3.2: Regular Expressions

Though we have used the term *string* throughout to refer to a sequence of symbols from an alphabet, an alternative term that is frequently used is *word*. The analogy seems fairly obvious: strings are made up of “letters” from an alphabet, just as words are in human languages like English. In English, however, there are no particular rules specifying which sequences of letters can be used to form legal English words—even unlikely combinations like *ghth* and *ckstr* have their place. While some formal languages may simply be random collections of arbitrary strings, more interesting languages are those where the strings in the language all share some common structure:

$L_{1} = \{x \in \{a, b\}^{*} | n_{a}(x) \}$ ; $L_{2} = \{\text{legal Java identifiers} \}$ ; $L_{3} = \{\text{legal C++ programs} \}$.

In all of these languages, there are structural rules which determine which sequences of symbols are in the language and which aren’t. So despite the terminology of “alphabet” and “word” in formal language theory, the concepts don’t necessarily match “alphabet” and “word” for human languages. A better parallel is to think of the *alphabet* in a formal language as corresponding to the words in a human language; the *words* in a formal language correspond to the *sentences* in a human language, as there are rules (grammar rules) which determine how they can legally be constructed.

One way of describing the grammatical structure of the strings in a language is to use a mathematical formalism called a regular expression. A regular expression is a pattern that “matches” strings that have a particular form. For example, consider the language (over alphabet $\Sigma = \{a, b\}$). $L = \{x \mid \text{x starts and ends with a} \}$ What is the symbol-by-symbol structure of strings in this language? Well, they start with an a, followed by zero or more a’s or b’s or both, followed by an a. The regular expression $a \cdot (a \mid b)^{*} \cdot a$ is a pattern that captures this structure and matches any string in $L$ (· and * have their usual meanings, and | designates or.1) Conversely, consider the regular expression $(a \cdot (a \mid b)^{*}) \mid ((a \mid b)^{*} \cdot a)$. This is a pattern that matches any string that either has the form “a followed by
zero or more a’s or b’s or both” (i.e. any string that starts with an a) or has the form “zero or more a’s or b’s or both followed by an a” (i.e. any string that ends with an a). Thus the regular expression generates the language of all strings that start or end (or both) in an a: this is the set of strings that match the regular expression.

Here are the formal definitions of a regular expression and the language generated by a regular expression:

Definition 3.2.

Let \(\Sigma\) be an alphabet. Then the following patterns are regular expressions over \(\Sigma\):

1. \(\emptyset\) and \(\varepsilon\) are regular expressions;
2. \(a\) is a regular expression, for each \(a \in \Sigma\);
3. if \(r_1\) and \(r_2\) are regular expressions, then so are \(r_1 \cup r_2\), \(r_1 \cdot r_2\), \(r_1^*\) and \((r_1)\) (and of course, \(r_2^*\) and \((r_2)\)). As in concatenation of strings, the \(\cdot\) is often left out of the second expression. (Note: the order of precedence of operators, from lowest to highest, is | , ·, \(\cdot\), \(\ast\)).

No other patterns are regular expressions.

Definition 3.3.

The language generated by a regular expression \(r\), denoted \(L(r)\), is defined as follows:

1. \(L(\emptyset) = \emptyset\), i.e. no strings match \(\emptyset\);
2. \(L(\varepsilon) = \{\varepsilon\}\), i.e. \(\varepsilon\) matches only the empty string;
3. \(L(a) = \{a\}\), i.e. \(a\) matches only the string \(a\);
4. \(L(r_1 \cup r_2) = L(r_1) \cup L(r_2)\), i.e. \(r_1 \cup r_2\) matches strings that match \(r_1\) or \(r_2\) or both;
5. \(L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)\), i.e. \(r_1 \cdot r_2\) matches strings of the form “something that matches \(r_1\) followed by something that matches \(r_2\)”;\)
6. \(L(r_1^*) = (L(r_1))^*\), i.e. \(r_1^*\) matches sequences of 0 or more strings each of which matches \(r_1\) .
7. \(L(r_1^*) = L(r_1)\), i.e. \(r_1^*\) matches exactly those strings matched by \(r_1\).

Example 3.7. Let \(\Sigma = \{a, b\}\) and consider the regular expression \(r = (a^* b^*)^*\). What is \(L(r)\)? Well, \(L(a) = \{a\}\) so \(L(L(a^*)^*) = (L(a)^*)^* = \{a\}^* \) and \(L(\{a\})\) is the set of all strings of zero or more a’s, so \(L(L(a^*)^*) = \{\text{varepsilon, a, a, a, a, a, a, a dots}\}\). Similarly, \(L(b^*) = \{\text{varepsilon, b, b, b, b, b dots}\}\) since \(L(L(a^*) b^*) = L(L(a)^* b^*) = L(L(a)^*) L(L(b^*) = (x y \mid x \in \Sigma) L(L(a)^*) L(L(b^*)\) wedge \(y \in \Sigma L(L(b^*)\) )\) we have \(L(L(a^*) b^*)^* = \{\text{varepsilon, a, a, a, a, a, a, a, a, a, a, a, a dots}\}\), which is the set of all strings of the form “zero or more a’s followed by zero or more b’s”.

https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_Foundations_of_Computation_(Critchlow_and_Eck)/03
Example 3.8. Let $\Sigma = \{a, b\}$ and consider the regular expression $r = (a|a a| a a a)(b b)^*$. Since $L(a) = \{a\}$, $L(a a) = L(a) * L(a) = \{a a\}$, and $L(a a a) = \{a a a\}$, and $L(b b) = \{b b\}$. Now $L(a|a a| a a a) = L(a) \cup L(a a) \cup L(a a a) = \{a, a a, a a a\}$, and $L((b b)^*) = (L((b b)))^* = (L(b b))^* = \{b b\}^* = \{\varepsilon, b b, b b b b, \ldots\}$. So $L(r)$ is the set of strings formed by concatenating $a$ or $a a$ or $a a a$ with zero or more pairs of $b$.

Definition 3.4.

A language is regular if it is generated by a regular expression.

Clearly the union of two regular languages is regular; likewise, the concatenation of regular languages is regular; and the Kleene closure of a regular language is regular. It is less clear whether the intersection of regular languages is always regular; nor is it clear whether the complement of a regular language is guaranteed to be regular. These are questions that will be taken up in Section 3.6.

Regular languages, then, are languages whose strings’ structure can be described in a very formal, mathematical way. The fact that a language can be “mechanically” described or generated means that we are likely to be able to get a computer to recognize strings in that language. We will pursue the question of mechanical language recognition in Section 3.4, and subsequently will see that our first attempt to model mechanical language recognition does in fact produce a family of “machines” that recognize exactly the regular languages. But first, in the next section, we will look at some practical applications of regular expressions.

Exercises

1. Give English-language descriptions of the languages generated by the following regular expressions.
   a) $(a | b)^*$
   b) $a^* | b^*$
   c) $b^* (a b^* a b^*)^*$
   d) $b^* (a b^*)$

2. Give regular expressions over $\Sigma = \{a, b\}$ that generate the following languages.
   a) $\{|x | x \text{ contains } 3 \text{ consecutive } a\text{'s}\}^*$
   b) $\{|x | x \text{ has even length}\}^*$
   c) $\{|x | n_{b}(x) = 2 \bmod 3\}^*$
   d) $\{|x | x \text{ contains the substring aaba}\}^*$
   e) $\{|x | n_{b}(x) < 2\}^*$
   f) $\{|x | x \text{ doesn't end in aa}\}^*$

3. Prove that all finite languages are regular.