

## Chapter 2

# Formal Language Theory: Basics

This chapter covers the basics of formal language theory. It covers all the notions that are necessary to follow the rest of this book. Apart from the classical rudiments, however, the chapter covers several lesser-known areas of this theory, such as parallel grammars, because these areas are also needed to fully grasp some upcoming topics discussed in this book. The chapter consists of four sections.

Section 2.1 introduces the very basics concerning strings, languages, and operations over them. Section 2.2 defines rewriting systems as fundamental language-defining devices. Section 2.3 overviews a variety of formal grammars, and Sect. 2.4 covers automata needed to follow the rest of this book.

Readers familiar with the classical concepts used in formal languages theory should primarily concentrate their attention on non-classical concepts, such as language-defining devices working in parallel.

### 2.1 Languages

An *alphabet*  $\Sigma$  is a finite, nonempty set of elements called *symbols*. If  $\text{card}(\Sigma) = 1$ , then  $\Sigma$  is a *unary alphabet*. A *string* or, synonymously, a *word* over  $\Sigma$  is any finite sequence of symbols from  $\Sigma$ . We omit all separating commas in strings; that is, for a string  $a_1, a_2, \dots, a_n$ , for some  $n \geq 1$ , we write  $a_1 a_2 \dots a_n$  instead. The *empty string*, denoted by  $\varepsilon$ , is the string that is formed by no symbols, i.e. the empty sequence. By  $\Sigma^*$ , we denote the set of all strings over  $\Sigma$  (including  $\varepsilon$ ). Set  $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ .

Let  $x$  be a string over  $\Sigma$ , i.e.  $x \in \Sigma^*$ , and express  $x$  as  $x = a_1 a_2 \dots a_n$ , where  $a_i \in \Sigma$ , for all  $i = 1 \dots, n$ , for some  $n \geq 0$  (the case when  $n = 0$  means that  $x = \varepsilon$ ). The *length* of  $x$ , denoted by  $|x|$ , is defined as  $|x| = n$ . The *reversal* of  $x$ , denoted by  $\text{reversal}(x)$ , is defined as  $\text{reversal}(x) = a_n a_{n-1} \dots a_1$ . The *alphabet* of  $x$ , denoted by  $\text{alph}(x)$ , is defined as  $\text{alph}(x) = \{a_1, a_2, \dots, a_n\}$ ; informally, it is the set of symbols appearing in  $x$ . For  $U \subseteq \Sigma$ ,  $\#_U(x)$  denotes the number of occurrences

of symbols from  $U$  in  $x$ . If  $U = \{a\}$ , then instead of  $\#_{\{a\}}(x)$ , we write just  $\#_a(x)$ . The *leftmost symbol* of  $x$ , denoted by  $\text{lms}(x)$ , is defined as  $\text{lms}(x) = a_1$  if  $n \geq 1$  and  $\text{lms}(x) = \varepsilon$  otherwise. The *rightmost symbol* of  $x$ , denoted by  $\text{rms}(x)$ , is defined analogously. If  $n \geq 1$ , then for every  $i = 1, \dots, n$ , let  $\text{sym}(x, i)$  denote the  $i$ th symbol in  $x$ . Notice that  $|\varepsilon| = 0$ ,  $\text{reversal}(\varepsilon) = \varepsilon$ , and  $\text{alph}(\varepsilon) = \emptyset$ ,

Let  $x$  and  $y$  be two strings over  $\Sigma$ . Then,  $xy$  is the *concatenation* of  $x$  and  $y$ . Note that  $x\varepsilon = \varepsilon x = x$ . If  $x$  can be written in the form  $x = uv$ , for some  $u, v \in \Sigma^*$ , then  $u$  is a *prefix* of  $x$  and  $v$  is a *suffix* of  $x$ . If  $0 < |u| < |x|$ , then  $u$  is a *proper prefix* of  $x$ ; similarly, if  $0 < |v| < |x|$ , then  $v$  is a *proper suffix* of  $x$ . Define  $\text{prefix}(x) = \{u \mid u \text{ is a prefix of } x\}$  and  $\text{suffix}(x) = \{v \mid v \text{ is a suffix of } x\}$ . For every  $i \geq 0$ ,  $\text{prefix}(x, i)$  is the prefix of  $x$  of length  $i$  if  $|x| \geq i$ , and  $\text{prefix}(x, i) = x$  if  $|x| < i$ . If  $x = uvw$ , for some  $u, v, w \in \Sigma^*$ , then  $v$  is a *substring* of  $x$ . The set of all substrings of  $x$  is denoted by  $\text{sub}(x)$ . Moreover,

$$\text{sub}(y, k) = \{x \mid x \in \text{sub}(y), |x| \leq k\}$$

Let  $n$  be a nonnegative integer. Then, the  $n$ th *power* of  $x$ , denoted by  $x^n$ , is a string over  $\Sigma$  recursively defined as

$$\begin{aligned} (1) \quad x^0 &= \varepsilon \\ (2) \quad x^n &= xx^{n-1} \text{ for } n \geq 1 \end{aligned}$$

Let  $x = a_1a_2 \cdots a_n$  be a string over  $\Sigma$ , for some  $n \geq 0$ . The set of all *permutations* of  $x$ , denoted by  $\text{perm}(x)$ , is defined as

$$\text{perm}(x) = \{b_1b_2 \cdots b_n \mid b_i \in \text{alph}(x), \text{ for all } i = 1, \dots, n, \text{ and } (b_1, b_2, \dots, b_n) \text{ is a permutation of } (a_1, a_2, \dots, a_n)\}$$

Note that  $\text{perm}(\varepsilon) = \varepsilon$ .

A *language*  $L$  over  $\Sigma$  is any set of strings over  $\Sigma$ , i.e.  $L \subseteq \Sigma^*$ . The set  $\Sigma^*$  is called the *universal language* because it consists of all strings over  $\Sigma$ . If  $L$  is a finite set, then it is a *finite language*; otherwise, it is an *infinite language*. The set of all finite languages over  $\Sigma$  is denoted by  $\text{fin}(\Sigma)$ . For  $L \in \text{fin}(\Sigma)$ ,  $\text{max-len}(L)$  denotes the length of the longest string in  $L$ . We set  $\text{max-len}(\emptyset) = 0$ . If  $\text{card}(\Sigma) = 1$ , then  $L$  is a *unary language*. The *empty language* is denoted by  $\emptyset$ .

The *alphabet* of  $L$ , denoted by  $\text{alph}(L)$ , is defined as

$$\text{alph}(L) = \bigcup_{x \in L} \text{alph}(x)$$

The *permutation* of  $L$ , denoted by  $\text{perm}(L)$ , is defined as

$$\text{perm}(L) = \{\text{perm}(x) \mid x \in L\}$$

The *reversal* of  $L$ , denoted by  $\text{reversal}(L)$ , is defined as

$$\text{reversal}(L) = \{\text{reversal}(x) \mid x \in L\}$$

For every  $L \subseteq \Sigma^*$ , where  $\{\varepsilon\} \subseteq L$ , and every  $x \in \Sigma^*$ ,  $\text{max-prefix}(x, L)$  denotes the longest prefix of  $x$  that is in  $L$ ; analogously,  $\text{max-suffix}(x, L)$  denotes the longest suffix of  $x$  that is in  $L$ .

Let  $L_1$  and  $L_2$  be two languages over  $\Sigma$ . Throughout the book, we consider  $L_1$  and  $L_2$  to be *equal*, symbolically written as  $L_1 = L_2$ , if  $L_1 \cup \{\varepsilon\}$  and  $L_2 \cup \{\varepsilon\}$  are identical. Similarly,  $L_1 \subseteq L_2$  if  $L_1 \cup \{\varepsilon\}$  is a subset of  $L_2 \cup \{\varepsilon\}$ .

As all languages are sets, all common operations over sets can be applied to them. Therefore,

$$\begin{aligned} L_1 \cup L_2 &= \{x \mid x \in L_1 \text{ or } x \in L_2\} \\ L_1 \cap L_2 &= \{x \mid x \in L_1 \text{ and } x \in L_2\} \\ L_1 - L_2 &= \{x \mid x \in L_1 \text{ and } x \notin L_2\} \end{aligned}$$

The *complement* of  $L$ , denoted by  $\bar{L}$ , is defined as

$$\bar{L} = \{x \mid x \in \Sigma^*, x \notin L\}$$

There are also some special operations which apply only to languages. The *concatenation* of  $L_1$  and  $L_2$ , denoted by  $L_1L_2$ , is the set

$$L_1L_2 = \{x_1x_2 \mid x_1 \in L_1 \text{ and } x_2 \in L_2\}$$

Note that  $L\{\varepsilon\} = \{\varepsilon\}L = L$ . For  $n \geq 0$ , the  *$n$ th power* of  $L$ , denoted by  $L^n$ , is recursively defined as

$$\begin{aligned} (1) \quad L^0 &= \{\varepsilon\} \\ (2) \quad L^n &= L^{n-1}L \end{aligned}$$

The *closure* (*Kleene star*) of a language  $L$ , denoted by  $L^*$ , is the set

$$L^* = \bigcup_{i \geq 0} L^i$$

The *positive closure* of a language  $L$ , denoted by  $L^+$ , is the set

$$L^+ = \bigcup_{i \geq 1} L^i$$

The *right quotient* of  $L_1$  with respect to  $L_2$ , denoted by  $L_1/L_2$ , is defined as

$$L_1/L_2 = \{y \mid yx \in L_1, \text{ for some } x \in L_2\}$$

Similarly, the *left quotient* of  $L_1$  with respect to  $L_2$ , denoted by  $L_2 \backslash L_1$ , is defined as

$$L_2 \backslash L_1 = \{y \mid xy \in L_1, \text{ for some } x \in L_2\}$$

We also use special types of the right and left quotients. The *exhaustive right quotient* of  $L_1$  with respect to  $L_2$ , denoted by  $L_1 \parallel L_2$ , is defined as

$$L_1 \parallel L_2 = \{y \mid yx \in L_1, \text{ for some } x \in L_2, \text{ and no } x' \in L_2 \text{ such that } |x'| > |x| \text{ is a proper suffix of } yx\}$$

Similarly, the *exhaustive left quotient* of  $L_1$  with respect to  $L_2$ , denoted by  $L_2 \parallel L_1$ , is defined as

$$L_2 \parallel L_1 = \{x \mid yx \in L_1, \text{ for some } y \in L_2, \text{ and no } y' \in L_2 \text{ such that } |y'| > |y| \text{ is a proper prefix of } yx\}$$

Let  $L_2 = \{\$\}^*$ , where  $\$$  is a symbol. Then,  $L_1 \parallel L_2$  is the *symbol-exhaustive right quotient* of  $L_1$  with respect to  $\$$ , and  $L_2 \parallel L_1$  is the *symbol-exhaustive left quotient* of  $L_1$  with respect to  $\$$ .

Let  $\Sigma$  be an alphabet. For  $x, y \in \Sigma^*$ , the *shuffle* of  $x$  and  $y$ , denoted by  $\text{shuffle}(x, y)$ , is defined as

$$\text{shuffle}(x, y) = \{x_1y_1x_2y_2 \cdots x_ny_n \mid x = x_1x_2 \cdots x_n, y = y_1y_2 \cdots y_n, x_i, y_i \in \Sigma^*, 1 \leq i \leq n, n \geq 1\}$$

We extend the shuffle operation on languages in the following way. For  $K_1, K_2 \subseteq \Sigma^*$ ,

$$\text{shuffle}(K_1, K_2) = \{z \mid z \in \text{shuffle}(x, y), x \in K_1, y \in K_2\}$$

Let  $\Sigma$  and  $\Gamma$  be two alphabets. Let  $K$  and  $L$  be languages over alphabets  $\Sigma$  and  $\Gamma$ , respectively. A *translation* from  $K$  to  $L$  is a relation  $\rho$  from  $\Sigma^*$  to  $\Gamma^*$  with  $\text{domain}(\rho) = K$  and  $\text{range}(\rho) = L$ . A total function  $\sigma$  from  $\Sigma^*$  to  $\Gamma^*$  such that  $\sigma(uv) = \sigma(u)\sigma(v)$ , for every  $u, v \in \Sigma^*$ , is a *substitution*. A substitution is  *$\varepsilon$ -free* if it is defined from  $\Sigma^*$  to  $\Gamma^{+}$ . If  $\sigma(a)$  for every  $a \in \Sigma$  is finite, then  $\sigma$  is said to be *finite*. By this definition,  $\sigma(\varepsilon) = \{\varepsilon\}$  and  $\sigma(a_1a_2 \cdots a_n) = \sigma(a_1)\sigma(a_2) \cdots \sigma(a_n)$ , where  $n \geq 1$  and  $a_i \in \Sigma$ , for all  $i = 1, 2, \dots, n$ , so  $\sigma$  is completely specified by defining  $\sigma(a)$  for each  $a \in \Sigma$ . For  $L \subseteq \Sigma^*$ , we extend the definition of  $\sigma$  to

$$\sigma(L) = \bigcup_{w \in L} \sigma(w)$$

A total function  $\varphi$  from  $\Sigma^*$  to  $\Gamma^*$  such that  $\varphi(uv) = \varphi(u)\varphi(v)$ , for every  $u, v \in \Sigma^*$ , is a *homomorphism* or, synonymously, a *morphism*. As any homomorphism is a

special case of finite substitution, we specify  $\varphi$  by analogy with the specification of  $\sigma$ . For  $L \subseteq \Sigma^*$ , we extend the definition of  $\varphi$  to

$$\varphi(L) = \{\varphi(w) \mid w \in L\}$$

By analogy with substitution,  $\varphi$  is  $\varepsilon$ -free if  $\varphi(a) \neq \varepsilon$ , for every  $a \in \Sigma$ . By  $\varphi^{-1}$ , we denote the *inverse homomorphism*, defined as

$$\varphi^{-1}(u) = \{w \mid \varphi(w) = u\}$$

A homomorphism  $\omega$  from  $\Sigma^*$  represents an *almost identity* if there exists a symbol  $\# \in \Sigma$  such that  $\omega(a) = a$ , for every  $a \in \Sigma - \{\#\}$ , and  $\omega(\#) \in \{\#, \varepsilon\}$ . A homomorphism  $\tau$  from  $\Sigma^*$  to  $\Gamma^*$  is a *coding* if  $\tau(a) \in \Gamma$ , for every  $a \in \Sigma$ .

Let  $L$  be a language over  $\Sigma$ , and let  $k$  be a positive integer. A homomorphism  $\lambda$  over  $\Sigma^*$  is a *k-linear erasing* with respect to  $L$  if and only if for each  $y \in L$ ,  $|y| \leq k|\lambda(y)|$ . Furthermore, if  $L \subseteq (\Sigma\{\varepsilon, c, c^2, \dots, c^k\})^*$ , for some  $c \notin \Sigma$  and  $k \geq 1$ , and  $\lambda$  is defined by  $\lambda(c) = \varepsilon$  and  $\lambda(a) = a$ , for all  $a \in \Sigma$ , then we say that  $\lambda$  is *k-restricted* with respect to  $L$ . Clearly, each *k-restricted* homomorphism is a *k-linear erasing*.

### 2.1.1 Language Families

By analogy with set theory, sets whose members are languages are called *families of languages*. A family of languages  $\mathcal{L}$  is  $\varepsilon$ -free if for every  $L \in \mathcal{L}$ ,  $\varepsilon \notin L$ . The family of finite languages is denoted by **FIN**.

Just like for languages, we consider two language families,  $\mathcal{L}_1$  and  $\mathcal{L}_2$ , *equal* if and only if

$$\bigcup_{L \in \mathcal{L}_1} L \cup \{\varepsilon\} = \bigcup_{K \in \mathcal{L}_2} K \cup \{\varepsilon\}$$

If  $\mathcal{L}_1$  and  $\mathcal{L}_2$  are equal, we write  $\mathcal{L}_1 = \mathcal{L}_2$ . We also say that these two families *coincide*.  $\mathcal{L}_1$  is a *subset* of  $\mathcal{L}_2$ , written as  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , if and only if

$$\bigcup_{L \in \mathcal{L}_1} L \cup \{\varepsilon\} \subseteq \bigcup_{K \in \mathcal{L}_2} K \cup \{\varepsilon\}$$

The closure of a language family under an operation is defined by analogy with the definition of the closure of a set. Next, we define three closure properties, discussed later in this book.

**Definition 2.1.1.** A language family  $\mathcal{L}$  is *closed under linear erasing* if and only if for all  $L \in \mathcal{L}$ ,  $\lambda(L)$  is also in  $\mathcal{L}$ , where  $\lambda$  is a  $k$ -linear erasing with respect to  $L$ , for some  $k \geq 1$ .  $\square$

**Definition 2.1.2.** A language family  $\mathcal{L}$  is *closed under restricted homomorphism* if and only if for all  $L \in \mathcal{L}$ ,  $\lambda(L)$  is also in  $\mathcal{L}$ , where  $\lambda$  is a  $k$ -restricted homomorphism with respect to  $L$ , for some  $k \geq 1$ .  $\square$

**Definition 2.1.3.** Let  $\mathcal{L}$  be a language family. We say that  $\mathcal{L}$  is *closed under endmarking* if and only if for every  $L \in \mathcal{L}$ , where  $L \subseteq \Sigma^*$ , for some alphabet  $\Sigma$ ,  $\# \notin \Sigma$  implies that  $L\{\#\} \in \mathcal{L}$ .  $\square$

## 2.2 Rewriting Systems as Basic Language Models

Just like finite sets, finite languages can be specified by listing all the strings they contain. Of course, infinite languages, including almost all programming and natural languages, cannot be specified by an exhaustive enumeration of the strings they contain. Therefore, we customarily specify languages by suitable mathematical models so the models are of finite size even if the languages being specified are not. In the present section, based upon the mathematical notion of a relation, we define rewriting systems for this purpose.

**Definition 2.2.1.** A *rewriting system* is a pair,  $M = (\Sigma, R)$ , where  $\Sigma$  is an alphabet, and  $R$  is a finite relation on  $\Sigma^*$ .  $\Sigma$  is called the *total alphabet of  $M$*  or, simply, the *alphabet of  $M$* . A member of  $R$  is called a *rule of  $M$* , and accordingly,  $R$  is referred to as the *set of rules in  $M$* .

The *rewriting relation* over  $\Sigma^*$  is denoted by  $\vdash_M$  and defined so that for every  $u, v \in \Sigma^*$ ,  $u \vdash_M v$  in  $M$  iff there exist  $(x, y) \in R$  and  $w, z \in \Sigma^*$  such that  $u = wxz$  and  $v = wyz$ . As usual,  $\vdash_M^+$  and  $\vdash_M^*$  denote the transitive and transitive and reflexive closure of  $\vdash_M$ , respectively.  $\square$

Let  $M = (\Sigma, R)$  be a rewriting system. Each rule  $(x, y) \in R$  is written as  $x \rightarrow y$  throughout this book. For  $x \rightarrow y \in R$ ,  $x$  and  $y$  represent the *left-hand side* of  $x \rightarrow y$  and the *right-hand side* of  $x \rightarrow y$ , respectively. We drop  $M$  in  $\vdash_M$  and, thereby, simplify  $\vdash_M$  to  $\vdash$  whenever  $M$  is automatically understood. By  $u \vdash v [x \rightarrow y]$ , where  $u, v \in \Sigma^*$  and  $x \rightarrow y \in R$ , we express that  $M$  directly rewrites  $u$  as  $v$  according to  $x \rightarrow y$ . Of course, whenever the information regarding the applied rule is immaterial, we omit its specification; in other words, we simplify  $u \vdash v [x \rightarrow y]$  to  $u \vdash v$ . By underlining, we specify the substring rewritten during a rewriting step if necessary. More formally, if  $u = wxz$ ,  $v = wyz$ ,  $x \rightarrow y \in R$ , where  $u, v, x, y \in \Sigma^*$ , then  $w\underline{xz} \vdash w\underline{yz} [x \rightarrow y]$  means that the  $x$  occurring behind  $w$  is rewritten during this step by using  $x \rightarrow y$  (we usually specify the rewritten occurrence of  $x$  in this way when other occurrences of  $x$  appear in  $w$  and  $z$ ).

To give a straightforward insight into the application of the defined notion, we now give the following examples. As the principle subject of this section, we discuss languages and their representations. It is thus only natural to illustrate our discussion by linguistically oriented examples.

*Example 2.2.2.* Let  $\Delta$  denote the alphabet of English small letters (this alphabet is used in all examples of this section). In the present example, we introduce a rewriting system  $M$  that translates every digital string to the string in which every digit is converted to its corresponding English name followed by #; for instance, 010 is translated to *zero#one#zero#*.

First, we define the finite function  $h$  from  $\{0, 1, \dots, 9\}$  to  $\Delta^*$  as

$$\begin{aligned} h(0) &= \text{zero}, \\ h(1) &= \text{one}, \\ h(2) &= \text{two}, \\ h(3) &= \text{three}, \\ h(4) &= \text{four}, \\ h(5) &= \text{five}, \\ h(6) &= \text{six}, \\ h(7) &= \text{seven}, \\ h(8) &= \text{eight}, \\ h(9) &= \text{nine} \end{aligned}$$

In words,  $h$  translates every member of  $\{0, 1, \dots, 9\}$  to its corresponding English name; for instance,  $h(9) = \text{nine}$ . Based upon  $h$ , we define  $M = (\Sigma, R)$  with  $\Sigma = \{0, 1, \dots, 9\} \cup \Delta \cup \{\#\}$  and  $R = \{i \rightarrow h(i)\# \mid i \in \{0, 1, \dots, 9\}\}$ . Finally, we define the function  $T(M)$  from  $\{0, 1, \dots, 9\}^*$  to  $(\Delta \cup \{\#\})^*$  as

$$T(M) = \{(s, t) \mid s \vdash^* t, s \in \{0, 1, \dots, 9\}^*, t \in (\Delta \cup \{\#\})^*\}$$

For instance,  $T(M)$  contains  $(911, \text{nine\#one\#one\#})$ . Indeed,  $M$  translates 911 to *nine#one#one#* as follows

$$\begin{array}{lll} 9\underline{1}1 & \vdash & 9\text{one\#}1 \quad [1 \rightarrow \text{one\#}] \\ 9\text{one\#}\underline{1} & \vdash & 9\text{one\#one\#} \quad [1 \rightarrow \text{one\#}] \\ 9\text{one\#one\#} & \vdash & \text{nine\#one\#one\#} \quad [9 \rightarrow \text{nine\#}] \end{array}$$

Thus,  $911 \vdash^* \text{nine\#one\#one\#} [1 \rightarrow \text{one\#}, 1 \rightarrow \text{one\#}, 9 \rightarrow \text{nine\#}]$ . Therefore,  $(911, \text{nine\#one\#one\#}) \in T(M)$ . Thus,  $M$  performs the desired translation.  $\square$

*Example 2.2.3.* This example strongly resembles a very simple morphological study—in linguistics, *morphology* studies the structure of words. Indeed, it discusses re-structuring strings consisting of English letters, including strings that does not represent any English words, such as *xxuy*. More precisely, we introduce a rewriting system  $M$  that

- (1) starts from non-empty strings consisting of small English letters delimited by angle brackets,

- (2) orders the letters lexicographically, and
- (3) eliminates the angle brackets.

For instance,  $M$  changes  $\langle xxyy \rangle$  to  $uxxy$ .

Let  $\Delta$  have the same meaning as in Example 2.2.2—that is,  $\Delta$  denotes the alphabet of English lowercases. Let  $_{\text{lex}} <$  denote the standardly defined lexical order over  $\Delta$ —that is,

$$a_{\text{lex}} < b_{\text{lex}} < c_{\text{lex}} < \dots < y_{\text{lex}} < z$$

We define  $M = (\Sigma, R)$  with  $\Sigma = \Delta \cup \{\langle, \rangle, 1, 2, 3\}$  and  $R$  containing the following rules

- (i)  $\langle \rightarrow 12, 12 \rightarrow 3$
- (ii)  $2\alpha \rightarrow \alpha 2$  and  $\alpha 2 \rightarrow 2\alpha$  for all  $\alpha \in \Delta$
- (iii)  $\beta 2\alpha \rightarrow \alpha 2\beta$  for all  $\alpha, \beta \in \Delta$  such that  $\alpha_{\text{lex}} < \beta$
- (iv)  $3\alpha\beta \rightarrow \alpha 3\beta$  for all  $\alpha, \beta \in \Delta$  such that  $\alpha_{\text{lex}} < \beta$  or  $\alpha = \beta$
- (v)  $3\alpha \rightarrow \alpha$  for all  $\alpha \in \Delta$

Define the function  $T(M)$  from  $(\{\langle, \rangle\} \cup \Delta)^+$  to  $\Delta^+$  as

$$T(M) = \{(\langle s \rangle, t) \mid \langle s \rangle \vdash^* t, \text{ where } s, t \in \Delta^+\}$$

Observe that  $(\langle s \rangle, t) \in TM$  if and only if  $t$  is a permutation of  $s$  such that  $t$  has its letters lexicographically ordered according to  $_{\text{lex}} <$ . For instance,  $T(M)$  contains  $(\langle \text{order} \rangle, \text{deorr})$ . Indeed,  $M$  translates  $\langle \text{order} \rangle$  to  $\text{deorr}$  as follows

$$\begin{aligned}
\langle \text{order} \rangle &\vdash 12\text{order} \rangle \quad [\langle \rightarrow 12] \\
12\text{order} \rangle &\vdash 1o2rder \rangle \quad [2o \rightarrow o2] \\
1o2rder \rangle &\vdash 1or2der \rangle \quad [2r \rightarrow r2] \\
1or2der \rangle &\vdash 1od2rer \rangle \quad [r2d \rightarrow d2r] \\
1od2rer \rangle &\vdash 1odr2er \rangle \quad [2r \rightarrow r2] \\
1odr2er \rangle &\vdash 1ode2rr \rangle \quad [r2e \rightarrow e2r] \\
1ode2rr \rangle &\vdash 1od2err \rangle \quad [e2 \rightarrow 2e] \\
1od2err \rangle &\vdash 1o2derr \rangle \quad [d2 \rightarrow 2d] \\
1o2derr \rangle &\vdash 1d2oerr \rangle \quad [o2d \rightarrow d2o] \\
1d2oerr \rangle &\vdash 1do2err \rangle \quad [2o \rightarrow o2] \\
1do2err \rangle &\vdash 1de2orr \rangle \quad [o2e \rightarrow e2o] \\
1de2orr \rangle &\vdash 1d2eorr \rangle \quad [e2 \rightarrow 2e] \\
1d2eorr \rangle &\vdash 12deorr \rangle \quad [d2 \rightarrow 2d] \\
12deorr \rangle &\vdash 3deorr \rangle \quad [12 \rightarrow 3] \\
3deorr \rangle &\vdash d3eorr \rangle \quad [3de \rightarrow d3e] \\
d3eorr \rangle &\vdash de3orr \rangle \quad [3eo \rightarrow e3o] \\
de3orr \rangle &\vdash deo3rr \rangle \quad [3or \rightarrow o3r] \\
deo3rr \rangle &\vdash deor3r \rangle \quad [3rr \rightarrow r3r] \\
deor3r \rangle &\vdash deorr \quad [3r \rightarrow r]
\end{aligned}$$



Observe that  $M$  can translate  $\langle order \rangle$  to  $deorr$  by a number of different sequences of rewriting steps. In fact, it can translate infinitely many members of  $T(M)$  in various ways. In general, this phenomenon is referred to as non-determinism; accordingly, rewriting systems working in this way are said to be non-deterministic. In mathematics, we usually design the basic versions of rewriting systems so they work non-deterministically. In terms of their implementation, however, we obviously prefer using their deterministic versions. Therefore, we usually place a restriction on the way the rules are applied so the rewriting systems restricted in this way necessarily work deterministically; simultaneously, we obviously want that the deterministic restricted versions perform the same job as their original unrestricted counterparts.  $\square$

In the rest of this section, we focus on its key subject, which consists in using rewriting systems as language-defining models.

Whenever we use a rewriting system,  $M = (\Sigma, R)$ , as a language-defining model, then for brevity, we denote the language that  $M$  defines by  $L(M)$ . In principal,  $M$  defines  $L(M)$  so it either *generates*  $L(M)$  or *accepts*  $L(M)$ . Next, we explain these two fundamental language-defining methods in a greater detail. Let  $S \in \Sigma^*$  and  $F \in \Sigma^*$  be a *start language* and a *final language*, respectively.

- (1) The *language generated* by  $M$  is defined as the set of all strings  $y \in F$  such that  $x \vdash^* y$  in  $M$  for some  $x \in S$ .  $M$  used in this way is generally referred to as a *language-generating model* or, more briefly, a *grammar*.
- (2) The *language accepted* by  $M$  is defined as the set of all strings  $x \in S$  such that  $x \vdash^* y$  in  $M$  for some  $y \in F$ .  $M$  used in this way is referred to as a *language-accepting model* or, more briefly, an *automaton*.

*Example 2.2.4.* Let  $\Delta$  have the same meaning as in Examples 2.2.2 and 2.2.3—that is,  $\Delta$  denotes the alphabet of English lowercases. Let  $L$  be the language consisting of all even-length palindromes over  $\Delta$ —a *palindrome* is a string that is the same whether written forwards or backward. For instance, *aa* and *noon* belong to  $L$ , but *ba* and *oops* do not. The present example introduces a grammar and an automaton that define  $L$ .

Let  $G = (\Sigma, P)$  be the rewriting system with  $\Sigma = \Delta \cup \{\#\}$  and

$$P = \{\# \rightarrow a\#a \mid a \in \Delta\} \cup \{\# \rightarrow \varepsilon\}$$

Set  $S = \{\#\}$  and  $F = \Delta^*$ . Define the language generated by  $G$  as

$$L(G) = \{t \mid s \vdash^* t, s \in S, t \in F\}$$

In other words,

$$L(G) = \{t \mid \# \vdash^* t \text{ with } t \in \Delta^*\}$$

Observe that  $G$  acts as a grammar that generates  $L$ . For instance,

$$\# \vdash n\#n \vdash no\#on \vdash noon$$

in  $G$ , so  $noon \in L(G)$ .

To give an automaton that accepts  $L$ , introduce the rewriting system  $A = (\Sigma, R)$  with  $\Sigma = \Delta \cup \{\#\}$  and

$$R = \{a\#a \rightarrow \# \mid a \in \Delta\}$$

Set  $S = \Delta^*\{\#\}\Delta^*$  and  $F = \{\#\}$ . Define the language accepted by  $A$  as

$$L(A) = \{st \mid s\#t \vdash^* u, \text{ where } s, t \in \Delta^*, u \in F\}$$

That is,

$$L(A) = \{st \mid s\#t \vdash^* \#, \text{ where } s, t \in \Delta^*\}$$

For instance,  $A$  accepts  $noon$

$$\underline{no}\#on \vdash \underline{n}\#n \vdash \#$$

(as stated in the comments following Definition 2.2.1, the underlined substrings denote the substrings that are rewritten). On the other hand, consider this sequence of rewriting steps

$$\underline{no}\#onn \vdash \underline{n}\#nn \vdash \#n$$

It starts from  $no\#onn$ , and after performing two steps, it ends up with  $\#n$ , which cannot be further rewritten. Since  $\#n \notin F$ , which equals  $\{\#\}$ ,  $M$  does not accept  $no\#onn$ . As an exercise, based upon these observations, demonstrate that  $L = L(A)$ , so  $A$  acts as an automaton that accepts  $L$ .  $\square$

Before closing this section, we make use of Example 2.2.4 to explain and illustrate the concept of equivalence and that of determinism in terms of rewriting systems that define languages.

### 2.2.1 Equivalence

If some rewriting systems define the same language, they are said to be *equivalent*. For instance, take  $G$  and  $A$  in Example 2.2.4. Both define the same language, so they are equivalent.

### 2.2.2 Determinism

Recall that Example 2.2.3 has already touched the topic of determinism in terms of rewriting systems. Notice that the language-defining rewriting system  $A$  from Example 2.2.4 works deterministically in the sense that  $A$  rewrites any string from

$K$  by no more than one rule, where  $K = S$ , so  $K = \Delta^*\{\#\}\Delta^*$ . To express this concept of determinism more generally, let  $M = (\Sigma, R)$  be a rewriting system and  $K \subseteq \Sigma^*$ .  $M$  is *deterministic over  $K$*  if for every  $w \in K$ , there is no more than one  $r \in R$  such that  $w \vdash v [r]$  with  $v \in K$ . Mathematically, if  $M$  is deterministic over  $K$ , then its rewriting relation  $\vdash$  represents a function over  $K$ —that is, for all  $u, v, w \in K$ , if  $u \vdash v$  and  $u \vdash w$ , then  $v = w$ . When  $K$  is understood, we usually just say that  $M$  is *deterministic*; frequently, we take  $K = \Sigma^*$ .

As already noted in Example 2.2.3, the basic versions of language-defining rewriting systems are always introduced quite generally and, therefore, non-deterministically. That is also why we first define the basic versions of these models in a non-deterministic way throughout this book. In practice, however, we obviously prefer their deterministic versions because they are easy to implement. Therefore, we always study whether any non-deterministic version can be converted to an equivalent deterministic version, and if so, we want to perform this conversion algorithmically. More specifically, we reconsider this crucially important topic of determinism in terms of finite automata in Sect. 2.4.

## 2.3 Grammars

In this section, based upon rewriting systems, we define grammars as formal devices that generate languages. Grammars play an important role throughout this book.

### 2.3.1 Grammars in General

Since grammars represent special cases of rewriting systems (see Sect. 2.2), we often use the mathematical terminology concerning these systems throughout the present section. Specifically, we apply the relations  $\vdash$ ,  $\vdash^n$ ,  $\vdash^+$ , and  $\vdash^*$  to these grammars.

The notion of a *grammar* represents a rewriting system  $G = (\Sigma, R)$ , where

- $\Sigma$  is divided into two disjoint subalphabets, denoted by  $N$  and  $T$ ;
- $R$  is a finite set of rules of the form  $A \rightarrow x$ , where  $A \in N$  and  $x \in \Sigma^*$ .

$N$  and  $T$  are referred to as the *alphabet of nonterminal symbols* and the *alphabet of terminal symbols*, respectively.  $N$  contains a special *start symbol*, denoted by  $S$ .

If  $S \vdash^* w$ , where  $w \in \Sigma^*$ ,  $G$  *derives*  $w$ , and  $w$  is a *sentential form*.  $F(G)$  denotes the set of all sentential forms derived by  $G$ . The *language generated by  $G$* , symbolically denoted by  $L(G)$ , is defined as  $L(G) = F(G) \cap T^*$ . Members of  $L(G)$  are called *sentences*. If  $S \vdash^* w$  and  $w$  is a sentence,  $S \vdash^* w$  is a *successful derivation* in  $G$ .

More customarily, however, the notion of a grammar or, more precisely that of a phrase-structure grammar is defined as follows.

**Definition 2.3.1.** A *phrase-structure grammar* is a quadruple

$$G = (V, T, P, S)$$

where

- $V$  is a *total alphabet*;
- $T$  is an alphabet of *terminals* such that  $T \subseteq V$ ;
- $P$  is a finite relation from  $V^* - T^*$  to  $V^*$ ;
- $S \in V - T$  is the *start symbol*.

The set  $N = V - T$  is the set of *nonterminals* such that  $N \cap T = \emptyset$ .

Pairs  $(u, v) \in P$  are called *rewriting rules* (abbreviated *rules*) or *productions*, and are written as  $u \rightarrow v$ . A rewriting rule  $u \rightarrow v \in P$  satisfying  $v = \varepsilon$  is called an *erasing rule*. If there is no such rule in  $P$ , then we say that  $G$  is a *propagating* (or  *$\varepsilon$ -free*) grammar.

The  $G$ -based *direct derivation relation* over  $V^*$  is denoted by  $\Rightarrow_G$  and defined as

$$x \Rightarrow_G y$$

if and only if  $x = x_1 u x_2, y = y_1 v y_2$ , and  $u \rightarrow v \in P$ , where  $x_1, x_2, y_1, y_2 \in V^*$ . Since  $\Rightarrow_G$  is a relation,  $\Rightarrow_G^k$  is the  $k$ th power of  $\Rightarrow_G$ , for  $k \geq 0$ ,  $\Rightarrow_G^+$  is the transitive closure of  $\Rightarrow_G$ , and  $\Rightarrow_G^*$  is the reflexive-transitive closure of  $\Rightarrow_G$ . Let  $D: S \Rightarrow_G^* x$  be a derivation, for some  $x \in V^*$ . Then,  $x$  is a *sentential form*. If  $x \in T^*$ , then  $x$  is a *sentence*. If  $x$  is a sentence, then  $D$  is a *successful* (or *terminal*) *derivation*.

The *language* of  $G$ , denoted by  $L(G)$ , is the set of all sentences defined as

$$L(G) = \{w \in T^* \mid S \Rightarrow_G^* w\} \quad \square$$

Next, for every phrase-structure grammar  $G$ , we define two sets,  $F(G)$  and  $\Delta(G)$ .  $F(G)$  contains all sentential forms of  $G$ .  $\Delta(G)$  contains all sentential forms from which there is a derivation of a string in  $L(G)$ .

**Definition 2.3.2.** Let  $G = (V, T, P, S)$  be a phrase-structure grammar. Set

$$F(G) = \{x \in V^* \mid S \Rightarrow_G^+ x\}$$

and

$$\Delta(G) = \{x \in F(G)^* \mid x \Rightarrow_G^* y, y \in T^*\} \quad \square$$

For brevity, we often denote a rule  $u \rightarrow v$  with a unique label  $r$  as  $r: u \rightarrow v$ , and instead of  $u \rightarrow v \in P$ , we simply write  $r \in P$ . The notion of rule labels is formalized in the following definition.

**Definition 2.3.3.** Let  $G = (V, T, P, S)$  be a phrase-structure grammar. Let  $\Psi$  be a set of symbols called *rule labels* such that  $\text{card}(\Psi) = \text{card}(P)$ , and  $\psi$  be a bijection from  $P$  to  $\Psi$ . For simplicity and brevity, to express that  $\psi$  maps a rule,  $u \rightarrow v \in P$ , to  $r$ , where  $r \in \Psi$ , we write  $r: u \rightarrow v \in P$ ; in other words,  $r: u \rightarrow v$  means that  $\psi(u \rightarrow v) = r$ . For  $r: u \rightarrow v \in P$ ,  $u$  and  $v$  represent the *left-hand side* of  $r$ , denoted by  $\text{lhs}(r)$ , and the *right-hand side* of  $r$ , denoted by  $\text{rhs}(r)$ , respectively. Let  $P^*$  and  $\Psi^*$  denote the set of all sequences of rules from  $P$  and the set of all sequences of rule labels from  $\Psi$ , respectively. Set  $P^+ = P^* - \{\varepsilon\}$  and  $\Psi^+ = \Psi^* - \{\varepsilon\}$ . As with strings, we omit all separating commas in these sequences.

We extend  $\psi$  from  $P$  to  $P^*$  in the following way

- (1)  $\psi(\varepsilon) = \varepsilon$
- (2)  $\psi(r_1 r_2 \cdots r_n) = \psi(r_1) \psi(r_2) \cdots \psi(r_n)$

for any sequence of rules  $r_1 r_2 \cdots r_n$ , where  $r_i \in P$ , for all  $i = 1, 2, \dots, n$ , for some  $n \geq 1$ .

Let  $w_0, w_1, \dots, w_n$  be a sequence of strings, where  $w_i \in V^*$ , for all  $i = 0, 1, \dots, n$ , for some  $n \geq 1$ . If  $w_{j-1} \Rightarrow_G w_j$  according to  $r_j$ , where  $r_j \in P$ , for all  $j = 1, 2, \dots, n$ , then we write

$$w_0 \Rightarrow_G^n w_n [\psi(r_1 r_2 \cdots r_n)]$$

For any string  $w$ , we write

$$w \Rightarrow_G^0 w [\varepsilon]$$

For any two strings  $w$  and  $y$ , if  $w \Rightarrow_G^n y [\rho]$  for  $n \geq 0$  and  $\rho \in \Psi^*$ , then we write

$$w \Rightarrow_G^* y [\rho]$$

If  $n \geq 1$ , which means that  $|\rho| \geq 1$ , then we write

$$w \Rightarrow_G^+ y [\rho]$$

If  $w = S$ , then  $\rho$  is called the *sequence of rules* (*rule labels*) used in the derivation of  $y$  or, more briefly, the *parse*<sup>1</sup> of  $y$ . □

For any phrase-structure grammar  $G$ , we automatically assume that  $V, N, T, S, P$ , and  $\Psi$  denote its total alphabet, the alphabet of nonterminals, the alphabet of terminals, the start symbol, the set of rules, and the set of rule labels, respectively. Sometimes, we write  $G = (V, T, \Psi, P, S)$  instead of  $G = (V, T, P, S)$  with  $\Psi$  having the above-defined meaning.

---

<sup>1</sup>Let us note that the notion of a parse represents a synonym of several other notions, including a *derivation word*, a *Szilar word*, and a *control word*.

In the literature, a phrase-structure grammar is also often defined with rules of the form  $x \rightarrow y$ , where  $x \in V^+$  and  $y \in V^*$  (see, for instance, [Woo87]). Both definitions are interchangeable in the sense that the grammars defined in these two ways generate the same family of languages—the family of recursively enumerable languages.

**Definition 2.3.4.** A *recursively enumerable language* is a language generated by a phrase-structure grammar. The family of recursively enumerable languages is denoted by **RE**.  $\square$

Throughout this book, in the proofs, we frequently make use of Turing-Church thesis (see [Chu36b, Chu36a, Tur36]), which we next state in terms of formal language theory. Before this, however, we need to explain how we understand the *intuitive notion of an effective procedure* or, briefly, a *procedure*, contained in this thesis. We surely agree that each procedure describes how to perform a task in an unambiguous and detailed way. We also agree that it consists of finitely many instructions, each of which can be executed mechanically in a fixed amount of time. When performed, a procedure reads input data, executes its instructions, and produces output data; of course, both the input data and the output data may be nil. We are now ready to state Turing-Church thesis in terms of **RE**—that is, the family of recursively enumerable languages, defined by phrase-structure grammars (see Definition 2.3.1).

**Turing-Church Thesis.** Let  $L$  be a language. Then,  $L \in \mathbf{RE}$  if and only if there is a procedure that defines  $L$  by listing all its strings.

All the grammars and automata discussed in this book obviously constitutes procedures in the above sense. Consequently, whenever grammars or automata of a new type are considered in this book, Turing-Church thesis automatically implies that the language family they define is necessarily contained in **RE**, and we frequently make use of this implication in the sequel.

Observe that Turing-Church thesis is indeed a thesis, not a theorem because it cannot be proved. Indeed, any proof of this kind would necessitate a formalization of our intuitive notion of a language-defining procedure so it can be rigorously compared with the notion of a phrase-structure grammar. At this point, however, there would be a problem whether this newly formalized notion is equivalent to the intuitive notion of a procedure, which would give rise to another thesis similar to Turing-Church thesis. Therefore, any attempt to prove this thesis inescapably ends up with an infinite regression. However, the evidence supporting Turing-Church thesis is hardly disputable because throughout its history, computer science has formalized the notion of a procedure in the intuitive sense by other language-defining models, such as Post systems (see [Pos43]) and Markov algorithms (see [Mar60]), and all of them have turned out to be equivalent with phrase-structure grammars. Even more importantly, nobody has ever come with a procedure that defines a language and demonstrated that the language cannot be generated by any phrase-structure grammar.

Originally, Turing-Church thesis have been stated in terms of Turing machines in [Tur36]. Indeed, Church and Turing hypothesized that any computational process

which could be reasonably called as a procedure could be simulated by a Turing machine (see [Rog87] for an in-depth discussion concerning to Turing-Church thesis). In the present monograph, however, we do not need the notion of a Turing machine while we frequently make use of the notion of a phrase-structure grammar. Therefore, for the purposes of this book, we have reformulated Turing-Church thesis in the above way. As phrase-structure grammars and Turing machines are equivalent (see [Med00a]), this reformulation is obviously perfectly correct and legal from a mathematical viewpoint.

Any language models that characterize **RE** are said to be *computationally complete* because they are as strong as all possible procedures in terms of language-defining power according to Turing-Church thesis. Apart from them, however, this book also discusses many *computationally incomplete* language models, which define proper subfamilies of **RE**. For instance, the following special versions of phrase-structure grammars are all computationally incomplete.

**Definition 2.3.5.** A *context-sensitive grammar* is a phrase-structure grammar

$$G = (V, T, P, S)$$

such that every  $u \rightarrow v$  in  $P$  is of the form

$$u = x_1Ax_2, v = x_1yx_2$$

where  $x_1, x_2 \in V^*$ ,  $A \in N$ , and  $y \in V^+$ . A *context-sensitive language* is a language generated by a context-sensitive grammar. The family of context-sensitive languages is denoted by **CS**.  $\square$

The family of context-sensitive languages is also characterized by monotone phrase-structure grammars.

**Definition 2.3.6.** A *monotone phrase-structure grammar* is a phrase-structure grammar

$$G = (V, T, P, S)$$

such that  $u \rightarrow v \in P$  satisfies  $|u| \leq |v|$ . A *monotone recursively enumerable language* is a language generated by a monotone phrase-structure grammar. The family of monotone recursively enumerable languages is denoted by **MON**.  $\square$

**Definition 2.3.7.** A *context-free grammar* is a phrase-structure grammar

$$G = (V, T, P, S)$$

such that every rule in  $P$  is of the form

$$A \rightarrow x$$

where  $A \in N$  and  $x \in V^*$ . A *context-free language* is a language generated by a context-free grammar. The family of context-free languages is denoted by **CF**.  $\square$

**Definition 2.3.8.** A *linear grammar* is a phrase-structure grammar

$$G = (V, T, P, S)$$

such that every rule in  $P$  is of the form

$$A \rightarrow xBy \text{ or } A \rightarrow x$$

where  $A, B \in N$  and  $x, y \in T^*$ . A *linear language* is a language generated by a linear grammar. The family of linear languages is denoted by **LIN**.  $\square$

**Definition 2.3.9.** A *regular grammar* is a phrase-structure grammar

$$G = (V, T, P, S)$$

such that every rule in  $P$  is of the form

$$A \rightarrow aB \text{ or } A \rightarrow a$$

where  $A, B \in N$  and  $a \in T$ . A *regular language* is a language generated by a regular grammar. The family of regular languages is denoted by **REG**.  $\square$

Alternatively, the family of regular languages is characterized by right-linear grammars, defined next.

**Definition 2.3.10.** A *right-linear grammar* is a phrase-structure grammar

$$G = (V, T, P, S)$$

such that every rule in  $P$  is of the form

$$A \rightarrow xB \text{ or } A \rightarrow x$$

where  $A, B \in N$  and  $x \in T^*$ . A *right-linear language* is a language generated by a right-linear grammar. The family of right-linear languages is denoted by **RLIN**.  $\square$

**Definition 2.3.11.** Define the following abbreviations. Let  $PSG$ ,  $MONG$ ,  $CSG$ ,  $CFG$ ,  $LG$ ,  $RLG$ , and  $RG$ , denote phrase-structure, monotone phrase-structure, context-sensitive, context-free, linear, right-linear, and regular grammar, respectively. Let  $X$  denotes a specific type of grammar. Then,  $X^{-\varepsilon}$  denotes its propagating variant. Additionally, for a specific type of grammar  $Y$ ,  $\Gamma_Y$  denotes the set of all grammars of type  $Y$ .  $\square$

**Definition 2.3.12.** Let  $G = (V, T, P, S)$  be a phrase-structure grammar. To explicitly specify that  $G$  use a derivation relation  $_d \Rightarrow$  to generate  $L(G)$ , we write

$$L(G, _d \Rightarrow) = \{x \in T^* \mid S _d \Rightarrow^* x\}$$



and say that  $L(G, \Rightarrow_d)$  is the *language that  $G$  generates by using  $\Rightarrow_d$* . For any  $X \subseteq \Gamma_{PSG}$ , set

$$\mathcal{L}(X, \Rightarrow_d) = \{L(G, \Rightarrow_d) \mid G \in X\} \quad \square$$

**Definition 2.3.13.** Let  $G = (V, T, P, S)$  be any grammar. Let  $w \in T^*$  and

$$\alpha: S = w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n = w$$

be a derivation in  $G$ , for some  $n \geq 0$ . Then,

$$\begin{aligned} \text{Ind}(G, w, \alpha) &= \max(\{\#_N(w_i) \mid 0 \leq i \leq n\}) \\ \text{Ind}(G, w) &= \min(\{\text{Ind}(G, w, \alpha) \mid \alpha \text{ is a derivation of } w \text{ in } G\}) \\ \text{Ind}(G) &= \sup(\{\text{Ind}(G, w) \mid w \in L(G)\}) \end{aligned}$$

is *index of derivation  $D$  of the string  $w$  in  $G$* , *index of the string  $w$  in  $G$* , and *index of  $G$* , respectively. If there is a constant  $k \geq 1$  such that  $\text{Ind}(G) = k$ ,  $G$  is said to be of index  $k$ .  $\square$

Set  $\mathbf{CF}_k = \{L \mid L = L(G), G \text{ is CFG}, \text{Ind}(G) = k, k \geq 1\}$  and  $\mathbf{CF}_{fin} = \{L \mid L \in \mathbf{CF}_i, \text{ for some } i \geq 1\}$ . For further details concerning finite index of grammars, see Chapter 3 in [DP89].

To illustrate the above-introduced notation, let  $G = (V, T, P, S)$  be a RLG; then,  $L(G, \Rightarrow) = \{x \in T^* \mid S \Rightarrow^* x\}$ , and  $\mathcal{L}(\Gamma_{RLG}, \Rightarrow) = \{L(G, \Rightarrow) \mid G \in \Gamma_{RLG}\}$ . To give another example,  $\mathcal{L}(\Gamma_{CFG}, \Rightarrow)$  denotes the family of all context-free languages.

Notice that  $\mathbf{REG} = \mathbf{RLIN} = \mathcal{L}(\Gamma_{RLG}, \Rightarrow) = \mathcal{L}(\Gamma_{RG}, \Rightarrow)$ ,  $\mathbf{LIN} = \mathcal{L}(\Gamma_{LG}, \Rightarrow)$ ,  $\mathbf{CF} = \mathcal{L}(\Gamma_{CFG}, \Rightarrow)$ ,  $\mathbf{CS} = \mathbf{MON} = \mathcal{L}(\Gamma_{MONG}, \Rightarrow) = \mathcal{L}(\Gamma_{CSG}, \Rightarrow)$ , and  $\mathbf{RE} = \mathcal{L}(\Gamma_{PSG}, \Rightarrow)$ .

Concerning the families of finite, regular, right-linear, linear, context-free of finite index, context-free, context-sensitive, monotone recursively enumerable, and recursively enumerable languages, the next important theorem holds true.

**Theorem 2.3.14 (Chomsky Hierarchy, see [Cho56, Cho59]).**

$$\mathbf{FIN} \subset \mathbf{REG} = \mathbf{RLIN} \subset \mathbf{LIN} \subset \mathbf{CF}_{fin} \subset \mathbf{CF} \subset \mathbf{CS} = \mathbf{MON} \subset \mathbf{RE}$$

Next, we recall canonical derivations in context-free grammars.

**Definition 2.3.15.** Let  $G = (V, T, \Psi, P, S)$  be a context-free grammar. The relation of a *direct leftmost derivation*, denoted by  $\Rightarrow_{lm, G}$ , is defined as follows: if  $u \in T^*$ ,  $v \in V^*$ , and  $r: A \rightarrow x \in P$ , then

$$uAv \Rightarrow_{lm, G} uxv [r]$$

Let  $\text{lm} \Rightarrow_G^n$ ,  $\text{lm} \Rightarrow_G^*$ , and  $\text{lm} \Rightarrow_G^+$  denote the  $n$ th power of  $\text{lm} \Rightarrow_G$ , for some  $n \geq 0$ , the reflexive-transitive closure of  $\text{lm} \Rightarrow_G$ , and the transitive closure of  $\text{lm} \Rightarrow_G$ , respectively. The language that  $G$  generates by using leftmost derivations is denoted by  $L(G, \text{lm} \Rightarrow)$  and defined as

$$L(G, \text{lm} \Rightarrow) = \{w \in T^* \mid S \text{lm} \Rightarrow_G^* w\}$$

If  $S \text{lm} \Rightarrow_G^* w [\rho]$ , where  $w \in T^*$ , then  $\rho$  is the *left parse* of  $w$ .  $\square$

By analogy with leftmost derivations and left parses, we define rightmost derivations and right parses.

**Definition 2.3.16.** Let  $G = (V, T, \Psi, P, S)$  be a context-free grammar. The relation of a *direct rightmost derivation*, denoted by  $\text{rm} \Rightarrow_G$ , is defined as follows: if  $u \in V^*$ ,  $v \in T^*$ , and  $r: A \rightarrow x \in P$ , then

$$uAv \text{rm} \Rightarrow_G uxv [r]$$

Let  $\text{rm} \Rightarrow_G^n$ ,  $\text{rm} \Rightarrow_G^*$ , and  $\text{rm} \Rightarrow_G^+$  denote the  $n$ th power of  $\text{rm} \Rightarrow_G$ , for some  $n \geq 0$ , the reflexive-transitive closure of  $\text{rm} \Rightarrow_G$ , and the transitive closure of  $\text{rm} \Rightarrow_G$ , respectively. The language that  $G$  generates by using rightmost derivations is denoted by  $L(G, \text{rm} \Rightarrow_G)$  and defined as

$$L(G, \text{rm} \Rightarrow) = \{w \in T^* \mid S \text{rm} \Rightarrow_G^* w\}$$

If  $S \text{rm} \Rightarrow_G^* w [\rho]$ , where  $w \in T^*$ , then  $\rho$  is the *right parse* of  $w$ .  $\square$

Without any loss of generality, in context-free grammars, we may consider only canonical derivations, which is formally stated in the following theorem.

**Theorem 2.3.17 (See [Med00a]).** *Let  $G$  be a context-free grammar. Then,*

$$L(G, \text{lm} \Rightarrow) = L(G, \text{rm} \Rightarrow) = L(G)$$

The following theorem gives a characterization of the family of recursively enumerable languages by context-free languages.

**Theorem 2.3.18 (See [GGH67]).** *For every recursively enumerable language  $K$ , there exist two context-free languages,  $L_1$  and  $L_2$ , and a homomorphism  $h$  such that*

$$K = h(L_1 \cap L_2)$$

The next theorem says that if a phrase-structure grammar generates each of its sentences by a derivation satisfying a length-limited condition, then the generated language is, in fact, context sensitive.

**Theorem 2.3.19 (Workspace Theorem, see [Sal73]).** *Let  $G = (V, T, P, S)$  be a phrase-structure grammar. If there is a positive integer  $k$  such that for every nonempty  $y \in L(G)$ , there exists a derivation*

$$D: S \Rightarrow_G x_1 \Rightarrow_G x_2 \Rightarrow_G \cdots \Rightarrow_G x_n = y$$

*where  $x_i \in V^*$  and  $|x_i| \leq k|y|$ , for all  $i = 1, 2, \dots, n$ , for some  $n \geq 1$ , then  $L(G) \in \mathbf{CS}$ .*

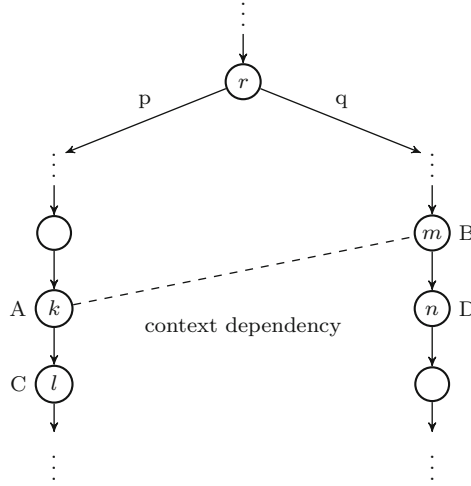
### 2.3.2 How to Prove Context-Freeness

Formal language theory has always intensively struggled to establish conditions under which phrase-structure grammars generate a proper subfamily of the family of recursively enumerable languages because results like this often significantly simplify proofs that some languages are members of the subfamily. To illustrate, consider the well-known workspace theorem for phrase-structure grammars, which fulfills a crucially important role in the grammatically oriented theory of formal languages as a whole (see Theorem III.10.1 in [Sal73]). This theorem represents a powerful tool to demonstrate that if a phrase-structure grammar  $H$  generates each of its sentences by a derivation satisfying a prescribed condition (specifically, this condition requires that there is a positive integer  $k$  such that  $H$  generates every sentence  $y$  in the generated language  $L(H)$  by a derivation in which every sentential form  $x$  satisfies  $|x| \leq k|y|$ ), then  $L(H)$  is a member of the context-sensitive language family. Regarding the membership in the context-free language family, however, formal language theory lacks a result like this. To fill this gap, the present section establishes a tree-based condition so every phrase-structure grammar satisfying this condition generates a member of the context-free language family.

To give an insight into this result, we first sketch some terminology. Recall that a phrase-structure grammar  $G$  is in Kuroda normal form (see Definition 3.1.1) if any rule satisfies one of these forms

$$AB \rightarrow CD, A \rightarrow BC, A \rightarrow B, A \rightarrow a, \text{ or } A \rightarrow \varepsilon$$

where  $A, B, C, D$  are nonterminals,  $a$  is a terminal, and  $\varepsilon$  is the empty string. We define the notion of a derivation tree  $t$  graphically representing a derivation in  $G$  by analogy with this notion in terms of an ordinary context-free grammar (see Definition 6.8 on page 92 in [Med14]). In addition, however, we introduce context-dependent pairs of nodes in  $t$  as follows. In  $t$ , two paths are neighboring if no other path occurs between them. Let  $p$  and  $q$  be two neighboring paths in  $t$ . Let  $p$  contain a node  $k$  with a single child  $l$ , where  $k$  and  $l$  are labelled with  $A$  and  $C$ , respectively, and let  $q$  contain a node  $m$  with a single child  $n$ , where  $m$  and  $n$  are labelled with  $B$  and  $D$ , respectively. Let this four-node portion of  $t$ ; consisting of  $k, l, m$ , and  $n$ ; graphically represents an application of  $AB \rightarrow CD$ . Then,  $k$  and  $m$  are a context-dependent pair of nodes (see Fig. 2.1).



**Fig. 2.1** Illustration of context dependency in  $t$

As its main result, the present section proves that the language of  $G$ ,  $L(G)$ , is context-free if there is a constant  $k$  such that every  $w \in L(G)$  is the frontier of a derivation tree  $d$  in which any pair of neighboring paths contains  $k$  or fewer context-dependent pairs of nodes. Apart from its theoretical value, this result may be of some interest in practice, too. Specifically, some language processors, such as compiler parsers, frequently require that the languages processed by them are context-free. As obvious, the result stated above may fulfill a useful role during the verification of this requirement.

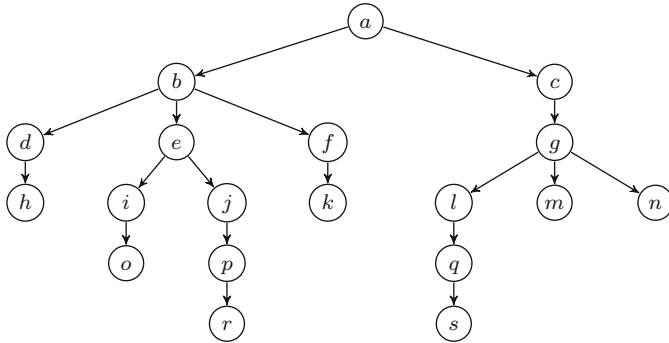
The section is organized as follows. First, we give all the necessary terminology. Then, we establish the main result of this section. Finally, we close this section by showing an application perspective of the main result.

### Definitions and Examples

**Definition 2.3.20.** Let  $t = (V, E)$  be a tree. Define a partial order relation  $<$  over  $V$  as follows. For a path  $\alpha = (m_0, m_1, \dots, m_k)$ , where  $m_0 = \text{troot}(t)$ ,  $m_i < m_k$ ,  $0 \leq i \leq k-1$ . An ordered tree is called *labelled*, if there exists a set of labels  $\mathcal{L}$  and a total mapping  $l : V \rightarrow \mathcal{L}$ . In what follows we substitute a node of a tree by its label if there is no risk of confusion.

Let  $t$  be an ordered tree with a node  $o$ . Let  $\alpha = (o, m_1, m_2, \dots, m_r)$  and  $\beta = (o, n_1, n_2, \dots, n_s)$  be two paths in  $t$ , for some  $r, s \geq 1$ , such that  $o$  is the parent of  $m_1$  and  $n_1$ , where

- (1)  $m_1$  is the direct left sibling of  $n_1$ ;
- (2)  $m_i$  is the rightmost child of  $m_{i-1}$ , and  $n_j$  is the leftmost child of  $n_{j-1}$ ,  $2 \leq i \leq r$ ,  $2 \leq j \leq s$ .



**Fig. 2.2** Labelled ordered tree  $t$

Then,  $\alpha$  and  $\beta$  are two *neighboring paths* in  $t$ ,  $\alpha$  is a *left neighboring path* to  $\beta$ , and  $\beta$  is a *right neighboring path* to  $\alpha$ .  $\square$

Let us demonstrate the tree-related notions by the following example.

**Example 2.3.21.** The following graph (Fig. 2.2.) represents labelled ordered tree  $t$ . The root node  $\text{troot}(t)$  is labelled  $a$ . It has no parent and two children  $b$  and  $c$ . Then,  $b$  is a sibling of  $c$  and  $c$  is a sibling of  $b$ . The leftmost child of  $b$  is  $d$ , while the rightmost is  $f$ . The node  $d$  is a left sibling of  $f$ , however, it is not the direct left sibling, which is  $e$ . The node  $f$  is the parent of  $k$ , but  $k$  has no child, so it is a leaf node.  $\text{horksmn} = \text{frontier}(t)$ . Consider the node  $e$ . The nodes  $a$  and  $b$  are predecessors of  $e$ , while  $i, j, o, p$ , and  $r$  are  $e$ 's descendants. The nodes  $c$  or  $d$  are not in predecessor relation with  $e$ , since they are neither predecessors of  $e$ , nor descendants of  $e$ . The sequence of nodes  $bejpr$  is a path in  $t$ . The path  $bfk$  is neighboring to  $bejpr$ ; unlike  $abfk$ ,  $eio$ , or  $bdh$ .  $\square$

**Definition 2.3.22.** Let  $G = (V, T, P, S)$  be a phrase-structure grammar.  $G$  is in the *binary form* if any  $p \in P$  has one of these forms,

$$AB \rightarrow CD, A \rightarrow BC, A \rightarrow X$$

where  $A, B, C, D \in N$ ,  $X \in V \cup \{\varepsilon\}$ . In what follows, unless explicitly stated otherwise, we automatically assume that every PSG is in the binary form.  $\square$

**Theorem 2.3.23.** A language  $L$  is context-sensitive iff  $L = L(G)$ , where  $G$  is a monotone phrase-structure grammar in the binary form.

*Proof.* A language  $L$  is context-sensitive iff  $L$  is generated by monotone phrase-structure grammar in Kuroda normal form (see Definition 3.1.1). Every monotone PSG in Kuroda normal form is a special case of a monotone PSG in the binary form.  $\square$

**Theorem 2.3.24.** A language  $L$  is recursively enumerable iff  $L = L(G)$ , where  $G$  is a phrase-structure grammar in the binary form.

*Proof.* A language  $L$  is recursively enumerable iff  $L$  is generated by phrase-structure grammar. Every PSG can be converted to the binary form (see Chapter 4 in [RS97a]).  $\square$

**Definition 2.3.25.** Let  $G = (V, T, P, S)$  be a PSG in the binary form.

- (1) For  $p: A \rightarrow x \in P$ ,  $A\langle x \rangle$  is the *rule tree* that represents  $p$ .
- (2) The *derivation trees* representing derivations in  $G$  are defined recursively as follows:
  - (a) One-node tree with a node labelled  $X$  is the derivation tree corresponding to  $X \Rightarrow^0 X$  in  $G$ , where  $X \in V$ . If  $X = \varepsilon$ , we refer to the node labeled  $X$  as  $\varepsilon$ -node ( $\varepsilon$ -leaf); otherwise, we call it *non- $\varepsilon$ -node* (*non- $\varepsilon$ -leaf*).
  - (b) Let  $d$  be the derivation tree with  $\text{frontier}(d) = uAv$  representing  $X \Rightarrow^* uAv$  and let  $p: A \rightarrow x \in P$ . The derivation tree that represents

$$X \Rightarrow^* uAv [\rho] \Rightarrow uxv [p]$$

is obtained by replacing the  $i$ th non- $\varepsilon$ -leaf in  $d$  labelled  $A$ , with rule tree corresponding to  $p$ ,  $A\langle x \rangle$ , where  $i = |uA|$ .

- (c) Let  $d$  be the derivation tree representing  $X \Rightarrow^* uABv [\rho]$  with  $\text{frontier}(d) = uABv$ , and let  $p: AB \rightarrow CD \in P$ . The derivation tree that represents

$$X \Rightarrow^* uABv [\rho] \Rightarrow uCDv [p]$$

is obtained by replacing the  $i$ th and  $(i + 1)$ th non- $\varepsilon$ -leaf in  $d$  labelled  $A$  and  $B$  with  $A\langle C \rangle$  and  $B\langle D \rangle$ , respectively, where  $i = |uA|$ .

- (3) A *derivation tree* in  $G$  is any tree  $t$  for which there is a derivation represented by  $t$  (see (2) in this definition).

Note, after replacement in (c), the nodes  $A$  and  $B$  are the parents of the new leaves  $C$  and  $D$ , respectively, and we say that  $A$  and  $B$  are *context-dependent*, alternatively speaking, we say that there is a context dependency between  $A$  and  $B$ . In a derivation tree, two nodes are *context-independent* if they are not context-dependent.

Then, for any  $p: A \rightarrow x \in P$ ,  ${}_G\Delta(p)$  denotes rule tree corresponding to  $p$ . For any  $A \Rightarrow^* x [\rho]$  in  $G$ , where  $A \in N$ ,  $x \in V^*$ , and  $\rho \in P^*$ ,  ${}_G\Delta(A \Rightarrow^* x [\rho])$  denotes the derivation tree corresponding to  $A \Rightarrow^* x [\rho]$ . Just like we often write  $A \Rightarrow^* x$  instead of  $A \Rightarrow^* x [\rho]$ , we sometimes simplify  ${}_G\Delta(A \Rightarrow^* x [\rho])$  to  ${}_G\Delta(A \Rightarrow^* x)$  in what follows if there is no danger of confusion. Let  ${}_G\blacktriangle$  denotes the set of all derivation trees in  $G$ . Finally, by  ${}_G\Delta_x \in {}_G\blacktriangle$ , we mean a derivation tree whose frontier is  $x$ , where  $x \in F(G)$ .

If a node is labelled with a terminal, it is called a *terminal node*. If a node is labelled with a nonterminal, it is called a *nonterminal node*.

Let  $\alpha = (o, m_1, m_2, \dots, m_r)$  and  $\beta = (o, n_1, n_2, \dots, n_s)$  be two neighboring paths, where  $r, s \geq 0$ ,  $\alpha$  is the left neighboring path to  $\beta$ , and  $m_r$  and  $n_s$  are terminal nodes. Then, there is a  $t$ -tuple  $\gamma = (g_1, g_2, \dots, g_t)$  of nodes from  $\alpha$  and  $t$ -tuple  $\delta = (h_1, h_2, \dots, h_t)$  of nodes from  $\beta$ , where  $g_p < g_q$ , for  $1 \leq p < q \leq t$ ,  $t < \min(r, s)$ , and  $g_i$  and  $h_i$  are context-dependent, for  $1 \leq i \leq t$ . Let  $\rho = p_1 p_2 \dots p_t$  be a string of non-context-free rules corresponding to context dependencies between  $\gamma$  and  $\delta$ . We call  $\rho$  the *right context of  $\alpha$*  and the *left context of  $\beta$*  or the *context of  $\alpha$  and  $\beta$* . Consider a node  $m_i$ , where  $1 \leq i \leq r$ , and two  $(t - k + 1)$ -tuples of nodes  $\sigma = (g_k, g_{k+1}, \dots, g_t)$  and  $\varphi = (h_k, h_{k+1}, \dots, h_t)$ , where  $k$  is a minimal integer such that  $m_i < g_k$ . Then, a string of non-context-free rules  $\tau = p_k p_{k+1} \dots p_t$  corresponding to context dependencies between  $\sigma$  and  $\varphi$  is called the *right descendant context of  $m_i$* , for some  $1 \leq k \leq t$ . Analogously, we define the notion of the *left descendant context of a node  $n_j$  in  $\beta$* , for some  $1 \leq j \leq s$ .  $\square$

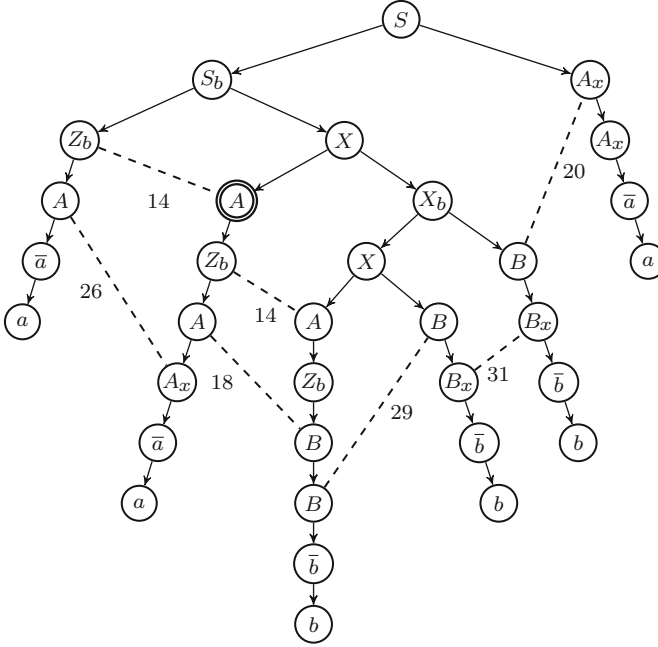
*Example 2.3.26.* Let  $G = (V, T, P, S)$  be a phrase-structure grammar, where  $V = \{S, S_a, S_b, X, X_a, X_b, Z_a, Z_b, A, 1, 2, 3, A_x, \bar{a}, a, B, B_x, \bar{b}, b\}$ ,  $T = \{a, b\}$ , and  $P$  contains the following rules:

- |                             |                                |   |  |
|-----------------------------|--------------------------------|---|--|
| (1) $S \rightarrow S_a B_x$ | (9) $X \rightarrow B X_a$      | (17) $Z_b \rightarrow B$                | (25) $A_x \rightarrow \bar{a}$             |
| (2) $S \rightarrow S_b A_x$ | (10) $X_a \rightarrow X A$     | (18) $AB \rightarrow A_x B$             | (26) $AA_x \rightarrow \bar{a} \bar{a}$    |
| (3) $S_a \rightarrow Z_a X$ | (11) $X_b \rightarrow X B$     | (19) $BA \rightarrow B_x A$             | (27) $1A_x \rightarrow \bar{a} \bar{a}$    |
| (4) $S_b \rightarrow Z_b X$ | (12) $Z_a A \rightarrow A Z_a$ | (20) $BA_x \rightarrow B_x A_x$         | (28) $2A_x \rightarrow \bar{a} \bar{a}$    |
| (5) $X \rightarrow XX$      | (13) $Z_a B \rightarrow B Z_a$ | (21) $AA \rightarrow \bar{a} 1$         | (29) $BB \rightarrow \bar{b} B_x$          |
| (6) $X \rightarrow AB$      | (14) $Z_b A \rightarrow A Z_b$ | (22) $1A \rightarrow \bar{a} 2$         | (30) $B_x B \rightarrow \bar{b} B_x$       |
| (7) $X \rightarrow BA$      | (15) $Z_b B \rightarrow B Z_b$ | (23) $2A \rightarrow \bar{a} 3$         | (31) $B_x B_x \rightarrow \bar{b} \bar{b}$ |
| (8) $X \rightarrow A X_b$   | (16) $Z_a \rightarrow A$       | (24) $3A_x \rightarrow \bar{a} \bar{a}$ | (32) $\bar{a} \rightarrow a$               |
|                             |                                |   | (33) $\bar{b} \rightarrow b$               |

At this point, let us make only an informal observation that  $L(G)$  is the language of all nonempty strings above  $T$  consisted of an equal number of  $a$ s and  $b$ s, where every sequence of  $a$ s is of a length between 1 and 5 and every sequence of  $b$ s is longer or equal 3. A rigorous proof comes later.

The string  $aabbba$  can be obtained by the following derivation:

$$\begin{array}{llll}
 S \Rightarrow S_b A_x & [(2)] & \Rightarrow Z_b X A_x & [(4)] \\
 \Rightarrow Z_b A X_b A_x & [(8)] & \Rightarrow Z_b A X B A_x & [(11)] \\
 \Rightarrow Z_b A A B B A_x & [(6)] & \Rightarrow A Z_b A B B A_x & [(14)] \\
 \Rightarrow A A Z_b B B A_x & [(14)] & \Rightarrow A A B B B A_x & [(17)] \\
 \Rightarrow A A_x B B B A_x & [(18)] & \Rightarrow A A_x B B B_x A_x & [(20)] \\
 \Rightarrow \bar{a} \bar{a} B B B_x A_x & [(26)] & \Rightarrow \bar{a} a b B_x B_x A_x & [(29)] \\
 \Rightarrow a a b b b A_x & [(31)] & \Rightarrow a a b b b a & [(25)] \\
 \Rightarrow a a b b b a & [(32)] & \Rightarrow a a b b b a & [(32)] \\
 \Rightarrow a a b b b a & [(33)] & \Rightarrow a a b b b a & [(33)] \\
 \Rightarrow a a b b b \bar{a} & [(33)] & \Rightarrow a a b b b a & [(32)]
 \end{array}$$



**Fig. 2.3**  $_G \Delta_{aabbba}$

A graph representing  $_G \Delta(S \Rightarrow^* aabbba)$  is illustrated in Fig. 2.3.

Let us note that dashed lines, numbers, and double circle contour only denote the context dependencies, applied non-context-free rules, and a specific node, respectively, and are not the part of the derivation tree.

Pairs of context-dependent nodes are linked with dashed lines, all the other nodes are context-independent. Since  $aabbba = \text{frontier}(_G \Delta_{aabbba})$ , all the leafs are terminal nodes. Every other node is nonterminal node. For a pair of neighboring paths  $\alpha = S_b Z_b A \bar{a} a$  and  $\beta = S_b X A Z_b A A_x \bar{a} a$ , a string  $\rho = 14\ 26$  is their context, it is the left context of  $\beta$  and the right context of  $\alpha$ . Consider the double circled node  $A$ . Then,  $\tau = 26$  is the left descendant context of  $A$  and  $\varphi = 14\ 18$  is the right descendant context of  $A$ .  $\square$

## Results

**Theorem 2.3.27.** *A language  $L$  is context-free iff there is a constant  $k \geq 0$  and a phrase-structure grammar  $G$  such that  $L = L(G)$  and for every  $x \in L(G)$ , there is a tree  $_G \Delta_x \in _G \blacktriangle$  that satisfies:*

- (1) *any two neighboring paths contain no more than  $k$  pairs of context-dependent nodes;*
- (2) *out of neighboring paths, every pair of nodes is context-independent.*



*Proof. Construction.* Consider any  $k \geq 0$ . Let  $G = (V, T, P, S)$  be a PSG such that  $L(G) = L$ . Set  $N = V - T$ . Let  $P_{cs} \subseteq P$  denote the set of all non-context-free rules of  $G$ . Set

$$N' = \{A_{l|r} \mid A \in N, l, r \in (P_{cs} \cup \{\varepsilon\})^k\}$$

Construct a grammar  $G' = (V', T, P', S_{\varepsilon|\varepsilon})$ , where  $V' = N' \cup T$ . Set  $P' = \emptyset$ . Construct  $P'$  by performing (I) through (IV) given next.

- (I) For all  $A \rightarrow B \in P, A, B \in N$ , and  $l, r \in (P_{cs} \cup \{\varepsilon\})^k$ , add  $A_{l|r} \rightarrow B_{l|r}$  to  $P'$ ;
- (II) for all  $A \rightarrow a \in P, A \in N, a \in (T \cup \{\varepsilon\})$ , add  $A_{\varepsilon|\varepsilon} \rightarrow a$  to  $P'$ ;
- (III) for all  $A \rightarrow BC \in P$ , where  $A, B, C \in N$ , and  $r, l, x \in (P_{cs} \cup \{\varepsilon\})^k$ , add  $A_{l|r} \rightarrow B_{l|x}C_{x|r}$  to  $P'$ ;
- (IV) for all  $p: AB \rightarrow CD \in P, A, B, C, D \in N, x, z \in (P_{cs} \cup \{\varepsilon\})^k$ , and  $y \in (P_{cs} \cup \{\varepsilon\})^{k-1}$ , add  $A_{x|py} \rightarrow C_{x|y}$  and  $B_{py|z} \rightarrow D_{y|z}$  to  $P'$ .

*Basic Idea.* Notice nonterminal symbols. Since every pair of neighboring paths of  $G$  contains a limited number of context-dependent nodes, all of its context-dependencies are encoded in nonterminals.  $G'$  nondeterministically decides about all context-dependencies while introducing a new pair of neighboring paths by rules (III). A new pair of neighboring paths is introduced with every application of

$$A_{l|r} \rightarrow B_{l|x}C_{x|r}$$

where  $x$  encodes a new descendant context. Context dependencies are realized later by context-free rules (IV).

Since  $P'$  contains no non-context-free rule,  $G'$  is context-free. Next, we proof  $L(G) = L(G')$  by establishing Claims 2.3.28 through 2.3.30. Define the new homomorphism  $\gamma : V' \rightarrow V$ ,  $\gamma(A_{l|r}) = A$ , for  $A_{l,r} \in N'$ , and  $\gamma(a) = a$  otherwise.

*Claim 2.3.28.* If  $S \Rightarrow^m w$  in  $G$ , where  $m \geq 0$  and  $w \in V^*$ , then  $S_{\varepsilon|\varepsilon} \Rightarrow^* w'$  in  $G'$ , where  $w' \in V'^*$  and  $\gamma(w') = w$ .

*Proof.* We prove this by induction on  $m \geq 0$ .

*Basis.* Let  $m = 0$ . That is  $S \Rightarrow^0 S$  in  $G$ . Clearly,  $S_{\varepsilon|\varepsilon} \Rightarrow^0 S_{\varepsilon|\varepsilon}$  in  $G'$ , where  $\gamma(S_{\varepsilon|\varepsilon}) = S$ , so the basis holds.

*Induction Hypothesis.* Suppose that there exists  $n \geq 0$  such that Claim 2.3.28 holds for all  $0 \leq m \leq n$ .

*Induction Step.* Let  $S \Rightarrow^{n+1} w$  in  $G$ . Then,  $S \Rightarrow^n v \Rightarrow w$ , where  $v \in V^*$ , and there exists  $p \in P$  such that  $v \Rightarrow w[p]$ . By the induction hypothesis,  $S_{\varepsilon|\varepsilon} \Rightarrow^* v'$ , where  $\gamma(v') = v$ , in  $G'$ . Next, we consider the following four forms of  $p$ .

- (I) Let  $p: A \rightarrow B \in P$ , for some  $A, B \in N$ . Without any loss of generality, suppose  $l$  and  $r$  are a left descendant context and a right descendant context of  $A$ . By the construction of  $G'$ , there exists a rule  $p': A_{l|r} \rightarrow B_{l|r} \in P'$ . Then, there exists a derivation  $v' \Rightarrow w'[p']$  in  $G'$ , where  $\gamma(w') = w$ .

- (II) Let  $p: A \rightarrow a \in P$ , for some  $A \in N$  and  $a \in T \cup \{\varepsilon\}$ . Since  $a$  is a terminal node, it has empty descendant contexts. By the construction of  $G'$ , there exists a rule  $p': A_{\varepsilon|\varepsilon} \rightarrow a \in P'$ . Then, there exists a derivation  $v' \Rightarrow w' [p']$  in  $G'$ , where  $\gamma(w') = w$ .
- (III) Let  $p: A \rightarrow BC \in P$ , for some  $A, B, C \in N$ . Without any loss of generality, suppose  $l$  and  $r$  are a left descendant context and a right descendant context of  $A$ , and  $x \in (P_{cs} \cup \{\varepsilon\})^k$  is a context of neighboring paths beginning at this node. By the construction of  $G'$ , there exists a rule  $p': A_{l|r} \rightarrow B_{l|x}C_{x|r} \in P'$ . Then, there exists a derivation  $v' \Rightarrow w' [p']$  in  $G'$ , where  $\gamma(w') = w$ .
- (IV) Let  $p: AB \rightarrow CD \in P$ , for some  $A, B, C, D \in N$ . By the assumption stated in Theorem 2.3.27,  $A$  and  $B$  occur in two neighboring paths denoted by  $\alpha$  and  $\beta$ , respectively. Without any loss of generality, suppose that a context of  $\alpha$  and  $\beta$  is a string  $c \in (P_{cs} \cup \varepsilon)^k$ , where  $c = pc_f$ , and  $l$  is a left descendant context,  $r$  is a right descendant context of  $A, B$ , respectively. By the construction of  $G'$ , there exist two rules

$$p'_l: A_{l|pc_f} \rightarrow C_{l|c_f}, \quad p'_r: B_{pc_f|r} \rightarrow D_{c_f|r} \in P'$$

Then, there exists a derivation  $v' \Rightarrow^2 w' [p'_l p'_r]$  in  $G'$ , where  $\gamma(w') = w$ .

Notice (IV). The preservation of the context is achieved by nonterminal symbols. Since the stored context is reduced symbol by symbol from left to right direction in both  $\alpha$  and  $\beta$ ,  $G'$  simulates the applications of non-context-free rules of  $G$ .

We covered all possible forms of  $p$ , so the claim holds.  $\square$

*Claim 2.3.29.* Every  $x \in F(G')$  can be derived in  $G'$  as follows.

$$S_{\varepsilon|\varepsilon} = x_0 \Rightarrow^{d_1} x_1 \Rightarrow^{d_2} x_2 \Rightarrow^{d_3} \dots \Rightarrow^{d_{h-1}} x_{h-1} \Rightarrow^{d_h} x_h = x$$

for some  $h \geq 0$ , where  $d_i \in \{1, 2\}$ ,  $1 \leq i \leq h$ , so that

(1) if  $d_i = 1$ , then  $x_{i-1} = uA_{l|r}v$ ,  $x_i = uzv$ ,  $x_{i-1} \Rightarrow x_i [A_{l|r} \rightarrow z]$ , where  $u, v \in V'^*$ ,  $z \in \{B_{l|r}, C_{l|x}D_{x|r}, a\}$ , for some  $A_{l|r}, B_{l|r}, C_{l|x}, D_{x|r} \in N'$ ,  $a \in (T \cup \{\varepsilon\})$ ;

(2) if  $d_i = 2$ , then  $x_{i-1} = uA_{x|py}B_{py|z}v$ ,  $x_i = uC_{x|y}D_{y|z}v$ , and

$$uA_{x|py}B_{py|z}v \Rightarrow uC_{x|y}B_{py|z}v [A_{x|py} \rightarrow C_{x|y}] \Rightarrow uC_{x|y}D_{y|z}v [B_{py|z} \rightarrow D_{y|z}]$$

for some  $u, v \in V'^*$  and  $A_{x|py}, B_{py|z}, C_{x|y}, D_{y|z} \in N'$ .

*Proof.* Since  $G'$  is context-free, without any loss of generality in every derivation of  $G'$  we can always reorder applied rules to satisfy Claim 2.3.29.  $\square$

*Claim 2.3.30.* Let  $S_{\varepsilon|\varepsilon} \Rightarrow^{d_1} x_1 \Rightarrow^{d_2} \dots \Rightarrow^{d_{m-1}} x_{m-1} \Rightarrow^{d_m} x_m$  in  $G'$  be a derivation that satisfies Claim 2.3.29, for some  $m \geq 0$ . Then,  $S \Rightarrow^* w$  in  $G$ , where  $\gamma(x_m) = w$ .

*Proof.* We prove this by induction on  $m \geq 0$ .

*Basis.* Let  $m = 0$ . That is  $S_{\varepsilon|\varepsilon} \Rightarrow^0 S_{\varepsilon|\varepsilon}$  in  $G'$ . Clearly,  $S \Rightarrow^0 S$  in  $G$ . Since  $\gamma(S_{\varepsilon|\varepsilon}) = S$ , the basis holds.

*Induction Hypothesis.* Suppose that there exists  $n \geq 0$  such that Claim 2.3.30 holds for all  $0 \leq m \leq n$ .

*Induction Step.* Let  $S_{\varepsilon|\varepsilon} \Rightarrow^{d_1} x_1 \Rightarrow^{d_2} \dots \Rightarrow^{d_{n-1}} x_{n-1} \Rightarrow^{d_n} x_n \Rightarrow^{d_{n+1}} x_{n+1}$  in  $G'$  be a derivation that satisfies Claim 2.3.29. By the induction hypothesis,  $S \Rightarrow^* v$ ,  $v \in V^*$ , where  $\gamma(x_n) = v$ , in  $G$ . Divide the proof into two parts according to  $d_{n+1}$ .

(A) Let  $d_{n+1} = 1$ . By the construction of  $G'$ , there exists a rule  $p' \in P'$  such that  $x_n \Rightarrow^{d_{n+1}} x_{n+1} [p']$ . Next, we consider the following three forms of  $p'$ .

- (A.I) Let  $p': A_{l|r} \rightarrow B_{l|r} \in P'$ , for some  $A, B \in N$  and  $l, r \in (P_{cs} \cup \{\varepsilon\})^k$ . By the construction of  $G'$ , rule  $p'$  was introduced by some rule  $p: A \rightarrow B \in P$ . Then, there exists a derivation  $v \Rightarrow w [p]$ , where  $\gamma(x_{n+1}) = w$ .
- (A.II) Let  $p': A_{\varepsilon|\varepsilon} \rightarrow a \in P'$ , for some  $A \in N$  and  $a \in T \cup \{\varepsilon\}$ . By the construction of  $G'$ , rule  $p'$  was introduced by some rule  $p: A \rightarrow a \in P$ . Then, there exists a derivation  $v \Rightarrow w [p]$ , where  $\gamma(x_{n+1}) = w$ .
- (A.III) Let  $p': A_{l|r} \rightarrow B_{l|x} C_{x|r} \in P'$ , for some  $A, B, C \in N$  and  $l, r, x \in (P_{cs} \cup \{\varepsilon\})^k$ . By the construction of  $G'$ , rule  $p'$  was introduced by some rule  $p: A \rightarrow BC \in P$ . Then, there exists a derivation  $v \Rightarrow w [p]$ , where  $\gamma(x_{n+1}) = w$ .

(B) Let  $d_{n+1} = 2$ . Then,  $x_n \Rightarrow^{d_{n+1}} x_{n+1}$  is equivalent to

$$u_1 A_{x|py} B_{py|z} u_2 \Rightarrow u_1 C_{x|y} B_{py|z} u_2 [p'_1] \Rightarrow u_1 C_{x|y} D_{y|z} u_2 [p'_2]$$

where  $x_n = u_1 A_{x|py} B_{py|z} u_2$ ,  $x_{n+1} = u_1 C_{x|y} D_{y|z} u_2$ , and

$$p'_1: A_{x|py} \rightarrow C_{x|y}, p'_2: B_{py|z} \rightarrow D_{y|z} \in P'$$

for some  $u_1, u_2 \in V'^*$  and  $A_{x|py}, B_{py|z}, C_{x|y}, D_{y|z} \in N'$ . By the construction of  $G'$ , rules  $p'_1$  and  $p'_2$  were introduced by some rule  $p: AB \rightarrow CD \in P$ . Then, there exists a derivation  $v \Rightarrow w [p]$ , where  $\gamma(x_{n+1}) = w$ .

We covered all possibilities, so the claim holds.  $\square$

By Claims 2.3.28 and 2.3.30,  $S \Rightarrow^* w$  in  $G$  iff  $S_{\varepsilon|\varepsilon} \Rightarrow^* w'$  in  $G'$ , where  $\gamma(w') = w$ . If  $S \Rightarrow^* w$  in  $G$  and  $w \in T^*$ , then  $w \in L(G)$ . Since  $\gamma(w') = w' = w$ , for  $w \in T^*$ ,  $w' \in L(G')$ . Therefore,  $L(G) = L(G')$  and Theorem 2.3.27 holds.  $\square$

Consider Theorem 2.3.27. Observe that the second condition is superfluous whenever  $G$  is monotone. Since a grammar is in the binary form and no symbol can be erased, all context dependencies are within pairs of neighboring paths.

**Theorem 2.3.31.** *A language  $L$  is context-free iff there is a constant  $k \geq 0$  and a monotone phrase-structure grammar  $G$  such that  $L = L(G)$  and for every  $x \in L(G)$ , there is a tree  ${}_G \Delta_x \in {}_G \blacktriangle$ , where any two neighboring paths contain no more than  $k$  pairs of context-dependent nodes.*

*Proof.* Prove this by analogy with the proof of Theorem 2.3.27.  $\square$

## Use

We close this section explaining how to apply the results achieved in the previous section in order to demonstrate the contextfreeness of a language,  $L$ . As a rule, this demonstration follows the next three-step proof scheme.

- (1) Construct a phrase-structure grammar  $G$  in the binary form.
- (2) Prove  $L(G) = L$ .
- (3) Prove that  $G$  satisfies conditions from Theorems 2.3.27 or 2.3.31, depending on whether  $G$  is monotone.

Reconsider the grammar  $G$  from Example 2.3.26. Following the proof scheme sketched above, we next prove that  $L(G) \in \mathbf{CF}$ .

Consider  $G$  constructed in Example 2.3.26. Next, we show that for  $G$ ,

$$L(G) = \{w \in (A \cup \{\varepsilon\})(BA)^*(B \cup \{\varepsilon\}) \mid \#_a(w) = \#_b(w), \\ A = \{a^i \mid 1 \leq i \leq 5\}, B = \{b^i \mid i \geq 3\}, \text{ and } |w| > 0\}$$

Without any loss of generality, every terminal derivation of  $G$  can be divided into the following 5 phases, where each rule may be used only in a specific phase:

- (a) (1)–(4) (b) (5)–(11) (c) (12)–(17) (d) (18)–(31) (e) (32)–(33)

Next, we describe these phases in a greater detail.

- (a) First, we generate one of the following two strings by rules (1) through (4).

$$Z_a X B_x, Z_b X A_x$$

Possibly applicable rule (25) may be postponed for phase (d) without affecting the derivation, since rules in the previous phases cannot rewrite  $A_x$ .

- (b) The rules (5) through (11) are the only with  $X$ ,  $X_a$ , or  $X_b$  on their left-hand sides, therefore we can group all their applications in a sequence to get a sentential form from

$$\{Z_a, Z_b\}\{A, B\}^*\{A_x, B_x\}$$

- (c) The rules (12) through (17) possibly shift  $Z_a$  or  $Z_b$  to the right and rewrite it to  $A$  or  $B$ , respectively. Since these rules are the only with  $Z_a$ ,  $Z_b$  on their left-hand sides, they can be always prioritized before the rest of rules without any loss of generality.

$$\{A, B\}^*\{A_x, B_x\}$$

- (d) All the remaining rules may be applied in this phase. However, we can exclude rules (32) and (33), so we get a sentential form from

$$\{\bar{a}, \bar{b}\}^*$$

- (e) Since rules (32) and (33) are context-free and produce terminal symbols, they can be always postponed until the end of any successful derivation.

$$\{a, b\}^* = T^*$$

Let us add a few remarks concerning (a) through (e).

Phase (a) is very straightforward. Only notice that it is decided whether the generated string finally ends with  $a$  or  $b$  and the paired symbol is stored in  $Z_a$  or  $Z_b$  for phase (c).

In phase (b) an arbitrary string of  $A$ s and  $B$ s is generated from the initial symbol  $X$ . However, for every  $A$ , one  $B$  is generated and vice versa, so their numbers are always kept equal.

In phase (a) the grammar decides about the last symbol and stores the paired one, which, however, need not to be the first one. Therefore phase (c) determines its final position, while possibly shifting it to the right and finally rewriting it to  $A$  or  $B$ .

Phase (d) is the most tricky. It starts with a sentential form  $wc$ , where  $w \in \{A, B\}^*$ ,  $c \in \{A_x, B_x\}$ . Informally speaking, it consists of the sequences of  $A$ s which should be at most 5 symbols long, and  $B$ s which should be at least 3 symbols long. Rules (18) through (31) are designed to ensure these restrictions. To give an example, suppose  $wc$  is as follows.

$$wc = AAAABBBBABBBAA_x$$

First, by rules (18) through (20) the last symbol in every sequence is marked with index  $x$ . Otherwise, rules (24) through (28) and rule (31) never become applicable and all the unmarked sequences become permanent resulting into an unsuccessful derivation. The last sequence is already marked.

$$\begin{aligned} & AAAABBBBABBBAA_x \\ \Rightarrow & AAAA_xBBBBABBBAA_x \quad [(18)] \\ \Rightarrow & AAAA_xBBBBB_xBBBA_x \quad [(18)] \\ \Rightarrow & AAAA_xBBBBB_xA_xBBBA_x \quad [(20)] \\ \Rightarrow & AAAA_xBBBBB_xA_xBBB_xAA_x \quad [(19)] \end{aligned}$$

Notice, one symbol sequence of  $A$ s is legal. Then, every sequence of  $A$ s is processed in left-to-right direction by rules (21) through (24), but can be successfully rewritten earlier by rules (25) through (28), in the case it consists of less than 5 symbols. Thus, a longer sequence leads to an unsuccessful derivation.

$$\begin{aligned} & AAAA_xBBBBB_xA_xBBB_xAA_x \\ \Rightarrow & \bar{a}1AA_xBBBBB_xA_xBBB_xAA_x \quad [(21)] \\ \Rightarrow & \bar{a}\bar{a}2A_xBBBBB_xA_xBBB_xAA_x \quad [(22)] \\ \Rightarrow & \bar{a}\bar{a}\bar{a}3A_xBBBBB_xA_xBBB_xAA_x \quad [(27)] \\ \Rightarrow & \bar{a}\bar{a}\bar{a}\bar{a}4A_xBBBBB_xA_xBBB_xAA_x \quad [(25)] \\ \Rightarrow & \bar{a}\bar{a}\bar{a}\bar{a}BBBBB_x\bar{a}BBB_x\bar{a}\bar{a} \quad [(26)] \end{aligned}$$

If the processing does not start from the leftmost symbol in the current sequence, it remains permanent. Every sequence of  $B$ s is processed by applying rule (29), zero or multiple times rule (30), and finally rule (31). It ensures the lengths of sequences of  $B$ s are at least 3 symbols.

$$\begin{aligned}
& \overline{aaaaBBBB_x\bar{a}BBB_x\bar{a}a} \\
\Rightarrow & \overline{aaaabB_xBB_x\bar{a}BBB_x\bar{a}a} & [(29)] \\
\Rightarrow & \overline{aaaabbB_xB_x\bar{a}BBB_x\bar{a}a} & [(30)] \\
\Rightarrow & \overline{aaaabbbb\bar{a}BBB_x\bar{a}a} & [(31)] \\
\Rightarrow & \overline{aaaabbbb\bar{a}bB_xB_x\bar{a}a} & [(29)] \\
\Rightarrow & \overline{aaaabbbb\bar{a}bbbaa} & [(31)]
\end{aligned}$$

Notice, it depends on the order of applied rules only within one sequence. Multiple sequences may be processed at random without affecting the derivation.

In phase (e), a resulting terminal string is generated by rules (32) and (33).

$$\overline{aaaabbbb\bar{a}bbbaa} \Rightarrow^* aaaabbbb\bar{a}bbbaa$$

Therefore, if the derivation is terminating, we achieve a string with an equal number of  $a$ s and  $b$ s, where every sequence of  $a$ s is at most 5 symbols long and every sequence of  $b$ s is at least 3 symbols long.

Grammar  $G$  is obviously a monotone phrase-structure grammar in the binary form. Let us now show that for any  $x \in L(G)$ , there is  ${}_G\Delta_x \in {}_G\blacktriangle$ , where any two neighboring paths contain no more than 2 pairs of context-dependent nodes.

Every pair of context-dependent nodes in  ${}_G\Delta_x$  corresponds to one non-context-free rule in  $S \Rightarrow^* x$ . Consider the six phases sketched above. Observe that phases (a), (b), and (e) contain only context-free rules, so we have only to investigate (c) and (d). On the other hand, (c) and (d) contain no rule of the form  $A \rightarrow BC$ , thus the number of neighboring paths remains unchanged.

In (c) by rules (12) through (17) the derivation may proceed in left-to-right direction through the whole sentence form (except the rightmost symbol) introducing a context dependency between every pair of neighboring paths.

In (d), first, the context dependency is introduced between all neighboring paths representing the borders between the sequences of  $A$ s and  $B$ s by rules (18) through (20). Second, every sequence of  $A$ s or  $B$ s is processed in the left-to-right direction by non-context-free rules (21) through (31) introducing a context dependency between all neighboring paths representing symbols inside the sequences of  $A$ s and  $B$ s.

No other non-context-free rule is applied, therefore no other context-dependent pair of nodes can occur. Then, every pair of neighboring paths may contain at most one context-dependent pair of nodes introduced in phase (c) and one introduced in phase (d).

Since  $G$  is a monotone PSG in the binary form, where for every  $x \in L(G)$ , there is  ${}_G\Delta_x \in {}_G\blacktriangle$ , where any two neighboring paths contain no more than 2 pairs of context-dependent nodes, by Theorem 2.3.31,  $L(G) \in \mathbf{CF}$ .

### 2.3.3 How to Disprove Context-Freeness

When examining complicated formal languages, we often need to demonstrate that they are non-context-free and, therefore, beyond the power of context-free grammars. The present section explains how to make a demonstration like this.

#### The Pumping Lemma for Context-Free Languages

The pumping lemma established in this section is frequently used to disprove that a language  $K$  is context-free. The lemma says that for every  $L \in \mathbf{CF}$ , there is a constant  $k \geq 1$  such that every  $z \in L$  with  $|z| \geq k$  can be expressed as  $z = uvwxy$  with  $vx \neq \varepsilon$  so that  $L$  also contains  $uv^mwx^my$ , for every  $m \geq 0$ . Consequently, to demonstrate the non-context-freeness of a language,  $K$ , by contradiction, assume that  $K \in \mathbf{CF}$  and  $k$  is its pumping-lemma constant. Select a string  $z \in K$  with  $|z| \geq k$ , consider all possible decompositions of  $z$  into  $uvwxy$ , and for each of these decompositions, prove that  $uv^mwx^my$  is out of  $K$ , for some  $m \geq 0$ , which contradicts the pumping lemma. Thus,  $K \notin \mathbf{CF}$ . Without any loss of generality, we prove the pumping lemma based on CFGs satisfying Chomsky normal form (see Definition 3.1.19). We also make use of some notions introduced earlier in this chapter, such as the derivation tree  $\Delta(A \Rightarrow^* x)$  corresponding to a derivation  $A \Rightarrow^* x$  in a CFG  $G$ , where  $A \in N$  and  $x \in T^*$ . In addition, we use some related graph-theory notions introduced in Sect. 1.3, such as  $\text{depth}(\Delta(A \Rightarrow^* x))$ , which denotes the depth of  $\Delta(A \Rightarrow^* x)$ .

**Lemma 2.3.32.** *Let  $G = (V, T, P, S)$  be a CFG in Chomsky normal form. For every derivation  $A \Rightarrow^* x$  in  $G$ , where  $A \in N$  and  $x \in T^*$ , its corresponding derivation tree  $\Delta(A \Rightarrow^* x)$  satisfies  $|x| \leq 2^{\text{depth}(\Delta(A \Rightarrow^* x)) - 1}$ .*

*Proof.* (by induction on  $\text{depth}(\Delta(A \Rightarrow^* x)) \geq 1$ ).

*Basis.* Let  $\text{depth}(\Delta(A \Rightarrow^* x)) = 1$ , where  $A \in N$  and  $x \in T^*$ . Because  $G$  is in Chomsky normal form,  $A \Rightarrow^* x$  [ $A \rightarrow x$ ] in  $G$ , where  $x \in T$ , so  $|x| = 1$ . For  $\text{depth}(\Delta(A \Rightarrow^* x)) = 1$ ,  $2^{\text{depth}(\Delta(A \Rightarrow^* x)) - 1} = 2^0$ . As  $2^0 = 1$ ,  $|x| \leq 2^{\text{depth}(\Delta(A \Rightarrow^* x)) - 1}$  in this case, so the basis holds true.

*Induction Hypothesis.* Suppose that this lemma holds for all derivation trees of depth  $n$  or less, for some  $n \geq 0$ .

*Induction Step.* Let  $A \Rightarrow^* x$  in  $G$  with  $\text{depth}(\Delta(A \Rightarrow^* x)) = n + 1$ , where  $A \in N$  and  $x \in T^*$ . Let  $A \Rightarrow^* x$  [ $r\rho$ ] in  $G$ , where  $r \in P$  and  $\rho \in P^*$ . As  $G$  is in Chomsky

normal form,  $r : A \rightarrow BC \in P$ , where  $B, C \in N$ . Let  $B \Rightarrow^* u [\varphi]$ ,  $C \Rightarrow^* v [\theta]$ ,  $\varphi, \theta \in P^*$ ,  $x = uv$ ,  $\rho = \varphi\theta$  so that  $A \Rightarrow^* x$  can be expressed in greater detail as  $A \Rightarrow BC [r] \Rightarrow^* uC [\varphi] \Rightarrow^* uv [\theta]$ . Observe that  $\text{depth}(\Delta(B \Rightarrow^* u [\varphi])) \leq \text{depth}(\Delta(A \Rightarrow^* x)) - 1 = n$ , so  $|u| \leq 2^{\text{depth}(\Delta(B \Rightarrow^* u)) - 1}$  by the induction hypothesis. Analogously, as  $\text{depth}(\Delta(C \Rightarrow^* v [\theta])) \leq \text{depth}(\Delta(A \Rightarrow^* x)) - 1 = n$ ,  $|v| \leq 2^{\text{depth}(\Delta(C \Rightarrow^* v)) - 1}$ . Thus,  $|x| = |u| + |v| \leq 2^{\text{depth}(\Delta(B \Rightarrow^* u)) - 1} + 2^{\text{depth}(\Delta(C \Rightarrow^* v)) - 1} \leq 2^{n-1} + 2^{n-1} = 2^n = 2^{\text{depth}(\Delta(A \Rightarrow^* x)) - 1}$ .  $\square$

**Corollary 2.3.33.** *Let  $G = (V, T, P, S)$  be a CFG in Chomsky normal form. For every derivation  $A \Rightarrow^* x$  in  $G$ , where  $A \in N$  and  $x \in T^*$  with  $|x| \geq 2^m$  for some  $m \geq 0$ , its corresponding derivation tree  $\Delta(A \Rightarrow^* x)$  satisfies  $\text{depth}(\Delta(A \Rightarrow^* x)) \geq m + 1$ .*

*Proof.* This corollary follows from Lemma 2.3.32 and the contrapositive law.  $\square$

**Lemma 2.3.34.** *Pumping Lemma for CF. Let  $L$  be an infinite context-free language. Then, there exists  $k \geq 1$  such that every string  $z \in L$  satisfying  $|z| \geq k$  can be expressed as  $z = uvwxy$ , where  $0 < |vx| < |vwx| \leq k$ , and  $uv^mwx^my \in L$ , for all  $m \geq 0$ .*

*Proof.* Let  $L \in \mathbf{CF}$ , and  $L = L(G)$ , where  $G = (V, T, P, S)$  be a CFG in Chomsky normal form. Let  $G$  have  $n$  nonterminals, for  $n \geq 1$ ; in symbols,  $\text{card}(N) = n$ . Set  $k = 2^n$ . Let  $z \in L(G)$  satisfying  $|z| \geq k$ . As  $z \in L(G)$ ,  $S \Rightarrow^* z$ , and by Corollary 2.3.33,  $\text{depth}(\Delta(S \Rightarrow^* z)) \geq \text{card}(N) + 1$ , so  $\Delta(S \Rightarrow^* z)$  contains some subtrees in which there is a path with two or more nodes labeled by the same nonterminal. Express  $S \Rightarrow^* z$  as  $S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvwxy$  with  $uvwxy = z$  so that the derivation tree corresponding to  $A \Rightarrow^+ vAx \Rightarrow^+ vwx$  contains no proper subtree with a path containing two or more different nodes labeled with the same nonterminal. To prove that  $0 < |vx| < |vwx| \leq k$ , recall that every rule in  $P$  has on its right-hand side either a terminal or two nonterminals because  $G$  is in Chomsky normal form. Thus,  $A \Rightarrow^+ vAx$  implies  $0 < |vx|$ , and  $vAx \Rightarrow^+ vwx$  implies  $|vx| < |vwx|$ . As the derivation tree corresponding to  $A \Rightarrow^+ vAx \Rightarrow^+ vwx$  contains no subtree with a path containing two different nodes labeled with the same nonterminal,  $\text{depth}(\Delta(A \Rightarrow^+ vwx)) \leq \text{card}(N) + 1$ , so by Lemma 2.3.32,  $|vx| < |vwx| \leq 2^n = k$ . Finally, we demonstrate that for all  $m \geq 0$ ,  $uv^mwx^my \in L$ . As  $S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvwxy$ ,  $S \Rightarrow^* uAy \Rightarrow^+ uwy$ , so  $uv^0wx^0y = uwy \in L$ . Similarly, since  $S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvwxy$ ,  $S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvvAxy \Rightarrow^+ \dots \Rightarrow^+ uv^mAx^my \Rightarrow^+ uv^mwx^my$ , so  $uv^mwx^my \in L$ , for all  $m \geq 1$ . Thus, Lemma 2.3.34 holds true.  $\square$

## Applications of the Pumping Lemma

We usually use the pumping lemma in a proof by contradiction to demonstrate that a given language  $L$  is not context-free. Typically, we make a proof of this kind in the following way.



- (1) Assume that  $L$  is context-free.
- (2) Select a string  $z \in L$  whose length depends on the pumping-lemma constant  $k$  so that  $|z| \geq k$  is necessarily true.
- (3) For all possible decompositions of  $z$  into  $uvwxy$  satisfying the pumping-lemma conditions, find  $m \geq 0$  such that  $uv^mwx^m y \notin L$ , which contradicts Lemma 2.3.34.
- (4) The contradiction obtained in (3) means that the assumption in (1) is incorrect; therefore,  $L$  is not context-free.

*Example 2.3.35.* Consider  $L = \{a^n b^n c^n \mid n \geq 1\}$ . Next, under the guidance of the recommended proof structure preceding this example, we demonstrate that  $L \notin \mathbf{CF}$ .

- (1) Assume that  $L \in \mathbf{CF}$ .
- (2) In  $L$ , select  $z = a^k b^k c^k$  with  $|z| = 3k \geq k$ , where  $k$  is the pumping-lemma constant.
- (3) By Lemma 2.3.34,  $z$  can be written as  $z = uvwxy$  so that this decomposition satisfies the pumping-lemma conditions. As  $0 < |vx| < |vwx| \leq k$ , either  $vwx \in \{a\}^* \{b\}^*$  or  $vwx \in \{b\}^* \{c\}^*$ . If  $vwx \in \{a\}^* \{b\}^*$ ,  $uv^0wx^0y$  has  $k$   $c$ s but fewer than  $k$   $a$ s or  $b$ s, so  $uv^0wx^0y \notin L$ , but by the pumping-lemma,  $uv^0wx^0y \in L$ . If  $vwx \in \{b\}^* \{c\}^*$ ,  $uv^0wx^0y$  has  $k$   $a$ s but fewer than  $k$   $b$ s or  $c$ s, so  $uv^0wx^0y \notin L$ , but by the pumping lemma,  $uv^0wx^0y \in L$ . In either case, we obtain the contradiction that  $uv^0wx^0y \notin L$  and, simultaneously,  $uv^0wx^0y \in L$ .
- (4) By the contradiction obtained in (3),  $L \notin \mathbf{CF}$ . □

Omitting some obvious details, we usually proceed in a briefer way than above when proving the non-context-freeness of a language by using Lemma 2.3.34.

*Example 2.3.36.* Let  $L = \{a^n b^m a^n b^m \mid n, m \geq 1\}$ . Assume that  $L$  is context-free. Set  $z = a^k b^k a^k b^k$  with  $|a^k b^k a^k b^k| = 4k \geq k$ . By Lemma 2.3.34, express  $z = uvwxy$ . Observe that  $0 < |vx| < |vwx| \leq k$  implies  $uwy \notin L$  in all possible occurrences of  $vwx$  in  $a^k b^k a^k b^k$ ; however, by Lemma 2.3.34,  $uwy \in L$ —a contradiction. Thus,  $L \notin \mathbf{CF}$ . □

Even some seemingly trivial unary languages are not context-free as shown next.

*Example 2.3.37.* Consider  $L = \{a^{n^2} \mid \text{for some } n \geq 0\}$ . To demonstrate  $L \notin \mathbf{CF}$ , assume that  $L \in \mathbf{CF}$  and select  $z = a^{k^2} \in L$  where  $k$  is the pumping-lemma constant. As a result,  $|z| = k^2 \geq k$ , so  $z = uvwxy$ , which satisfies the pumping-lemma conditions. As  $k^2 < |uv^2wx^2y| \leq k^2 + k < k^2 + 2k + 1 = (k+1)^2$ , we have  $uv^2wx^2y \notin L$ , but by Lemma 2.3.34,  $uv^2wx^2y \in L$ —a contradiction. Thus,  $L \notin \mathbf{CF}$ . □

### 2.3.4 Parallel Grammars

Parallel grammars represent modified versions of context-free grammars that rewrite their strings in parallel. Most often, this book represents a grammatical parallelism like this by *Extended tabled zero-sided Lindenmayer grammars* or,

more briefly, ETOL grammars and their special cases, such as EOL grammars (see [PHHM96b, PL90a, RS80, RS86]). Originally, these grammars were introduced in connection with a theory proposed for the development of filamentous organisms. Developmental stages of cellular arrays are described by strings with each symbol being a cell. Rules correspond to developmental instructions with which organisms can be produced. They are applied simultaneously to all cell-representing symbols because in a growing organism development proceeds simultaneously everywhere. That is, all symbols, including terminals, are always rewritten to adequately reflect the development of real organisms that contain no dead cells which would remain permanently fixed in their place in the organism; disappearing cells are represented by  $\varepsilon$ . Instead of a single set of rules, ETOL grammars have a finite set of sets containing rules. Each of them contains rules that describe developmental instructions corresponding to a specific biological circumstances, such as environmental conditions concerning temperature or coexistence of other organisms. Naturally, during a single derivation step, rules from only one of these sets can be applied to the rewritten string.

Considering these biologically motivated features of ETOL grammars, we see the following three main conceptual differences between them and the previously discussed sequential grammars, such as context-free grammars (see Sect. 2.3).

- (I) Instead of a single set of rules, they have finitely many sets of rules.
- (II) The left-hand side of a rule may be formed by any grammatical symbol, including a terminal.
- (III) All symbols of a string are simultaneously rewritten during a single derivation step.

**Definition 2.3.38.** An *ETOL grammar* is a  $(t + 3)$ -tuple

$$G = (V, T, P_1, \dots, P_t, w)$$

where  $t \geq 1$ , and  $V, T$ , and  $w$  are the *total alphabet*, the *terminal alphabet* ( $T \subseteq V$ ), and the *start string* ( $w \in V^+$ ), respectively. Each  $P_i$  is a finite set of *rules* of the form

$$a \rightarrow x$$

where  $a \in V$  and  $x \in V^*$ . If  $a \rightarrow x \in P_i$  implies that  $x \neq \varepsilon$  for all  $i = 1, \dots, t$ , then  $G$  is said to be *propagating* (an *EPTOL grammar* for short).

Let  $u, v \in V^*$ ,  $u = a_1 a_2 \dots a_q$ ,  $v = v_1 v_2 \dots v_q$ ,  $q = |u|$ ,  $a_j \in V$ ,  $v_j \in V^*$ , and  $p_1, p_2, \dots, p_q$  is a sequence of rules of the form  $p_j = a_j \rightarrow v_j \in P_i$  for all  $j = 1, \dots, q$ , for some  $i \in \{1, \dots, t\}$ . Then,  $u$  *directly derives*  $v$  according to the rules  $p_1$  through  $p_q$ , denoted by

$$u \Rightarrow_G v [p_1, p_2, \dots, p_q]$$

If  $p_1$  through  $p_q$  are immaterial, we write just  $u \Rightarrow_G v$ . In the standard manner, we define the relations  $\Rightarrow_G^n$  ( $n \geq 0$ ),  $\Rightarrow_G^*$ , and  $\Rightarrow_G^+$ .

The *language* of  $G$ , denoted by  $L(G)$ , is defined as

$$L(G) = \{y \in T^* \mid w \Rightarrow_G^* y\} \quad \square$$

The families of languages generated by ETOL and EPTOL grammars are denoted by **ETOL** and **EPTOL**, respectively.

**Definition 2.3.39.** Let  $G = (V, T, P_1, \dots, P_t, w)$  be an ETOL grammar, for some  $t \geq 1$ . If  $t = 1$ , then  $G$  is called an *EOL grammar*.  $\square$

The families of languages generated by EOL and propagating EOL grammars (*EPOL grammars* for short) are denoted by **EOL** and **EPOL**, respectively.

**Definition 2.3.40.** An *OL grammar* is defined by analogy with an EOL grammar except that  $V = T$ .  $\square$

For simplicity, we specify an OL grammar as a triple  $G = (T, P, S)$  rather than a quadruple  $G = (T, T, P, S)$ . By **OL**, we denote the family of languages generated by OL grammars.

**Theorem 2.3.41** (See [RS80]).

$$\mathbf{CF} \subset \mathbf{EOL} = \mathbf{EPOL} \subset \mathbf{ETOL} = \mathbf{EPTOL} \subset \mathbf{CS}$$

ETOL grammars work in a totally parallel way because they simultaneously rewrite all symbols of the current sentential form during every single derivation step. Apart from them, however, there also exist semi-parallel grammars, which simultaneously change only selected symbols in the rewritten strings while keeping their rest unchanged. We close this section by defining one of them, namely, queue grammars. However, let us point out that we study several other semi-parallel grammatical mechanisms later in this book (see Sect. 4.1 and Chap. 12).

## Queue Grammars

Queue grammars (see [KR83]) always rewrite the first and the last symbol in the strings in parallel; all the other symbols in between them remain unchanged. That is, as their name indicates, queue grammars rewrite strings in a way that resemble the standard way of working with an abstract data type referred to as a queue. Indeed, these grammars work on strings based upon the well-known first-in-first-out principle—that is, the first symbol added to the string will be the first one to be removed. More specifically, during every derivation step, these grammars attach a string as a suffix to the current sentential form while eliminating the leftmost symbol of this form; as a result, all symbols that were attached prior to this step have to be removed before the newly attached suffix is removed. Next, we define these grammars rigorously.

**Definition 2.3.42.** A *queue grammar* is a sextuple

$$Q = (V, T, W, F, R, g)$$

where  $V$  and  $W$  are two alphabets satisfying  $V \cap W = \emptyset$ ,  $T \subset V$ ,  $F \subset W$ ,  $g \in (V - T)(W - F)$ , and

$$R \subseteq V \times (W - F) \times V^* \times W$$

is a finite relation such that for each  $a \in V$ , there exists an element  $(a, b, x, c) \in R$ . If  $u = arb$ ,  $v = rxc$ , and  $(a, b, x, c) \in R$ ,  $r, x \in V^*$ , where  $a \in V$  and  $b, c \in W$ , then  $Q$  makes a *derivation step* from  $u$  to  $v$  according to  $(a, b, x, c)$ , symbolically written as

$$u \Rightarrow_Q v [(a, b, x, c)]$$

or, simply,  $u \Rightarrow_Q v$ . We define  $\Rightarrow_Q^n$  ( $n \geq 0$ ),  $\Rightarrow_Q^+$ , and  $\Rightarrow_Q^*$  in the standard way. The *language* of  $Q$ , denoted by  $L(Q)$ , is defined as

$$L(Q) = \{x \in T^* \mid g \Rightarrow_Q^* xf, f \in F\} \quad \square$$

As an example, consider a queue grammar  $G = (V, T, W, F, s, P)$ , where  $V = \{S, A, a, b\}$ ,  $T = \{a, b\}$ ,  $W = \{Q, f\}$ ,  $F = \{f\}$ ,  $s = SQ$  and  $P = \{p_1, p_2\}$ ,  $p_1 = (S, Q, Aaa, Q)$  and  $p_2 = (A, Q, bb, f)$ . Then, there exists a derivation

$$s = SQ \Rightarrow AaaQ[p_1] \Rightarrow aabbbf[p_2]$$

in this queue grammar, which generates  $aabb$ .

**Theorem 2.3.43 (See [KR83]).** For every recursively enumerable language  $K$ , there is a queue grammar  $Q$  such that  $L(Q) = K$ .

Next, we slightly modify the definition of a queue grammar.

**Definition 2.3.44 (See [Med04, KM00, Med03c]).** A *left-extended queue grammar* is a sextuple

$$Q = (V, T, W, F, R, g)$$

where  $V, T, W, F, R, g$  have the same meaning as in a queue grammar; in addition, assume that  $\# \notin V \cup W$ . If  $u, v \in V^*\{\#\}V^*W$  so  $u = w\#arb$ ,  $v = wa\#rzc$ ,  $a \in V$ ,  $r, z, w \in V^*$ ,  $b, c \in W$ , and  $(a, b, z, c) \in R$ , then

$$u \Rightarrow_Q v [(a, b, z, c)]$$

or, simply,  $u \Rightarrow_Q v$ . In the standard manner, extend  $\Rightarrow_Q$  to  $\Rightarrow_Q^n$ , where  $n \geq 0$ . Based on  $\Rightarrow_Q^n$ , define  $\Rightarrow_Q^+$  and  $\Rightarrow_Q^*$ . The *language* of  $Q$ , denoted by  $L(Q)$ , is defined as

$$L(Q) = \{v \in T^* \mid \#g \Rightarrow_Q^* w\#vf \text{ for some } w \in V^* \text{ and } f \in F\} \quad \square$$

Less formally, during every step of a derivation, a left-extended queue grammar shifts the rewritten symbol over  $\#$ ; in this way, it records the derivation history, which represents a property fulfilling a crucial role in several proofs later in this book.

For example, consider a left-extended queue grammar, which has the same components as the previously mentioned queue grammar  $G$ . Then, there exists a derivation

$$\#s = \#SQ \Rightarrow S\#AaaQ[p_1] \Rightarrow SA\#aabbf[p_2]$$

in this left-extended queue grammar, which generates  $aabb$ . Moreover, this type of queue grammar saves symbols from the first components of rules which were used in the derivation.

**Theorem 2.3.45.** *For every queue grammar  $Q$ , there is an equivalent left-extended queue grammar  $Q'$  such that  $L(Q') = L(Q)$ .*

The proof is trivial and left to the reader.

**Theorem 2.3.46.** *For every recursively enumerable language  $K$ , there is a left-extended queue grammar  $Q$  such that  $L(Q) = K$ .*

*Proof.* Follows from Theorems 2.3.43 and 2.3.45.

## 2.4 Automata

In this section, based upon rewriting systems, we define automata as fundamental formal devices that accept strings of a given language (see [Med00a]). Specifically, we give the definitions of finite and pushdown automata, which accept the families of regular and context-free languages, respectively. As they are special cases of rewriting systems, introduced in Sect. 2.2, we straightforwardly apply the mathematical terminology concerning rewriting systems to finite automata. Perhaps most significantly, we apply relations  $\vdash$ ,  $\vdash''$ ,  $\vdash^+$ , and  $\vdash^*$  to them.

A *finite automaton* can be viewed as a rewriting system  $M = (\Gamma, R)$ , where

- the total alphabet  $\Delta$  contains subsets  $Q$ ,  $F$ , and  $\Sigma$  such that  $\Delta = Q \cup \Sigma$ ,  $F \subseteq Q$ , and  $Q \cap \Sigma = \emptyset$ ;
- $R$  is a finite *set of rules* of the form  $qa \rightarrow p$ , where  $q, p \in Q$  and  $a \in \Sigma \cup \{\varepsilon\}$ .

More commonly, however, the notion of a finite automaton is specified as follows.

**Definition 2.4.1.** A *general finite automaton* is a quintuple

$$M = (Q, \Sigma, R, s, F)$$

where

- $Q$  is a finite set of *states*;
- $\Sigma$  is an *input alphabet*;
- $R \subseteq Q \times \Sigma^* \times Q$  is a finite relation, called the set of *rules* (or *transitions*);
- $s \in Q$  is the *start state*;
- $F \subseteq Q$  is the set of *final states*.

Instead of  $(p, y, q) \in R$ , we write  $py \rightarrow q \in R$ . If  $py \rightarrow q \in R$  implies that  $y \neq \varepsilon$ , then  $M$  is said to be  *$\varepsilon$ -free*.

A *configuration* of  $M$  is any string from  $Q\Sigma^*$ . The relation of a *move*, symbolically denoted by  $\vdash_M$ , is defined over  $Q\Sigma^*$  as follows:

$$pyx \vdash_M qx$$

if and only if  $pyx, qx \in Q\Sigma^*$  and  $py \rightarrow q \in R$ .

Let  $\vdash_M^n$ ,  $\vdash_M^*$ , and  $\vdash_M^+$  denote the  $n$ th power of  $\vdash_M$ , for some  $n \geq 0$ , the reflexive-transitive closure of  $\vdash_M$ , and the transitive closure of  $\vdash_M$ , respectively. The *language* of  $M$  is denoted by  $L(M)$  and defined as

$$L(M) = \{w \in \Sigma^* \mid sw \vdash_M^* f, f \in F\} \quad \square$$

Next, we define three special variants of general finite automata.

**Definition 2.4.2.** Let  $M = (Q, \Sigma, R, s, F)$  be a general finite automaton.  $M$  is a *finite automaton* if and only if  $py \rightarrow q \in R$  implies that  $|y| \leq 1$ .  $M$  is said to be *deterministic* if and only if  $py \rightarrow q \in R$  implies that  $|y| = 1$  and  $py \rightarrow q_1, py \rightarrow q_2 \in R$  implies that  $q_1 = q_2$ , for all  $p, q, q_1, q_2 \in Q$  and  $y \in \Sigma^*$ .  $M$  is said to be *complete* if and only if  $M$  is deterministic and for all  $p \in Q$  and all  $a \in \Sigma$ ,  $pa \rightarrow q \in R$  for some  $q \in Q$ .  $\square$

To make several definitions and proofs concerning finite automata more concise, we sometimes denote a rule  $pa \rightarrow q$  with a unique label  $r$  as  $r: pa \rightarrow q$ . This notion of rule labels is formalized in the following definition.

**Definition 2.4.3.** Let  $M = (Q, \Sigma, R, s, F)$  be a finite automaton. Let  $\Psi$  be an alphabet of *rule labels* such that  $\text{card}(\Psi) = \text{card}(R)$ , and  $\psi$  be a bijection from  $R$  to  $\Psi$ . For simplicity, to express that  $\psi$  maps a rule,  $pa \rightarrow q \in R$ , to  $r$ , where  $r \in \Psi$ , we write  $r: pa \rightarrow q \in R$ ; in other words,  $r: pa \rightarrow q$  means  $\psi(pa \rightarrow q) = r$ .

For every  $y \in \Sigma^*$  and  $r: pa \rightarrow q \in R$ ,  $M$  makes a *move* from configuration  $pay$  to configuration  $qy$  according to  $r$ , written as

$$pay \vdash_M qy [r]$$

Let  $\chi$  be any configuration of  $M$ .  $M$  makes *zero moves* from  $\chi$  to  $\chi$  according to  $\varepsilon$ , symbolically written as

$$\chi \vdash_M^0 \chi [\varepsilon]$$

Let there exist a sequence of configurations  $\chi_0, \chi_1, \dots, \chi_n$  for some  $n \geq 1$  such that  $\chi_{i-1} \vdash_M \chi_i [r_i]$ , where  $r_i \in \Psi$ , for  $i = 1, \dots, n$ , then  $M$  makes *n moves* from  $\chi_0$  to  $\chi_n$  according to  $r_1 \cdots r_n$ , symbolically written as

$$\chi_0 \vdash_M^n \chi_n [r_1 \cdots r_n]$$

Define  $\vdash_M^*$  and  $\vdash_M^+$  in the standard manner. □

Sometimes, we specify a finite automaton as  $M = (Q, \Sigma, \Psi, R, s, F)$ , where  $Q$ ,  $\Sigma$ ,  $\Psi$ ,  $R$ ,  $s$ , and  $F$  are the set of states, the input alphabet, the alphabet of rule labels, the set of rules, the start state, and the set of final states, respectively.

**Theorem 2.4.4** (See [Woo87]). *For every general finite automaton  $M$ , there is a complete finite automaton  $M'$  such that  $L(M') = L(M)$ .*

Finite automata accept precisely the family of regular languages:

**Theorem 2.4.5** (See [Woo87]). *A language  $K$  is regular if and only if there is a complete finite automaton  $M$  such that  $K = L(M)$ .*

Pushdown automata represent finite automata extended by a potentially unbounded pushdown store. We first define their general version, customarily referred to as extended pushdown automata.

**Definition 2.4.6.** An *extended pushdown automaton* is a septuple

$$M = (Q, \Sigma, \Gamma, R, s, S, F)$$

where

- $Q$ ,  $\Sigma$ ,  $s$ , and  $F$  are defined as in a finite automaton;
- $\Gamma$  is a *pushdown alphabet*;
- $R \subseteq \Gamma^* \times Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$  is a finite relation, called the set of *rules* (or *transitions*);
- $S$  is the *initial pushdown symbol*.

$Q$  and  $(\Sigma \cup \Gamma)$  are always assumed to be disjoint. By analogy with finite automata, instead of  $(\gamma, p, a, w, q) \in R$ , we write  $\gamma pa \rightarrow wq$ .

A *configuration* of  $M$  is any string from  $\Gamma^* Q \Sigma^*$ . The relation of a *move*, symbolically denoted by  $\vdash_M$ , is defined over  $\Gamma^* Q \Sigma^*$  as follows:

$$x\gamma pay \vdash_M xwqy$$

if and only if  $x\gamma pay, xwqy \in \Gamma^* Q \Sigma^*$  and  $\gamma pa \rightarrow wq \in R$ .

Let  $\vdash_M^k$ ,  $\vdash_M^*$ , and  $\vdash_M^+$  denote the  $k$ th power of  $\vdash_M$ , for some  $k \geq 0$ , the reflexive-transitive closure of  $\vdash_M$ , and the transitive closure of  $\vdash_M$ , respectively.  $\square$

For an extended pushdown automaton, there exist three ways of language acceptance: (1) by entering a final state, (2) by emptying its pushdown, and (3) by entering a final state and emptying its pushdown. All of them are defined next.

**Definition 2.4.7.** Let  $M = (Q, \Sigma, \Gamma, R, s, S, F)$  be an extended pushdown automaton. The *language accepted by  $M$  by final state* is denoted by  $L_f(M)$  and defined as

$$L_f(M) = \{w \in \Sigma^* \mid Ssw \vdash_M^* \gamma f, f \in F, \gamma \in \Gamma^*\}$$

The *language accepted by  $M$  by empty pushdown* is denoted by  $L_e(M)$  and defined as

$$L_e(M) = \{w \in \Sigma^* \mid Ssw \vdash_M^* q, q \in Q\}$$

The *language accepted by  $M$  by empty pushdown and final state*, denoted by  $L_{ef}(M)$ , is defined as

$$L_{ef}(M) = \{w \in \Sigma^* \mid Ssw \vdash_M^* f, f \in F\} \quad \square$$

Let  $\mathbf{EPDA}_f$ ,  $\mathbf{EPDA}_e$ , and  $\mathbf{EPDA}_{ef}$  denote the language families accepted by extended pushdown automata accepting by final state, by empty pushdown, and by final state and empty pushdown, respectively.

All of the three ways of acceptance are equivalent:

**Theorem 2.4.8** (See [Med00a]).  $\mathbf{EPDA}_f = \mathbf{EPDA}_e = \mathbf{EPDA}_{ef}$

If an extended pushdown automaton rewrites a single symbol on its pushdown top during every move, we obtain a pushdown automaton, defined next.

**Definition 2.4.9.** Let  $M = (Q, \Sigma, \Gamma, R, s, S, F)$  be an extended pushdown automaton. Then,  $M$  is a *pushdown automaton* if and only if  $\gamma pa \rightarrow wq \in R$  implies that  $|\gamma| = 1$ .  $\square$

To make definitions and proofs concerning pushdown automata more readable, we sometimes denote a rule  $Apa \rightarrow wq$  with a unique label  $r$  as  $r: Apa \rightarrow wq$ , which is formalized in the following definition.

**Definition 2.4.10.** Let  $M = (Q, \Sigma, \Gamma, R, s, S, F)$  be a pushdown automaton. Let  $\Psi$  be an alphabet of *rule labels* such that  $\text{card}(\Psi) = \text{card}(R)$ , and  $\psi$  be a bijection from  $R$  to  $\Psi$ . For simplicity, to express that  $\psi$  maps a rule,  $Apa \rightarrow wq \in R$ , to  $r$ , where  $r \in \Psi$ , we write  $r: Apa \rightarrow wq \in R$ ; in other words,  $r: Apa \rightarrow wq$  means  $\psi(Apa \rightarrow wq) = r$ .



For every  $x \in \Gamma^*$ ,  $y \in \Sigma^*$ , and  $r: Apa \rightarrow wq \in R$ ,  $M$  makes a *move* from configuration  $xApay$  to configuration  $xwqy$  according to  $r$ , written as

$$xApay \vdash_M xwqy [r]$$

Let  $\chi$  be any configuration of  $M$ .  $M$  makes *zero moves* from  $\chi$  to  $\chi$  according to  $\varepsilon$ , symbolically written as

$$\chi \vdash_M^0 \chi [\varepsilon]$$

Let there exist a sequence of configurations  $\chi_0, \chi_1, \dots, \chi_n$  for some  $n \geq 1$  such that  $\chi_{i-1} \vdash_M \chi_i [r_i]$ , where  $r_i \in \Psi$ , for  $i = 1, \dots, n$ , then  $M$  makes *n moves* from  $\chi_0$  to  $\chi_n$  according to  $r_1 \cdots r_n$ , symbolically written as

$$\chi_0 \vdash_M^n \chi_n [r_1 \cdots r_n]$$

Define  $\vdash_M^*$  and  $\vdash_M^+$  in the standard manner. □

Let  $\mathbf{PDA}_f$ ,  $\mathbf{PDA}_e$ , and  $\mathbf{PDA}_{ef}$  denote the language families accepted by pushdown automata accepting by final state, by empty pushdown, and by final state and empty pushdown, respectively.

**Theorem 2.4.11** (See [Med00a]).  $\mathbf{PDA}_f = \mathbf{PDA}_e = \mathbf{PDA}_{ef}$

As the next theorem states, pushdown automata characterize the family of context-free languages.

**Theorem 2.4.12** (See [Med00a]).

$$\mathbf{CF} = \mathbf{EPDA}_f = \mathbf{EPDA}_e = \mathbf{EPDA}_{ef} = \mathbf{PDA}_f = \mathbf{PDA}_e = \mathbf{PDA}_{ef}$$

Finally, we define deterministic versions of PDAs.

**Definition 2.4.13.** Let  $M = (Q, \Sigma, \Gamma, R, s, S, F)$  be a pushdown automaton.  $M$  is a *deterministic pushdown automaton* if

- (1) for any  $q \in Q$ ,  $a \in \Sigma \cup \{\varepsilon\}$ ,  $x \in \Gamma$ , the set  $\{(q, a, x) \mid (q, a, x) \in R\}$ , has at most one element and
- (2) if  $\{(q, \varepsilon, x) \mid (q, \varepsilon, x) \in R\} \neq \emptyset$ , then  $\{(q, a, x) \mid (q, a, x) \in R, a \in \Sigma\} = \emptyset$  for every  $q$  and  $a$ . □

Let  $\mathbf{DPDA}$  denote the family of languages accepted by deterministic pushdown automata. Unlike finite automata, determinism in the case of pushdown automata decrease the acceptance power.

**Theorem 2.4.14** (See [Med00a]).

$$\mathbf{REG} \subset \mathbf{DPDA} \subset \mathbf{CF}$$



<http://www.springer.com/978-3-319-63099-1>

Modern Language Models and Computation

Theory with Applications

Meduna, A.; Soukup, O.

2017, XIX, 548 p. 20 illus., Hardcover

ISBN: 978-3-319-63099-1