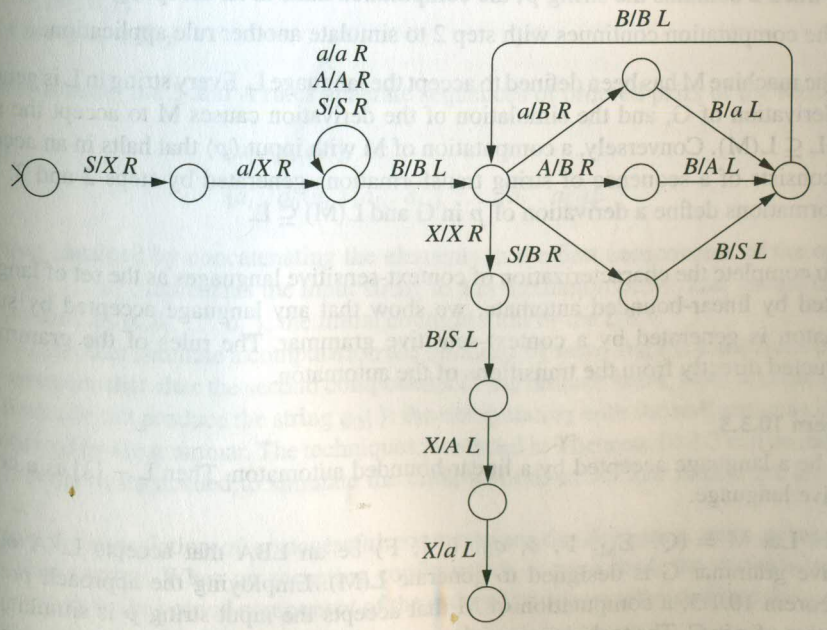


FIGURE 10.1 LBA simulation of application of $Sa \rightarrow aAs$.

Theorem 10.3.2

Let L be a context-sensitive language. Then there is a linear-bounded automaton M with $L(M) = L$.

Proof. Since L is a context-sensitive language, $L = L(G)$ for some context-sensitive grammar $G = (V, \Sigma, P, S)$. An LBA M with a two-track tape is constructed to simulate the derivations of G. The first track contains the input, including the endmarkers. The second track holds the string generated by the simulated derivation.

Each rule of G is encoded in a submachine of M. A computation of M with input $\langle p \rangle$ consists of the following sequence of actions:

1. S is written on track 2 in position one.
2. The tape head is moved into a position in which it scans a symbol of the string on track 2.
3. A rule $u \rightarrow v$ is nondeterministically selected, and the computation attempts to apply the rule.
4. a) If a substring on track 2 beginning at the position of the tape head does not match u, the computation halts in a nonaccepting state.
   b) If the tape head is scanning u but the string obtained by replacing u by v is greater than $length(p)$, then the computation halts in a nonaccepting state.
   c) Otherwise, u is replaced by v on track 2.
5. If track 2 contains the string p, the computation halts in an accepting state.
6. The computation continues with step 2 to simulate another rule application.

The machine M has been defined to accept the language L. Every string in L is generated by a derivation of G, and the simulation of the derivation causes M to accept the string. Thus, $L \subseteq L(M)$. Conversely, a computation of M with input $\langle p \rangle$ that halts in an accepting state consists of a sequence of string transformations generated by steps 2 and 3. These transformations define a derivation of p in G and $L(M) \subseteq L$. ∎

To complete the characterization of context-sensitive languages as the set of languages accepted by linear-bounded automata, we show that any language accepted by such an automaton is generated by a context-sensitive grammar. The rules of the grammar are constructed directly from the transitions of the automaton.

Theorem 10.3.3

Let L be a language accepted by a linear-bounded automaton. Then $L - \{\lambda\}$ is a context-sensitive language.

Proof. Let $M = (Q, \Sigma_M, \Gamma, \delta, q_0, \langle, \rangle, F)$ be an LBA that accepts L. A context-sensitive grammar G is designed to generate L(M). Employing the approach presented in Theorem 10.1.3, a computation of M that accepts the input string p is simulated by a derivation of p in G. The techniques used to construct an unrestricted grammar that simulates a Turing machine computation cannot be employed since the rules that erase the simulation do not satisfy the monotonicity restrictions of a context-sensitive grammar. The inability to erase symbols in the derivation of a context-sensitive grammar restricts the length of a derived string to that of the input. The simulation is accomplished by using composite objects as variables.

The terminal alphabet of G is obtained from the input alphabet of M by deleting the endmarkers. Ordered pairs are used as variables. The first component of an ordered pair is a terminal symbol. The second is a string consisting of a combination of a tape symbol and possibly a state and endmarker(s).

$$\Sigma_G = \Sigma_M - \{\langle, \rangle\} = \{a_i, a_2, \ldots, a_n\}$$

$$V = \{S, A, [a_i, x], [a_i, \langle x], [a_i, x\rangle], [a_i, \langle x\rangle], [a_i, q_k x], [a_i, q_k\langle x], [a_i, \langle q_k x], [a_i, q_k x\rangle], [a_i, x q_k\rangle], [a_i, q_k\langle x\rangle], [a_i, \langle q_k x\rangle], [a_i, \langle x q_k\rangle]\},$$

where $a_i \in \Sigma_G$, $x \in \Gamma$, and $q_k \in Q$.

The S and A rules generate ordered pairs whose components represent the input string and the initial configuration of a computation of M.

1. $S \rightarrow [a_i, q_0\langle a_i]A$
   $\rightarrow [a_i, q_0\langle a_i\rangle]$
   for every $a_i \in \Sigma_G$
2. $A \rightarrow [a_i, a_i]A$
   $\rightarrow [a_i, a_i\rangle]$
   for every $a_i \in \Sigma_G$

Derivations using the S and A rules generate sequences of ordered pairs of the form

$$[a_i, q_0\langle a_i\rangle], \text{ or}$$

$$[a_{i_1}, q_0\langle a_{i_1}][a_{i_2}, a_{i_2}] \ldots [a_{i_n}, a_{i_n}\rangle].$$

The string obtained by concatenating the elements in the first components of the ordered pairs, $a_{i_1}a_{i_2} \ldots a_{i_n}$, represents the input string to a computation of M. The second components produce $q_0\langle a_{i_1}a_{i_2} \ldots a_{i_n}\rangle$, the initial configuration of the LBA.

The rules that simulate a computation are obtained by rewriting the transitions of M as transformations that alter the second components of the ordered pairs. Note that the second components do not produce the string $q_0\langle\,\rangle$; the computation with the null string as input is not simulated by the grammar. The techniques presented in Theorem 10.1.3 can be modified to produce the rules needed to simulate the computations of M. The details are left as an exercise.

Upon the completion of a successful computation, the derivation must generate the original input string. When an accepting configuration is generated, the variable with the accepting state in the second component of the ordered pair is transformed into the terminal symbol contained in the first component.

3. $[a_i, q_k\langle x] \rightarrow a_i$
$[a_i, q_k\langle x\rangle] \rightarrow a_i$
whenever $\delta(q_k, \langle) = \emptyset$ and $q_k \in F$

$[a_i, xq_k\rangle] \rightarrow a_i$
$[a_i, \langle xq_k\rangle] \rightarrow a_i$
whenever $\delta(q_k, \rangle) = \emptyset$ and $q_k \in F$

$[a_i, q_k x] \rightarrow a_i$
$[a_i, q_k x\rangle] \rightarrow a_i$
$[a_i, \langle q_k x] \rightarrow a_i$
$[a_i, \langle q_k x\rangle] \rightarrow a_i$
whenever $\delta(q_k, x) = \emptyset$ and $q_k \in F$

The derivation is completed by transforming the remaining variables to the terminal contained in the first component.

4. $[a_i, u]a_j \rightarrow a_i a_j$
$a_j[a_i, u] \rightarrow a_j a_i$
for every $a_j \in \Sigma_G$ and $[a_i, u] \in V$ ■

10.4 The Chomsky Hierarchy

Chomsky numbered the four families of grammars (and languages) that make up the hierarchy. Unrestricted, context-sensitive, context-free, and regular grammars are referred to as type 0, type 1, type 2, and type 3 grammars, respectively. The restrictions placed on the rules increase with the number of the grammar. The nesting of the families of grammars of the Chomsky hierarchy induces a nesting of the corresponding languages. Every context-free language containing the null string is generated by a context-free grammar in which $S \rightarrow \lambda$ is the only λ-rule (Theorem 4.2.3). Removing this single λ-rule produces a context-sensitive grammar that generates $L - \{\lambda\}$. Thus, the language $L - \{\lambda\}$ is context-sensitive whenever L is context-free. Ignoring the complications presented by the null string in context-sensitive languages, every type i language is also type $(i - 1)$.

The preceding inclusions are proper. The set $\{a^i b^i \mid i \geq 0\}$ is context-free but not regular (Theorem 6.5.1). Similarly, $\{a^i b^i c^i \mid i > 0\}$ is context-sensitive but not context-free (Example 7.4.1). In Chapter 11, the language of the Halting Problem is shown to be recursively enumerable but not recursive. Combining this result with Theorem 10.2.2 establishes the proper inclusion of context-sensitive languages in the set of recursively enumerable languages.

Each class of languages in the Chomsky hierarchy has been characterized as the languages generated by a family of grammars and accepted by a type of machine. The relationships developed between generation and recognition are summarized in the following table.

Grammars	Languages	Accepting Machines
Type 0 grammars, phrase-structure grammars, unrestricted grammars	Recursively enumerable	Turing machine, nondeterministic Turing machine
Type 1 grammars, context-sensitive grammars,	Context-sensitive	Linear-bounded automata
Type 2 grammars, context-free grammars	Context-free	Pushdown automata
Type 3 grammars, regular grammars, left-linear grammars, right-linear grammars	Regular	Deterministic finite automata, nondeterministic finite automata

Exercises

1. Design unrestricted grammars to generate the following languages:
   a) $\{a^i b^j a^i b^j \mid i, j \geq 0\}$
   b) $\{a^i b^i c^i d^i \mid i \geq 0\}$
   c) $\{www \mid w \in \{a, b\}^*\}$

2. Prove that every terminal string generated by the grammar

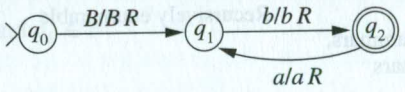
$$S \rightarrow aAbc \mid \lambda$$
$$A \rightarrow aAbC \mid \lambda$$
$$Cb \rightarrow bC$$
$$Cc \rightarrow cc$$

has the form $a^i b^i c^i$ for some $i \geq 0$.

*3. Prove that every recursively enumerable language is generated by a grammar in which each rule has the form $u \rightarrow v$ where $u \in V^+$ and $v \in (V \cup \Sigma)^*$.

4. Prove that the recursively enumerable languages are closed under the following operations:
   a) union
   b) intersection
   c) concatenation
   d) Kleene star
   e) homomorphic images

5. Let M be the Turing machine



a) Give a regular expression for L(M).

b) Using the techniques from Theorem 10.1.3, give the rules of an unrestricted grammar G that accepts L(M).

c) Trace the computation of M when run with input *bab* and give the corresponding derivation in G.

6. Let G be the context-sensitive grammar

$$\begin{aligned} \text{G:}\quad & S \to SBA \mid a \\ & BA \to AB \\ & aA \to aaB \\ & B \to b. \end{aligned}$$

a) Give a derivation of *aabb*.

b) What is L(G)?

c) Construct a context-free grammar that generates L(G).

7. Let L be the language $\{a^i b^{2i} a^i \mid i > 0\}$.

a) Use the pumping lemma for context-free languages to show that L is not context-free.

b) Construct a context-sensitive grammar G that generates L.

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

d) Construct an LBA M that accepts L.

e) Trace the computation of M with input *aabbbbaa*.

* 8. Let $L = \{a^i b^j c^k \mid 0 < i \leq j \leq k\}$.

a) L is not context-free. Can this be proved using the pumping lemma for context-free languages? If so, do so. If not, show that the pumping lemma is incapable of establishing that L is not context-free.

b) Give a context-sensitive grammar that generates L.

9. Let M be an LBA with alphabet Σ. Outline a general approach to construct monotonic rules that simulate the computation of M. The rules of the grammar should consist of variables in the set

$$\{[a_i, x],\ [a_i, \langle x],\ [a_i, x\rangle],\ [a_i, \langle x\rangle],\ [a_i, q_k x],\ [a_i, q_k\langle x],\ [a_i, \langle q_k x],\ [a_i, q_k x\rangle],$$
$$[a_i, xq_k\rangle],\ [a_i, q_k\langle x\rangle],\ [a_i, \langle q_k x\rangle],\ [a_i, \langle xq_k\rangle]\},$$

where $a_i \in \Sigma$, $x \in \Gamma$, and $q_i \in Q$. This completes the construction of the grammar in Theorem 10.3.3.

* 10. Let $u \to v$ be a monotonic rule. Construct a sequence of monotonic rules, each of whose right-hand side has length two or less, that defines the same transformation as $u \to v$.

11. Construct a sequence of context-sensitive rules $uAv \to uwv$ that define the same transformation as the monotonic rule $AB \to CD$. *Hint:* A sequence of three rules, each of whose left-hand side and right-hand side is of length two, suffices.

12. Use the results from Exercises 10 and 11 to prove that every context-sensitive language is generated by a grammar in which each rule has the form $uAv \to uwv$, where $w \in (V \cup \Sigma)^+$ and $u, v \in (V \cup \Sigma)^*$.

* 13. Let T be a full binary tree. A path through T is a sequence of left-down (L), right-down (R), or up (U) moves. Thus paths may be identified with strings over $\Sigma = \{L, R, U\}$. Consider the language $L = \{w \in \Sigma^* \mid w$ describes a path from the root back to the root$\}$. For example, λ, LU, $LRUULU \in L$, and U, $LRU \notin L$. Establish L's place in the Chomsky hierarchy.

14. Prove that the context-sensitive languages are not closed under arbitrary homomorphisms. A homomorphism is λ-free if $h(u) = \lambda$ implies $u = \lambda$. Prove that the context-sensitive grammars are closed under λ-free homomorphisms.

* 15. Let L be a recursively enumerable language over Σ and c a terminal symbol not in Σ. Show that there is a context-sensitive language L′ over $\Sigma \cup \{c\}$ such that for every $w \in \Sigma^*$, $w \in L$ if, and only if, $wc^i \in L'$ for some $i \geq 0$.

16. Prove that every recursively enumerable language is the homomorphic image of a context-sensitive language. *Hint:* Use Exercise 15.

17. A grammar is said to be context-sensitive with erasing if every rule has the form $uAv \to uvw$, where $A \in V$ and $u, v, w \in (V \cup \Sigma)^*$. Prove that this family of grammars generates the recursively enumerable languages.

18. A linear-bounded automaton is deterministic if at most one transition is specified for each state and tape symbol. Prove that every context-free language is accepted by a deterministic LBA.

19. Let L be a context-sensitive language that is accepted by a deterministic LBA. Prove that $\overline{L}$ is context-sensitive. Recall that a computation in an arbitrary deterministic LBA need not halt.

Bibliographic Notes

The Chomsky hierarchy was introduced by Chomsky [1956, 1959]. This paper includes the proof that the unrestricted grammars generate precisely recursively enumerable languages. Linear-bounded automata were presented in Myhill [1960]. The relationship between linear-bounded automata and context-sensitive languages was developed by Landweber [1963] and Kuroda [1964]. Solutions to Exercises 10, 11, and 12, which exhibit the relationship between monotonic and context-sensitive grammars, can be found in Kuroda [1964].