

1.7 ALPHABETS AND LANGUAGES

The algorithms we studied informally in the last section have much that is left vague. For example, we have not specified exactly how the relations R and R^* that need to be accessed and modified are represented and stored. In computational practice, such *data* are encoded in the computer's memory as strings of bits or other symbols appropriate for manipulation by a computer. The mathematical study of the theory of computation must therefore begin by understanding the *mathematics of strings of symbols*.

We start with the notion of an **alphabet**: a finite set of symbols. An example is, naturally, the Roman alphabet $\{a, b, \dots, z\}$. An alphabet particularly pertinent to the theory of computation is the **binary alphabet** $\{0, 1\}$. In fact, any object can be in an alphabet; from a formal point of view, an alphabet is simply a finite set of any sort. For simplicity, however, we use as symbols only letters, numerals, and other common characters such as \$, or #.

A string over an alphabet is a finite sequence of symbols from the alphabet. Instead of writing strings with parentheses and commas, as we have written other sequences, we simply juxtapose the symbols. Thus *watermelon* is a string over the alphabet $\{a, b, \dots, z\}$, and 0111011 is a string over $\{0, 1\}$. Also, using the natural isomorphism, we identify a string of only one symbol with the symbol itself; thus the symbol a is the same as the string a . A string may have no symbols at all; in this case it is called the **empty string** and is denoted by ϵ . We generally use u, v, x, y, z , and Greek letters to denote strings; for example, we might use w as a name for the string abc . Of course, to avoid confusion it is a good practice to refrain from using as symbols letters we also use as names of strings. The set of all strings, including the empty string, over an alphabet Σ is denoted by Σ^* .

The length of a string is its length as a sequence; thus the length of the string $acrd$ is 4. We denote the length of a string w by $|w|$; thus $|101| = 3$ and $|e| = 0$. Alternatively (that is, via a natural isomorphism) a string $w \in \Sigma^*$ can be considered as a function $w : \{1, \dots, |w|\} \rightarrow \Sigma$; the value of $w(j)$, where $1 \leq j \leq |w|$, is the symbol in the j th position of w . For example, if $w = \text{accord\acute{e}on}$, then $w(3) = w(2) = c$, and $w(1) = a$. This alternative viewpoint brings out a possible point of confusion. Naturally, the symbol c in the third position is identical to that in the second. If, however, we need to distinguish identical symbols at different positions in a string, we shall refer to them as **different occurrences** of the symbol. That is, the symbol $\sigma \in \Sigma$ occurs in the j th position of the string $w \in \Sigma^*$ if $w(j) = \sigma$.

Two strings over the same alphabet can be combined to form a third by the operation of **concatenation**. The concatenation of strings x and y , written $x \circ y$ or simply xy , is the string x followed by the string y ; formally, $w = x \circ y$ if

1.7: Alphabets and Languages

and only if $|w| = |x| + |y|$, $w(j) = x(j)$ for $j = 1, \dots, |x|$, and $w(|x| + j) = y(j)$ for $j = 1, \dots, |y|$. For example, $01 \circ 001 = 01001$, and $\text{beach} \circ \text{boy} = \text{beachboy}$. Of course, $w \circ e = e \circ w = w$ for any string w . And concatenation is associative: $(wx)y = w(xy)$ for any strings w, x , and y . A string v is a **substring** of a string w if and only if there are strings x and y such that $w = xvy$. Both x and y could be ϵ , so every string is a substring of itself; and taking $x = w$ and $v = y = \epsilon$, we see that ϵ is a substring of every string. If $w = xv$ for some x , then v is a **suffix** of w ; if $w = vy$ for some y , then v is a **prefix** of w . Thus *road* is a prefix of *roadrunner*, a suffix of *abroad*, and a substring of both these and of *broader*. A string may have several occurrences of the same substring; for example, *ababab* has three occurrences of *ab* and two of *abab*.

For each string w and each natural number i , the string w^i is defined as

$$w^0 = \epsilon, \quad \text{the empty string}$$

$$w^{i+1} = w^i \circ w \quad \text{for each } i \geq 0$$

Thus $w^1 = w$, and $do^2 = dodo$.

This definition is our first instance of a very common type: **definition by induction**. We have already seen *proofs* by induction, and the underlying idea is much the same. There is a basis case of the definition, here the definition of w^i for $i = 0$; then when that which is being defined has been specified for all $j \leq i$, it is defined for $j = i + 1$. In the example above, w^{i+1} is defined in terms of w^i . To see exactly how any case of the definition can be traced back to the basis case, consider the example of do^2 . According to the definition (with $i = 1$), $(do)^2 = (do)^1 \circ do$. Again according to the definition (with $i = 0$) $(do)^1 = (do)^0 \circ do$. Now the basis case applies: $(do)^0 = \epsilon$. So $(do)^2 = (\epsilon \circ do) \circ do = dodo$.

The reversal of a string w , denoted by w^R , is the string "spelled backwards": for example, $\text{reverse}^R = \text{esrever}$. A formal definition can be given by induction on the length of a string:

- (1) If w is a string of length 0, then $w^R = w = \epsilon$.
- (2) If w is a string of length $n + 1 > 0$, then $w = ua$ for some $a \in \Sigma$, and $w^R = aw^R$.

Let us use this definition to illustrate how a proof by induction can depend on a definition by induction. We shall show that for any strings w and x , $(wx)^R = x^R w^R$. For example, $(\text{dogcat})^R = (\text{cat})^R (\text{dog})^R = \text{tacgod}$. We proceed by induction on the length of x .

Basis Step. $|x| = 0$. Then $x = \epsilon$, and $(wx)^R = (w\epsilon)^R = w^R = \epsilon w^R = \epsilon^R w^R = x^R w^R$.

Induction Hypothesis. If $|x| \leq n$, then $(wx)^R = x^R w^R$.

If Σ is the Roman alphabet and the ordering of $\Sigma = \{a_1, \dots, a_{26}\}$ is the usual one $\{a, \dots, z\}$, then the lexicographic order for strings of equal length is the order used in dictionaries; however, the ordering described by (1) and (2) for all strings in Σ^* differs from the dictionary ordering by listing shorter strings before longer ones.

Since languages are sets, they can be combined by the set operations of union, intersection, and difference. When a particular alphabet Σ is understood from context, we shall write \bar{A} —the complement of A —instead of the difference $\Sigma^* - A$.

In addition, certain operations are meaningful only for languages. The first of these is the concatenation of languages. If L_1 and L_2 are languages over Σ , their concatenation is $L = L_1 \circ L_2$, or simply $L = L_1 L_2$, where

$$L = \{w \in \Sigma^* : w = x \circ y \text{ for some } x \in L_1 \text{ and } y \in L_2\}.$$

For example, if $\Sigma = \{0, 1\}$, $L_1 = \{w \in \Sigma^* : w \text{ has an even number of 0's}\}$ and $L_2 = \{w : w \text{ starts with a 0 and the rest of the symbols are 1's}\}$, then $L_1 \circ L_2 = \{w : w \text{ has an odd number of 0's}\}$.

Another language operation is the Kleene star of a language L , denoted by L^* . L^* is the set of all strings obtained by concatenating zero or more strings from L . (The concatenation of zero strings is e , and the concatenation of one string is the string itself.) Thus,

$$L^* = \{w \in \Sigma^* : w = w_1 \circ \dots \circ w_k \text{ for some } k \geq 0 \text{ and some } w_1, \dots, w_k \in L\}.$$

For example, if $L = \{01, 1, 100\}$, then $110001110011 \in L^*$, since $110001110011 = 1 \circ 100 \circ 01 \circ 1 \circ 100 \circ 1 \circ 1$, and each of these strings is in L .

Note that the use of Σ^* to denote the set of all strings over Σ is consistent with the notation for the Kleene star of Σ , regarded as a finite language. That is, if we let $L = \Sigma$ and apply the definition above, then Σ^* is the set of all strings that can be written as $w_1 \circ \dots \circ w_k$ for some $k \geq 0$ and some $w_1, \dots, w_k \in \Sigma$. Since the w_i are then simply individual symbols in Σ , it follows that Σ^* is, as originally defined, the set of all finite strings whose symbols are in Σ .

For another extreme example, observe that $\emptyset^* = \{e\}$. For let $L = \emptyset$ in the above definition. The only possible concatenation $w_1 \circ w_2 \circ \dots \circ w_k$ with $k \geq 0$ and $w_1, \dots, w_k \in L$ is that with $k = 0$, that is, the concatenation of zero strings; so the sole member of L^* in this case is e !

As a final example, let us show that if L is the language $\{w \in \{0, 1\}^* : w \text{ has an unequal number of 0's and 1's}\}$, then $L^* = \{0, 1\}^*$. To see this, first note that for any languages L_1 and L_2 , if $L_1 \subseteq L_2$, then $L_1^* \subseteq L_2^*$ as is evident from the definition of Kleene star. Second, $\{0, 1\} \subseteq L$, since each of 0 and 1, regarded as a string, has an unequal number of 0's and 1's. Hence $\{0, 1\}^* \subseteq L^*$; but $L^* \subseteq \{0, 1\}^*$ by definition, so $L^* = \{0, 1\}^*$.

Induction Step. Let $|x| = n + 1$. Then $x = ua$ for some $u \in \Sigma^*$ and $a \in \Sigma$ such that $|u| = n$.

$$\begin{aligned} (wx)^R &= (w(ua))^R && \text{since } x = ua \\ &= ((wu)a)^R && \text{since concatenation is associative} \\ &= a(wu)^R && \text{by the definition of the reversal of } (wu)a \\ &= aw^R u^R && \text{by the induction hypothesis} \\ &= (ua)^R w^R && \text{by the definition of the reversal of } ua \\ &= x^R w^R && \text{since } x = ua \end{aligned}$$

Now we move from the study of individual strings to the study of finite and infinite sets of strings. The simple models of computing machines we shall soon be studying will be characterized in terms of regularities in the way they handle many different strings, so it is important first to understand general ways of describing and combining classes of strings.

Any set of strings over an alphabet Σ —that is, any subset of Σ^* —will be called a language. Thus Σ^* , \emptyset , and Σ are languages. Since a language is simply a special kind of set, we can specify a finite language by listing all its strings. For example, $\{aba, czz, d, f\}$ is a language over $\{a, b, \dots, z\}$. However, most languages of interest are infinite, so that listing all the strings is not possible. Languages that might be considered are $\{0, 01, 011, 0111, \dots\}$, $\{w \in \{0, 1\}^* : w \text{ has an equal number of 0's and 1's}\}$, and $\{w \in \Sigma^* : w = w^R\}$. Thus we shall specify infinite languages by the scheme

$$L = \{w \in \Sigma^* : w \text{ has property } P\},$$

following the general form we have used for specifying infinite sets.

If Σ is a finite alphabet, then Σ^* is certainly infinite; but is it a countably infinite set? It is not hard to see that this is indeed the case. To construct a bijection $f : \mathbb{N} \rightarrow \Sigma^*$, first fix some ordering of the alphabet, say $\Sigma = \{a_1, \dots, a_n\}$, where a_1, \dots, a_n are distinct. The members of Σ^* can then be enumerated in the following way.

- (1) For each $k \geq 0$, all strings of length k are enumerated before all strings of length $k + 1$.
- (2) The n^k strings of length exactly k are enumerated lexicographically, that is, $a_{i_1} \dots a_{i_k}$ precedes $a_{j_1} \dots a_{j_k}$, provided that, for some m , $0 \leq m \leq k - 1$, $i_\ell = j_\ell$ for $\ell = 1, \dots, m$, and $i_{m+1} < j_{m+1}$.

For example, if $\Sigma = \{0, 1\}$, the order would be as follows:

$$e, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots$$

We write L^+ for the language LL^* . Equivalently, L^+ is the language

$$\{w \in \Sigma^* : w = w_1 \circ w_2 \circ \dots \circ w_k \text{ for some } k \geq 1 \text{ and some } w_1, \dots, w_k \in L\}.$$

Notice that L^+ can be considered as the *closure* of L under the function of concatenation. That is, L^+ is the smallest language that includes L and all strings that are concatenations of strings in L .

Problems for Section 1.7

1.7.1. (a) Prove, using the definition of concatenation given in the text, that concatenation of strings is associative.

(b) Give an inductive definition of the concatenation of strings.

(c) Using the inductive definition from (b), prove that the concatenation of strings is associative.

1.7.2. Prove each of the following using the inductive definition of reversal given in the text.

(a) $(w^R)^R = w$ for any string w .

(b) If v is a substring of w , then v^R is a substring of w^R .

(c) $(w^i)^R = (w^R)^i$ for any string w and $i \geq 0$.

1.7.3. Let $\Sigma = \{a_1, \dots, a_{26}\}$ be the Roman alphabet. Carefully define the binary relation $<$ on Σ^* such that $x < y$ if and only if x would precede y in a standard dictionary.

1.7.4. Show each of the following.

(a) $\{e\}^* = \{e\}$

(b) For any alphabet Σ and any $L \subseteq \Sigma^*$, $(L^*)^* = L^*$.

(c) If a and b are distinct symbols, then $\{a, b\}^* = \{a\}^* \{b\}^* \{a\}^*$.

(d) If Σ is any alphabet, $e \in L_1 \subseteq \Sigma^*$ and $e \in L_2 \subseteq \Sigma^*$, then $(L_1 \Sigma^* L_2)^* = \Sigma^*$.

(e) For any language L , $\emptyset L = L \emptyset = \emptyset$.

1.7.5. Give some examples of strings in, and not in, these sets, where $\Sigma = \{a, b\}$.

(a) $\{w : \text{for some } u \in \Sigma\Sigma, w = uu^R u\}$

(b) $\{w : ww = wvw\}$

(c) $\{w : \text{for some } u, v \in \Sigma^*, uvw = wvu\}$

(d) $\{w : \text{for some } u \in \Sigma^*, uvw = wu\}$

1.7.6. Under what circumstances is $L^+ = L^* - \{e\}$?

1.7.7. The Kleene star of a language L is the closure of L under which relations?

1.8 FINITE REPRESENTATIONS OF LANGUAGES

A central issue in the theory of computation is the representation of languages by finite specifications. Naturally, any finite language is amenable to finite representation by exhaustive enumeration of all the strings in the language. The issue becomes challenging only when infinite languages are considered.

Let us be somewhat more precise about the notion of "finite representation of a language." The first point to be made is that any such representation must itself be a string, a finite sequence of symbols over some alphabet Σ . Second, we certainly want different languages to have different representations, otherwise the term *representation* could hardly be considered appropriate. But these two requirements already imply that the possibilities for finite representation are severely limited. For the set Σ^* of strings over an alphabet Σ is countably infinite, so the number of possible representations of languages is countably infinite. (This would remain true even if we were not bound to use a particular alphabet Σ , so long as the total number of available symbols was countably infinite.) On the other hand, the set of all possible languages over a given alphabet Σ —that is, 2^{Σ^*} —is uncountably infinite, since 2^{\aleph} , and hence the power set of any countably infinite set is not countably infinite. With only a countable number of representations and an uncountable number of things to represent, we are unable to represent all languages finitely. Thus, the most we can hope for is to find finite representations, of one sort or another, for at least some of the more interesting languages.

This is our first result in the theory of computation: No matter how powerful are the methods we use for representing languages, only countably many languages can be represented, so long as the representations themselves are finite. There being uncountably many languages in all, the vast majority of them will inevitably be missed under any finite representational scheme.

Of course, this is not the last thing we shall have to say along these lines. We shall describe several ways of describing and representing languages, each more powerful than the last in the sense that each is capable of describing languages the previous one cannot. This hierarchy does not contradict the fact that all these finite representational methods are inevitably limited in scope for the reasons just explained.

We shall also want to derive ways of exhibiting particular languages that cannot be represented by the various representational methods we study. We know that the world of languages is inhabited by vast numbers of such unrepresentable specimens, but, strangely perhaps, it can be exceedingly difficult to catch one, put it on display, and document it. Diagonalization arguments will eventually assist us here.

To begin our study of finite representations, we consider expressions —

strings of symbols—that describe how languages can be built up by using the operations described in the previous section.

Example 1.8.1: Let $L = \{w \in \{0, 1\}^* : w \text{ has two or three occurrences of } 1, \text{ the first and second of which are not consecutive}\}$. This language can be described using only singleton sets and the symbols \cup , \circ , and $*$ as

$$\{0\}^* \circ \{1\} \circ \{0\}^* \circ \{0\} \circ \{1\} \circ \{0\}^* \circ (\{1\} \circ \{0\}^*) \cup \emptyset^*$$

It is not hard to see that the language represented by the above expression is precisely the language L defined above. The important thing to notice is that the only symbols used in this representation are the braces $\{$ and $\}$, the parentheses $($ and $)$, \emptyset , 0 , 1 , $*$, \circ , and \cup . In fact, we may dispense with the braces and \circ and write simply

$$L = 0^* 10^* 010^* (10^* \cup \emptyset^*)$$

◇

Roughly speaking, an expression such as the one for L in Example 1.8.1 is called a **regular expression**. That is, a regular expression describes a language exclusively by means of single symbols and \emptyset , combined perhaps with the symbols \cup and $*$, possibly with the aid of parentheses. But in order to keep straight the expressions about which we are talking and the “mathematical English” we are using for discussing them, we must tread rather carefully. Instead of using \cup , $*$, and \emptyset , which are the names in this book for certain operations and sets, we introduce special symbols \cup , $*$, and \emptyset , which should be regarded for the moment as completely free of meaningful overtones, just like the symbols a , b , and 0 used in earlier examples. In the same way, we introduce special symbols $($ and $)$ instead of the parentheses $($ and $)$ we have been using for doing mathematics. The regular expressions over an alphabet Σ^* are all strings over the alphabet $\Sigma \cup \{(\cdot), \emptyset, \cup, *\}$ that can be obtained as follows.

- (1) \emptyset and each member of Σ is a regular expression.
- (2) If α and β are regular expressions, then so is $(\alpha\beta)$.
- (3) If α and β are regular expressions, then so is $(\alpha \cup \beta)$.
- (4) If α is a regular expression, then so is α^* .
- (5) Nothing is a regular expression unless it follows from (1) through (4).

Every regular expression represents a language, according to the interpretation of the symbols \cup and $*$ as set union and Kleene star, and of juxtaposition of expressions as concatenation. Formally, the relation between regular expressions and the languages they represent is established by a function \mathcal{L} , such that if α is any regular expression, then $\mathcal{L}(\alpha)$ is the language represented by α . That is, \mathcal{L} is a function from strings to languages. The function \mathcal{L} is defined as follows.

- (1) $\mathcal{L}(\emptyset) = \emptyset$, and $\mathcal{L}(a) = \{a\}$ for each $a \in \Sigma$.
- (2) If α and β are regular expressions, then $\mathcal{L}((\alpha\beta)) = \mathcal{L}(\alpha)\mathcal{L}(\beta)$.
- (3) If α and β are regular expressions, then $\mathcal{L}((\alpha \cup \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$.
- (4) If α is a regular expression, then $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$.

Statement 1 defines $\mathcal{L}(\alpha)$ for each regular expression α that consists of a single symbol; if $n > 1$, then (2) through (4) define $\mathcal{L}(\alpha)$ for regular expressions of some length in terms of $\mathcal{L}(\alpha')$ for one or two regular expressions α' of smaller length. Thus every regular expression is associated in this way with some language.

Example 1.8.2: What is $\mathcal{L}(((a\cup b)^* a))$? We have the following.

$$\begin{aligned} \mathcal{L}(((a\cup b)^* a)) &= \mathcal{L}((a\cup b)^* \mathcal{L}(a)) \text{ by (2)} \\ &= \mathcal{L}((a\cup b)^* \{a\}) \text{ by (1)} \\ &= \mathcal{L}((a\cup b))^* \{a\} \text{ by (4)} \\ &= (\mathcal{L}(a) \cup \mathcal{L}(b))^* \{a\} \text{ by (3)} \\ &= (\{a\} \cup \{b\})^* \{a\} \text{ by (1) twice} \\ &= \{a, b\}^* \{a\} \\ &= \{w \in \{a, b\}^* : w \text{ ends with an } a\} \end{aligned}$$

◇

Example 1.8.3: What language is represented by $(c^*(a\cup(bc^*))^*)$? This regular expression represents the set of all strings over $\{a, b, c\}$ that do not have the substring ac . Clearly no string in $\mathcal{L}((c^*(a\cup(bc^*))^*))$ can contain the substring ac , since each occurrence of a in such a string is either at the end of the string, or is followed by another occurrence of a , or is followed by an occurrence of b . On the other hand, let w be a string with no substring ac . Then w begins with zero or more c 's. If they are removed, the result is a string with no sub-string ac and not beginning with c . Any such string is in $\mathcal{L}((a\cup(bc^*))^*)$; for it can be read, left to right, as a sequence of a 's, b 's, and c 's, with any blocks of c 's immediately following b 's (not following a 's, and not at the beginning of the string). Therefore $w \in \mathcal{L}((c^*(a\cup(bc^*))^*))$. ◇

Example 1.8.4: $(0^* \cup (((0^*(1\cup(11)))((00^*)(1\cup(11)))^*)0^*))$ represents the set of all strings over $\{0, 1\}$ that do not have the substring 111. ◇

Every language that can be represented by a regular expression can be represented by infinitely many of them. For example, α and $(\alpha \cup \emptyset)$ always represent the same language; so do $((\alpha \cup \beta) \cup \gamma)$ and $(\alpha \cup (\beta \cup \gamma))$. Since set union

and concatenation are associative operations—that is, since $(L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$ for all L_1, L_2, L_3 , and the same for concatenation—we normally omit the extra (and) symbols in regular expressions; for example, we treat $aUUbUc$ as a regular expression even though “officially” it is not. For another example, the regular expression of Example 1.8.4 might be rewritten as $0^*U0^*(1U11)(00^*(1U11))^*0^*$.

Moreover, now that we have shown that regular expressions and the languages they represent can be defined formally and unambiguously, we feel free, when no confusion can result, to blur the distinction between the regular expressions and the “mathematical English” we are using for talking about languages. Thus we may say at one point that a^*b^* is the set of all strings consisting of some number of a 's followed by some number of b 's—to be precise, we should have written $\{a\}^* \circ \{b\}^*$. At another point, we might say that a^*b^* is a regular expression representing that set; in this case, to be precise, we should have written (a^*b^*) .

The class of **regular languages** over an alphabet Σ is defined to consist of all languages L such that $L = \mathcal{L}(\alpha)$ for some regular expression α over Σ . That is, regular languages are all languages that can be described by regular expressions. Alternatively, regular languages can be thought of in terms of *closures*. The class of regular languages over Σ is precisely the closure of the set of languages

$$\{\{\sigma\} : \sigma \in \Sigma\} \cup \{\emptyset\}$$

with respect to the functions of union, concatenation, and Kleene star.

We have already seen that regular expressions do describe some nontrivial and interesting languages. Unfortunately, we cannot describe by regular expressions some languages that have very simple descriptions by other means. For example, $\{0^n 1^n : n \geq 0\}$ will be shown in Chapter 2 *not* to be regular. Surely any theory of the finite representation of languages will have to accommodate at least such simple languages as this. Thus regular expressions are an inadequate specification method in general.

In search of a general method for finitely specifying languages, we might return to our general scheme

$$L = \{w \in \Sigma^* : w \text{ has property } P\}.$$

But which properties P should we entail? For example, what makes the preceding properties, “ w consists of a number of 0's followed by an equal number of 1's” and “ w has no occurrence of 111” such obvious candidates? The reader may ponder about the right answer; but let us for now allow *algorithmic properties*, and only these. That is, for a property P of strings to be admissible as a specification of a language, there must be an algorithm for deciding whether a given string belongs to the language. An algorithm that is specifically designed,

for some language L , to answer questions of the form “Is string w a member of L ?” will be called a **language recognition device**. For example, a device for recognizing the language

$$L = \{w \in \{0, 1\}^* : w \text{ does not have } 111 \text{ as a substring}\}.$$

by reading strings, a symbol at a time, from left to right, might operate like this: Keep a count, which starts at zero and is set back to zero every time a 0 is encountered in the input; add one every time a 1 is encountered in the input; stop with a No answer if the count ever reaches three, and stop with a Yes answer if the whole string is read without the count reaching three.

An alternative and somewhat orthogonal method for specifying a language is to describe how a generic specimen in the language is produced. For example, a regular expression such as $(e \cup b \cup bb)(a \cup ab \cup abb)^*$ may be viewed as a way of *generating* members of a language:

To produce a member of L , first write down either nothing, or b , or bb ; then write down a or ab , or abb , and do this any number of times, including zero; all and only members of L can be produced in this way.

Such language generators are not algorithms, since they are not designed to answer questions and are not completely explicit about what to do (how are we to choose which of a , ab , or abb is to be written down?) But they are important and useful means of representing languages all the same. The relation between language recognition devices and language generators, both of which are types of finite language specifications, is another major subject of this book.

Problems for Section 1.8

1.8.1. What language is represented by the regular expression $((a^*a)b)Ub$?

1.8.2. Rewrite each of these regular expressions as a simpler expression representing the same set.

- $\emptyset^*Ua^*Ub^*U(aUb)^*$
- $((a^*b^*)^*(b^*a^*)^*)^*$
- $(a^*b)^*U(b^*a)^*$
- $(aUb)^*a(aUb)^*$

1.8.3. Let $\Sigma = \{a, b\}$. Write regular expressions for the following sets:

- All strings in Σ^* with no more than three a 's.
- All strings in Σ^* with a number of a 's divisible by three.
- All strings in Σ^* with exactly one occurrence of the substring aaa .

1.8.4. Prove that if L is regular, then so is $L' = \{w : uw \in L \text{ for some string } u\}$. (Show how to construct a regular expression for L' from one for L .)

1.8.5. Which of the following are true? Explain.

- (a) $baa \in a^*b^*a^*b^*$
- (b) $b^*a^* \cap a^*b^* = a^* \cup b^*$
- (c) $a^*b^* \cap b^*c^* = \emptyset$
- (d) $abcd \in (a(cd)^*b)^*$

1.8.6. The star height $h(\alpha)$ of a regular expression α is defined by induction as follows.

$$h(\emptyset) = 0$$

$$h(a) = 0 \text{ for each } a \in \Sigma$$

$$h(\alpha\cup\beta) = h(\alpha\beta) = \text{the maximum of } h(\alpha) \text{ and } h(\beta).$$

$$h(\alpha^*) = h(\alpha) + 1$$

For example, if $\alpha = ((ab)^*Ub^*)^*Ua^*$, then $h(\alpha) = 2$. Find, in each case, a regular expression which represents the same language and has star height as small as possible.

- (a) $((abc)^*ab)^*$
- (b) $(a(ab^*c)^*)^*$
- (c) $(c(a^*b)^*)^*$
- (d) $(a^*Ub^*Uab)^*$
- (e) $(abb^*a)^*$

1.8.7. A regular expression is in **disjunctive normal form** if it is of the form $(\alpha_1\cup\alpha_2\cup\cdots\cup\alpha_n)$ for some $n \geq 1$, where none of the α_i 's contains an occurrence of \cup . Show that every regular language is represented by one in disjunctive normal form.

REFERENCES

- An excellent source on informal set theory is the book*
- o P. Halmos *Naive Set Theory*, Princeton, N.J.: D. Van Nostrand, 1960.
- A splendid book on mathematical induction is*
- o G. Polya *Induction and Analogy in Mathematics*, Princeton, N.J.: Princeton University Press, 1954.
- A number of examples of applications of the pigeonhole principle appear in the first chapter of*
- o C. L. Liu *Topics in Combinatorial Mathematics*, Buffalo, N.Y.: Mathematical Association of America, 1972.
- Cantor's original diagonalization argument can be found in*
- o G. Cantor *Contributions to the Foundations of the Theory of Transfinite Numbers* New York: Dover Publications, 1947.
- The \mathcal{O} -notation and several variants were introduced in*

References

- o D. E. Knuth "Big omicron and big omega and big theta," *ACM SIGACT News*, 8 (2), pp. 18-23, 1976.
- The $\mathcal{O}(n^3)$ algorithm for the reflexive-transitive closure is from*
- o S. Warshall "A theorem on Boolean matrices," *Journal of the ACM*, 9, 1, pp. 11-12, 1962.
- Two books on algorithms and their analysis are*
- o T. H. Cormen, C. E. Leiserson, R. L. Rivest *Introduction to Algorithms*, Cambridge, Mass.: The MIT Press, 1990, and
 - o G. Brassard, P. Bratley *Fundamentals of Algorithms*, Englewood Cliffs, N.J.: Prentice Hall, 1996.
- Two advanced books on language theory are*
- o A. Salomaa *Formal Languages* New York: Academic Press, 1973.
 - o M. A. Harrison *Introduction to Formal Language Theory*, Reading, Massach.: Addison-Wesley, 1978.