Chapter 4

Alphabets, Strings, and Languages

In the field of computer science, the word "language" has a precise and easily-stated definition that is extremely broad, encompassing everything from familiar programming languages to very abstract problems of computer science. Simple languages also show up in numerous familiar contexts such as word processors, text editors, and search engines.

To distinguish the mathematically-defined languages that we will study in the context of computer science from those we employ in communicating to fellow humans, we refer to the former as *formal languages* and the latter as *natural languages*.

This chapter introduces the basic concepts needed to study formal languages, which are used to frame many problems of computer science. As we shall see, many fundamental problems can be viewed as *language recognition* problems. In fact, our first view of *automata*, or abstract machines, shall be as language recognizers. The books by Lewis and Papadimitiou [17], Sipser [21], Sudkamp [23], Howie [11], and Linz [18] are recommended for further reading on these topics. We start by defining *alphabets* and *strings*, which will be the building blocks of formal languages.

4.1 Alphabets and Strings

An *alphabet* is a non-empty finite set whose elements are referred to as *letters*, *symbols*, or *characters*. We will typically denote an alphabet by the symbol Σ, or by explicitly listing its letters as elements of a set, such as $\{a, b, c\}$. A *string* is a finite (possibly zero-length) sequence of symbols over some alphabet, denoted by juxtaposing the symbols in the sequence. For instance, `aardvark` is a string over the standard 26-letter alphabet used in the west, and `ab#aa` is a string over any alphabet containing the symbols "`a`", "`b`", and "`#`."

If u and v both denote strings, then $u \circ v$ denotes the *concatenation* of u and v, which is the string obtained by juxtaposing the letters of u and the letters of v. If $u = abc$, and $v = xyz$, then $u \circ v$ is the string $abcxyz$. We will frequently omit the concatenation operator "$\circ$" and simply write uv for the concatenation of strings u and v, which is the same convention commonly employed for denoting multiplicative operations. As a special case of string concatenation, we will often write "au" to denote the letter a, viewed as a one-letter string, concatenated with the string u.

If Σ is an alphabet and w is a string over Σ, then $|w|$ denotes the total number of letters in w. A useful extension of this notation is $|w|_x$, by which we shall mean the number of occurrences of the letter x in the string w. Hence, $|abbaa|_a = 3$. By $w[i]$ we mean the ith symbol of w, with $w[1]$ being the first letter of the string, and by w^R we mean the string consisting of the same sequence of symbols as w, but in reverse order. We could define w^R to be the string such that

$$w^R[k] \;=\; w[\,|w| - k + 1\,],$$

for $k = 1, 2, \ldots, |w|$. If a string v consists of a contiguous sequence of the symbols found in another string w, then we refer to v as a *substring* of w. That is, if for some $k \geq 0$ and $m \geq 0$, $v[i] \;=\; w[i+k]$ for $i = 1, 2, \ldots, m$, then v is a substring of w. The *empty string* is the string consisting of zero symbols. We denote the empty string by ε, which we shall always assume is a symbol that is distinct from any other symbol in the alphabet under consideration. If $\Sigma = \{a, b\}$, then ε, a, $aaaa$, and $abbaab$ are all strings over Σ. For any string w, both ε and w are always considered to be substrings.

A symbol or a string with a natural number exponent, written as a^n, denotes the string consisting of n concatenated copies of a; this operation is referred to as *iterated concatenation*. By definition, $a^0 = \varepsilon$, the empty string. Let Σ denote any alphabet. Then Σ^* is the *Kleene Star* of Σ, which is the set of all strings over Σ. That is,

$$\Sigma^* \;\overset{\text{def}}{=}\; \{c_1 c_2 \cdots c_k \mid k \in \mathbb{N}, c_i \in \Sigma \text{ for each } i\}\,. \tag{4.1}$$

As special cases, observe that Σ^* always contains the empty string, ε, all the *singleton strings*, which are simply the symbols of Σ interpreted as strings of length one, and all the singletons raised to all positive integer powers. If $\Sigma = \{a, b\}$, then

$$\Sigma \;=\; \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}\,. \tag{4.2}$$

It is easy to see that Σ^* is a countably infinite set for any alphabet Σ. One way to see this is to observe that

$$\Sigma^* \;=\; \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \cdots$$

where Σ_k consists of all strings over Σ that consist of exactly k symbols. Since Σ_k is finite, and therefore countable, for all k, Σ^* is the countable union of countable sets. It follows that Σ^* is countable for any alphabet Σ.

Both *string reversal* and *iterated concatenation* can be conveniently defined using *inductive definitions*, as shown below.

Definition 9 For any $w \in \Sigma^*$, the *reversal* w^R is defined inductively by

$$\begin{aligned} \varepsilon^R &= \varepsilon, \\ (au)^R &= u^R a, \end{aligned}$$

where $a \in \Sigma$ and $u \in \Sigma^*$ are such that $w = au$.

Definition 10 For any $w \in \Sigma^*$, the *iterated concatenation* w^n is defined inductively by

$$\begin{aligned} w^0 &= \varepsilon, \\ w^n &= w \circ w^{n-1}, \end{aligned}$$

where $n \in \mathbb{N} - \{0\}$, and $u \circ v$ denotes string concatenation.

In both cases the definitions are self-referential in that the definition for a string of length one or more is stated in terms of shorter strings. Each definition also includes a rule that handles trivial cases. Definitions of this form can be used to define patterns such as well-formed propositional formulas and Lisp S-expressions by building them up from trivial formulas and expressions.

Inductive definitions are frequently convenient for proving statements about the operators, particularly when the proof is by induction. The example of reversing the concatenation of two strings will illustrate this point.

Theorem 14 *For any $u, v \in \Sigma^*$, $(uv)^R = v^R u^R$.*

Proof: Let u and v be strings in Σ^*. We shall prove the theorem by induction on the length of u. Thus, we let $P(n)$ be true iff the theorem holds for all $u \in \Sigma^*$ with $|u| = n$. First, $|u| = 0$ implies that $u = \varepsilon$, and $(\varepsilon v)^R = v^R = v^R\varepsilon = v^R\varepsilon^R$, which verifies the basis step, $P(0)$. Next, assume that the theorem holds for $|u| = n$ (the inductive hypothesis), and let $a \in \Sigma$. Then $(auv)^R = (uv)^R a$, by the inductive definition of string reversal and the associativity of string concatenation. But $(uv)^R a = v^R u^R a$, by the inductive hypothesis, since $|u| = n$. Finally, $v^R u^R a = v^R (au)^R$, again by the definition of string reversal. Because au is an arbitrary string of length $n+1$, and we have shown that $(auv)^R = v^R(au)^R$, it follows that $P(n+1)$ holds, which verifies the induction step. Therefore, by induction, the formula holds for all $n \in \mathbb{N}$. □

Since Σ^* is countably infinite, it can be put into one-to-one correspondence with the natural numbers $\mathbb{N}$. The most common method of doing this is by means of *lexicographic ordering*, which we will define to be different from standard "dictionary" ordering [1]. If u and v are two strings in Σ^*, then we define the binary relation "$<$" on the set $\Sigma^* \times \Sigma^*$ to mean

$$u < v \quad \text{if and only if} \quad \begin{cases} |u| < |v| \quad \text{or} \\ \\ |u| = |v| \quad \text{and} \quad u[k] < v[k] \quad \text{for some} \quad k \in \{1, 2, \ldots, |u|\} \\ \quad \text{such that} \quad u[i] = v[i] \quad \text{for all} \quad i \in \{1, 2, \ldots, k-1\}, \end{cases}$$

where the relation $<$ on the right-hand side is used in two distinct ways: both as the standard "less than" predicate on the natural numbers, and as the natural *alphabetical* order relation on Σ; that is, $u[k] < v[k]$ means that the symbol $u[k]$ comes before the symbol $v[k]$ in some given ordering of the letters in the alphabet. As an example, let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\},$$

[1] Many text books use the terms "lexicographic ordering" and "dictionary ordering" synonymously. We shall find it convenient to reserve the word "lexicographic" for the more robust type of ordering that works for both finite and countably infinite sets.

where the entries of the set are shown in lexicographic order with respect the the obvious alphabetical ordering of the letters in Σ. As we have defined it, lexicographic ordering is *not* the same as standard "dictionary" ordering. In a dictionary the word "aardvark" is listed before the word "emu." However, "emu" is first lexicographically because it has fewer letters. This is an important difference when dealing with infinite collections of strings, since dictionary ordering of the set $\{a, b\}^*$ would place all elements of the set $\{\varepsilon, a, aa, aaa, aaaa, \ldots\}$ before the first string containing a "b," which fails to define a one-to-one correspondence between $\{a, b\}^*$ and $\{0, 1, 2, \ldots\}$.

4.2 Languages

A *language* (over a given alphabet) is simply a set of strings; the set may be finite, infinite, or even empty. That is, if Σ is an alphabet, then every subset $L \subseteq \Sigma^*$ is a language over Σ. The elements of any language can be put into lexicographic order in the same way that Σ^* can be ordered, as described above. The following sets are examples of languages over the alphabet $\{a, b\}$.

$$\begin{aligned}
L_1 &= \{\, a, abb, aaaa \} \\
L_2 &= \{\, a^n \mid n \in \mathbb{N} \text{ is prime} \} \\
L_3 &= \{\, b^n a^n b^m \mid n, m \in \mathbb{N} \text{ and } n = m \mod 3 \,\} \\
L_4 &= \text{The set of all } w \in \Sigma^* \text{ with at most three } a\text{'s} \\
L_5 &= \{\, a^n \mid n \in \mathbb{N} \text{ and } \exists\, x, y, z \in \mathbb{N} - \{0\} \text{ such that } x^n + y^n = z^n \}
\end{aligned}$$

The languages L_1 and L_5 are finite sets, whereas L_2, L_3, and L_4 are countably infinite. Although language L_5 is today known to be $\{a, aa\}$, because Fermat's last theorem has been proven, it illustrates how the definition of a language (even over a trivial alphabet) can encode the answers to deep questions of number theory.

Since languages are sets, one can define new languages with set operations. For example, given any languages L_1 and L_2 over an alphabet Σ, the sets $L_1 \cup L_2$ and $L_1 \cap L_2$ are also languages over Σ. Another means of defining a new language from a given one is with the *Kleene star* operator. If L is a language over Σ, then L^* is the set of all strings formed by *concatenating* (joining together) a finite number of strings from L. That is,

$$L^* = \{ w_1 w_2 \cdots w_k \mid k \in \mathbb{N} \text{ and } w_i \in L \text{ for } i = 1, 2, \ldots k \},$$

where string concatenation is denoted by juxtaposing the symbols representing strings. Notice that the definition of Σ^* above is consistent with the definition of the Kleene star operator; it corresponds to the language obtained by taking the Kleene star of the language consisting of all strings of length one.

A language consisting of an infinite number of strings may be referred to as an *infinite language* if it is important to emphasize the cardinality of the language. A language consisting of a finite number of strings may similarly be referred to as a *finite language*. Observe that so long as a language L is non-empty (i.e. it contains at least one string), then L^* is an infinite language.

Using the concept of set cardinality we can place some very general limits on what can be expressed through *symbolic representations*, whether they be mathematical formulas, sentences in a natural language such as English or French, or computer programs written in a programming language such as C, Lisp, or Java.

The languages $L_1, \ldots, L_5$ in section 4.2 are all described using a finite number of symbols, such as "{", "}", "n", and "m", in addition to the symbols of the original alphabet $\{a, b\}$. Since any such description must be of finite length, we can view the definitions themselves as being strings in a *meta-language*; in this case a language that is a superset of both English and formal mathematics. This raises an interesting question: Are *all* languages (over a given alphabet) defined by *some* string in a suitable meta-language? We can answer this question negatively using only the tools of set theory.

Let M denote the alphabet of the meta-language and observe that the set M^* of all strings in this language is countably infinite. While the set of all strings over Σ is also countably infinite, the set of all *languages* over Σ has the cardinality of 2^{Σ^*}, which is uncountable. Since $|M^*| < \left|2^{\Sigma^*}\right|$, it follows that there cannot be a surjection from M^*, or any other language over M, onto the set of all languages over Σ; there simply are not enough strings in the meta-language to go around. We must conclude that, regardless of the meta-language used, there must exist languages over Σ that have *no finite description*.

Since languages are sets, all the the traditional set operations can be used to form new languages from existing languages. For example, if L and L' are languages over a common alphabet Σ, then so are the sets $L \cup L'$, $L \cap L'$, $L - L'$, L^*, $\overline{L}$, and $L \circ L'$, where

$$\begin{aligned}
L - L' &\overset{\text{def}}{=} \{w \in L \mid w \notin L'\} \\
\overline{L} &\overset{\text{def}}{=} \Sigma^* - L \\
L^* &\overset{\text{def}}{=} \{w_1 w_2 \cdots w_n \mid n \in \mathbb{N}, w_i \in L\} \\
L \circ L' &\overset{\text{def}}{=} \{uv \mid u \in L, v \in L'\} \\
L^R &\overset{\text{def}}{=} \left\{w^R \mid w \in L\right\}.
\end{aligned}$$

The operations denoted by "$\cup$", "$\cap$", "$-$", and the bar are simply the standard set operations of union, intersection, difference, and complementation, respectively. The sets denoted by L^* and $L \circ L'$ are called the *Kleene star* and the *concatenation*, respectively. The Kleene star operator creates a new set of strings by concatenating every finite sequence of original strings; by definition, this includes the empty string ε, which is the concatenation of zero strings. The Kleene star operator applied to an alphabet, Σ^*, as defined earlier, is simply a special case of the operator defined here; that is, we can view Σ as the set of all singleton strings over Σ.

Let $\Sigma = \{0, 1, a, b\}$, and let L_1 and L_2 be the following trivial languages over Σ:

$$\begin{aligned}
L_1 &= \{00, 1a, a, aa\} \\
L_2 &= \{a, ab, a01\}
\end{aligned}$$

then

$$\begin{aligned}
L_1 - L_2 &= \{00, 1a, aa\} \\
{L_2}^* &= \{\varepsilon, a, aa, aaa, ab, aab, abab, aba, aa01, ab01, \ldots\} \\
L_1 \circ L_2 &= \{00a, 00ab, 00a01, 1aa, 1aab, 1aa01, \ldots\} \\
\overline{L_1} &= \{0, 1, b, 11, bb, 01, 0a, 0b, 10, 11, 1b, 000, \ldots\}.
\end{aligned}$$

The elements of the infinite sets shown above are merely intended to give an impression of the set; the ellipsis does not indicate an obvious pattern that is to be continued. This raises the question as to whether there exists a natural way to list such elements. The answer is yes, as we discuss in the following section.

4.3 Regular Languages

In many contexts it suffices to define a trivial language. For example, database queries, search engine queries, command-line arguments, string replacement operations, and defining numbers and identifiers in a programming language all involve very simple patterns of symbols, despite the fact that they are (in principle) infinite languages. In particular, the strings in such languages can frequently be specified in terms of several very basic operations:

1. **Concatenation of strings**
2. **Alternative substrings**
3. **Repeated substrings**

For example, we could define the language of proper binary numbers over the alphabet $\Sigma = \{0, 1\}$ to be the strings consisting of either 0, or a 1 followed by zero or more 0's and 1's, in any combination. A reasonable notation for this might be

$$0 \mid \big(1\,(0 \mid 1)^*\big)\,,$$

where "|" indicates an either-or choice, and the star notation indicates that the expression within can be repeated any number of times, including zero. Juxtaposing the "1" and the starred expression indicates concatenation. If we further allow an optional sign in front of the binary number, we could express the collection of all such strings over the alphabet $\Sigma = \{0, 1, -, +\}$ by

$$(+ \mid - \mid \varepsilon)\ \big(0 \mid \big(1\,(0 \mid 1)^*\big)\big)\,,$$

where ε denotes the empty string. The two expressions above make use of each of the operations mentioned earlier: concatenation, alternatives, and repetition. In addition to symbols of the alphabet and the operators "|" and "$*$", these expression also contains two meta-symbols: "(" and ")", which help to define the scope of the operators. Such strings themselves form a language, each string of which can be interpreted as *defining* a language. We will now make these ideas more precise.

The language of the signed binary numbers is actually an example of a very common form of language, known as a *regular language*. With just a few symbols we were able to define some elementary operations that allowed us to unambiguously specify an infinite collection of strings that meet our specification. We shall now take this concept and develop it more carefully by first defining the language of *regular expressions* themselves, and then defining precisely how such expressions are to be interpreted.

Given an alphabet Σ, we can express the large and important class of regular languages over Σ by means of another special-purpose language; the language of regular expressions. To construct regular expressions for encoding languages over any given alphabet Σ, we first introduce an expanded alphabet:

$$\widehat{\Sigma} \;\stackrel{\text{def}}{=}\; \Sigma \cup \{\text{ “(”, “)”, “*”, “|”, “}\emptyset\text{” }\}$$

This alphabet extends the original alphabet by adding several special symbols, which we shall assume are distinct from the symbols in Σ. Next, we give an inductive definition of the regular expressions, which form a language R_Σ over the extended alphabet $\widehat{\Sigma}$. These rules precisely define the *form*, or *syntax*, of the regular expressions.

1. $\emptyset \in R_\Sigma$
2. $\Sigma \subset R_\Sigma$
3. $u, v \in R_\Sigma \implies (u \,|\, v) \in R_\Sigma$
4. $u, v \in R_\Sigma \implies (uv) \in R_\Sigma$
5. $u \in R_\Sigma \implies (u)^* \in R_\Sigma$

Moreover, only strings that can be constructed by the applications of these rules are in R_Σ. Since the strings of any language are of finite length (by definition), each element of R_Σ must result from the application of only a finite number of these rules. We now define a function

$$\mathfrak{R} : R_\Sigma \to 2^{\Sigma^*}$$

that associates each string in R_Σ with a (possibly infinite) language over Σ. That is,

$$\mathfrak{R}(\text{ \textit{a regular expression over} } \widehat{\Sigma}\ ) \;=\; \text{\textit{the corresponding regular language over} } \Sigma.$$

Hence, the frunction $\mathfrak{R}$ provides the *meaning*, or *semantics*, of each regular expression by mapping it to the language it represents. We define the function $\mathfrak{R}$ inductively with the following collection of rules, which are exactly analogous to the rules used to form the regular expressions themselves. These rules define the meaning, or *semantics*, of the regular expressions. Here “a” denotes a symbol in Σ, and “u” and “v” denote strings in R_Σ.

1. $\mathfrak{R}(\ \emptyset\ ) = \emptyset$
2. $\mathfrak{R}(\ a\ ) = \{a\}$
3. $\mathfrak{R}(\ (u \,|\, v)\ ) = \mathfrak{R}(u) \cup \mathfrak{R}(v)$
4. $\mathfrak{R}(\ (uv)\ ) = \mathfrak{R}(u) \circ \mathfrak{R}(v)$
5. $\mathfrak{R}(\ (u)^*\ ) = \mathfrak{R}(u)^*$

Note that the arguments to the function $\mathfrak{R}$ above are to be interpreted as strings over the alphabet $\widehat{\Sigma}$, while the objects on the right hand sides are to be interpreted as sets and operations on sets. Thus, in rule 1 above, the “$\emptyset$” on the left denotes a symbol in $\widehat{\Sigma}$, while on the right it denotes the empty set. Similarly, the “$*$” on the left denotes a symbol in $\widehat{\Sigma}$, while on the right it denotes the Kleene star operator. It is by virtue of multiple interpretations of the same symbols that the function $\mathfrak{R}$ provides the semantics of regular expressions; that is, it provides the meaning of certain strings over $\widehat{\Sigma}$.

It is possible to interpret any regular expression as a set of strings by directly applying the function $\mathfrak{R}$ defined above. For example, let $\Sigma = \{a, b\}$ and consider the regular expression $((a)^*(b)^*)$. To interpret this string as a language, we simply apply the rules defining $\mathfrak{R}$, starting with a rule that is applicable to the entire string. Thus, we have

$$\begin{aligned}
\mathfrak{R}(\ ((a)^*(b)^*)\ ) &= \mathfrak{R}(\ (a)^*\ )\ \circ\ \mathfrak{R}(\ (b)^*\ ) \\
&= \mathfrak{R}(\ a\ )^*\ \circ\ \mathfrak{R}(\ b\ )^* \\
&= \{a\}^*\ \circ\ \{b\}^* \\
&= \{a^n \mid n \geq 0\}\ \circ\ \{b^n \mid n \geq 0\} \\
&= \left\{a^n b^k \mid n \geq 0,\, k \geq 0\right\}.
\end{aligned}$$

As a second example, we'll derive the meaning of the expression $(a \,|\, \emptyset^*)(aa \,|\, bb)^*$. Here we shall drop some of the parentheses where the meaning is clear without them.

$$\begin{aligned}
\mathfrak{R}(\ (a \,|\, \emptyset^*)(aa \,|\, bb)^*\ ) &= \mathfrak{R}(\ a \,|\, \emptyset^*\ )\ \circ\ \mathfrak{R}(\ (aa \,|\, bb)^*\ ) \\
&= (\mathfrak{R}(a) \cup \mathfrak{R}(\emptyset^*))\ \circ\ \mathfrak{R}(\ aa \,|\, bb\ )^* \\
&= (\{a\} \cup \emptyset^*)\ \circ\ (\mathfrak{R}(aa) \cup \mathfrak{R}(bb))^* \\
&= (\{a\} \cup \{\varepsilon\})\ \circ\ ((\mathfrak{R}(a) \circ \mathfrak{R}(a)) \cup (\mathfrak{R}(b) \cup \mathfrak{R}(b)))^* \\
&= \{a, \varepsilon\}\ \circ\ ((\{a\} \circ \{a\}) \cup (\{b\} \cup \{b\}))^* \\
&= \{a, \varepsilon\}\ \circ\ (\{aa\} \cup \{bb\})^* \\
&= \{a, \varepsilon\}\ \circ\ \{aa, bb\}^* \\
&= \left\{a^k w_1 w_2 \cdots w_n \mid k \in \{0, 1\}, n \in \mathbb{N}, w_i \in \{aa, bb\}\right\}
\end{aligned}$$

Both the definition of regular expressions and the definition of $\mathfrak{R}$ used parentheses liberally to avoid confusion. However, as we saw above, the parentheses are frequently redundant. This follows from that fact that both union and concatenation operations are associative. Consequently, it is customary to eliminate the unnecessary parentheses, making the following identifications

$$\begin{aligned}
((ab)c) &= abc \\
((a \,|\, b) \,|\, c) &= a \,|\, b \,|\, c \\
(a)^* &= a^*
\end{aligned}$$

Of course, not all parentheses are redundant, as the following examples demonstrate.

$$\begin{aligned}
(a \,|\, b)c &\neq a \,|\, bc \\
(a \,|\, b)^* &\neq a \,|\, b^* \\
(ab)^* &\neq ab^*
\end{aligned}$$

We call $L \subseteq \Sigma^*$ a regular language if and only if it can be represented as a regular expression over Σ. Note that both the empty language $\emptyset$ and the language consisting of only the empty string, $\{\varepsilon\}$, are regular languages. (Be sure to understand that these languages are distinct.) The empty language is regular because $\emptyset$ is explicitly included in the extended alphabet $\widehat{\Sigma}$ and given the semantics of the empty set by rule 1 defining $\mathfrak{R}$ above. The language $\{\varepsilon\}$ is regular because $\mathfrak{R}((\emptyset)^*) = \emptyset^* = \{\varepsilon\}$; that is, the empty string is an element of L^* for any language L, including the empty language $\emptyset$.

4.4 Exercises

1. Investigate the effect of extending the standard definitions of “string” and “alphabet” to the countably infinite case by determining the cardinalities of the following sets:

   (a) The set of all *infinite* strings over a finite alphabet.

   (b) The set of all finite strings over an *infinite* alphabet.

   (c) The set of all *infinite* strings over an *infinite* alphabet.

   Here “infinite” means “countably infinite” in each case.

2. Write a regular expression that corresponds to each of the following languages, where the alphabet is $\Sigma = \{a, b\}$. You need not give any justification.

   (a) All strings in Σ^* that do not contain two or more contiguous a’s.

   (b) All strings in Σ^* that contain a sequence of four or more b’s.

   (c) $\left\{a^{2n}b^{2m+1} : n \geq 0 \quad \wedge \quad m \geq 0\right\}$