

Regular Expressions

Often you are faced with the task of recognizing strings of a certain type. For example, suppose you want a method or procedure that recognizes decimal numbers. To write such a method, we need to be clear about what we mean by a decimal number. We can define a decimal number as follows: "Some digits followed maybe by a point and some more digits."

That definition is far too sloppy to be of use. The following is better: "optional minus sign, any sequence of digits, followed by optional point, and if so then optional sequence of digits." To be more precise, we can write the description of a decimal number as a regular expression:

$$(- + \epsilon) D D^* (\epsilon + . D^*)$$

where D stands for a digit. But then you need to understand what a regular expression is

2.1 Regular Expressions

A regular expression (RE) corresponds to a set of strings; that is, a regular expression describes a language. It is built up using the three regular operations called **union**, **concatenation**, and **star**, described in the following paragraphs. A regular expression uses normal symbols as well as four special symbols:

+ * ()

A regular expression is interpreted using the following rules:

- The parentheses (and) are used for grouping, just as in normal math.
- The plus sign (+) means **union**. Thus, writing

$$0 + 1$$

means either a zero or a one. We also refer to the + as **or**.

- The **concatenation** of two expressions is obtained by simply writing one after the other without spacing between them. For example,

$$(0 + 1)0$$

corresponds to the pair of strings 00 and 10.

- The symbol ϵ stands for the empty string. Thus, the regular expression

$$(0 + 1)(0 + \epsilon)$$

corresponds to the four strings 00, 0, 10, 1.

- The asterisk (*) is pronounced *star* and means zero or more copies.

For example, the regular expression

$$a^*$$

corresponds to any string of a's: $\{\epsilon, a, aa, aaa, \dots\}$. Also,

$$(0 + 1)^*$$

corresponds to all binary strings. (It's $\epsilon + (0 + 1) + (0 + 1)(0 + 1) + \dots$)

Example 2.1

The regular expression

$$(01)^*$$

corresponds to the set containing $\epsilon, 01, 0101, 010101$, and so forth.

Example 2.2

What about a regular expression for the language of all binary strings of length at least 2 that begin and end in the same symbol?

This is given by:

$$0(0 + 1)^*0 + 1(0 + 1)^*1$$

Note too the **precedence** of the regular operators: star first, then concatenation, then or. That is, the *star* always refers to the smallest piece it can, the *or* to the largest.

Example 2.3

Consider the regular expression

$$((0 + 1)^*1 + \epsilon)(00)^*00$$

The language of this RE is the collection of all binary strings that end with an even nonzero number of 0's.

There are snares waiting for the unwary. For example, someone proposed the RE

$$(0 + 1)^*(00)^*00$$

for this language. But this RE corresponds to any binary string that ends with 00: take any string you like from the $(0 + 1)^*$ part, take nothing from the $(00)^*$ part, and 00 from the last part.

In general, if you form an RE by the **or** of two REs, call them R and S , then the resulting language is the union of the languages of R and S . This is why + is called the union operation.

If you form an RE by the **concatenation** of two REs, call them R and S , then the resulting language consists of all strings that can be formed by taking one string from the language of R and one string from the language of S and concatenating them.

If you form an RE by taking the star of an RE R , then the resulting language consists of all strings that can be formed by taking any number of strings from the language of R (they need not be the same and they need not be different), and concatenating them.

Indeed, we can talk about the union, concatenation, or star of languages. For example, here is a recursive definition of the star of a language:

Recursive definition of L^*

1. $\epsilon \in L^*$
2. If $x \in L^*$ and $y \in L$ then $xy \in L^*$

Example 2.4

If language L is $\{ma, pa\}$ and language M is $\{be, bop\}$, then

$$L + M \text{ is } \{ma, pa, be, bop\}.$$

$$LM \text{ is } \{mabe, mabop, pabe, pabop\}.$$

$$L^* \text{ is } \{\epsilon, ma, pa, mama, \dots, pamamapa, \dots\}.$$

We also use the following notation.

Notation If Σ is some alphabet, then Σ^* is the set of all strings using that alphabet.

2.2 Kleene's Theorem

It is not hard to write a recognizer to accept a string if and only if the string is of the required form described by the preceding example regular expressions. But what about a complicated regular expression? Fortunately, there is an algorithm (and hence a program) for doing this. In fact, you can always build a finite automaton.

Example 2.5

Give an FA for the language of the RE

$$(0+1)^*00(0+1)^*$$

This was given in Example 1.1.

In the next chapter we will prove the following:

Kleene's Theorem There is an FA for a language if and only if there is an RE for the language.

This theorem is not just of theoretical benefit: for, it is often easy to describe a recognition problem by an RE, and it is easy to write a procedure that simulates an FA. The theorem, translated into an algorithm, gives one the bridge to convert an RE into an FA. We will use the term **regular language** for a language that is accepted by some FA or described by an RE.

2.3 Applications of REs

We mention here a few of the applications of finite automata and regular expressions. Several more are discussed in Chapter 5.

Regular expressions are used in the design of compilers to describe some of the pieces of the language. For example, in Pascal or Java an integer has a certain form, a real number has a certain form, and so forth. The RE for

each piece can be automatically converted into an FA that recognizes them. A scanner or **tokenizer** is a program that scans the input and determines and identifies the next piece. A tokenizer is provided as a standard part of Java and Unix.

Regular expressions can also be used for searching a file. For example, suppose in a long document you've written hexadecimal numbers with an *h* behind, such as 273h or 22h, and your boss now says that hexadecimal numbers should be written with a # in front, such as #273 or #22. Well, if you have a good search-and-replace option (such as in Perl), you can type something like

```
find: ([0123456789]+)h
replace with: #\1
```

In the find text, the symbol + is like * and stands for one or more copies; the square brackets specify a set of symbols; and the parentheses specify an expression. The \1 in the replacement text means the first expression in the find text.

↑ For You to Do!

Give an RE for each of the following three languages:

1. All binary strings with at least one 0
2. All binary strings with at most one 0
3. All binary strings starting and ending with 0

EXERCISES

2.1 For each RE, state which of the following strings is in the language of the RE: ϵ , abba, bababb, and baaaa.

- a) $(a+b)^*ab(a+b)^*$
- b) $b^*ab^*ab^*$
- c) $a+(a^*b)^*$

2.2 For each RE, give two strings that are in the corresponding language and two strings that are not:

- a) $a(a+b)^*b$
- b) $a^*a+\epsilon+b^*$
- c) $(ab+ba)^*$

- 2.3 Give REs for:
- All binary strings with exactly two 1's
 - All binary strings with a double symbol (contains 00 or 11) somewhere
 - All binary strings that contain both 00 and 11 as substrings
 - All binary strings without a double symbol anywhere
- ★ 2.4 Give an FA for decimal numbers as described at the start of this chapter.
- 2.5 Give an RE for the language of all binary strings of length at least two that begin and end with the same symbol.
- 2.6 Give REs and FAs with alphabet $\{a, b\}$ for
- All strings containing the substring aaa
 - All strings not containing the substring aaa
 - All strings that do not end with aaa
 - All strings with exactly 3 a's.
 - All strings with the number of a's divisible by 3.
- 2.7 Give an RE for the language of Exercise 1.12.
- ★ 2.8 Give an RE for the language of Exercise 1.13.
- 2.9 Give an RE for the language of Exercise 1.14.
- 2.10 Give an RE for the language of Exercise 1.15.
- 2.11 Give an RE for the language of Exercise 1.16.
- ★ 2.12 Give an RE for the language of Exercise 1.19.
- 2.13 Give a regular expression for the complement of the RE 111 with respect to the alphabet $\{0, 1\}$. That is, the RE should allow every possible binary string except for the string 111.
- 2.14 Does every regular language correspond to only *one* RE? Does every RE correspond to only *one* regular language? Discuss.
- 2.15 Show that the language of RE $(0^*1^*)^*$ is all binary strings.
- ★ 2.16 Simplify the following REs (that is, find a simpler RE for the same language):
- $(r + \epsilon)^*$
 - $ss^* + \epsilon$
 - $(\epsilon + r)(r + s)^*(\epsilon + r)$
 - $(r + s + rs + sr)^*$
- 2.17 In a string, a **block** is a substring in which all symbols are the same and which cannot be enlarged. For example, 0001100 has *three* blocks. Let L be the language consisting of all binary strings such that every block has length 2 or 3. Give both an FA and an RE for L .
- Ⓜ 2.18 Let C_n denote the set of all binary numbers that are a multiple of n . Show that C_7 is regular. (Hint: Find an FA.)

- 2.19 Propose a formal definition of a regular expression in the spirit of the one given for an FA on Page 9.

"For You to Do" Exercise Solutions

- $(0+1)^*0(0+1)^*$
- $1^*+1^*01^*$
- $0(0+1)^*0+0$

In each case several answers are possible.