**REGULAR EXPRESSIONS and REGULAR LANGUAGES**

***DEFINITION:*** *Regular expressions* over an alphabet  are the strings over the alphabet:  ∪ { ), (, Ø, ∪, °, \*}such that the following hold.

1.  is a regular expression

1. Ø is a regular expression.

2. Each element a ∈  is a regular expression.

3. If  and  are regular expressions then so is their concatenation (°)

4. If  and  are regular expressions then so is their union (∪ )

5. If  is a regular expression then so is its Kleene star \*

6. Nothing is a regular expression unless it follows from (1) through (5) above.

Regular expressions represent a language called *Regular Language*.

NOTE 1: If  = {a, b} then the alphabet for the strings forming the regular expressions is: {a, b, ), (, ∪, Ø, °, \*}

NOTE2: Many authors write (+ ) instead of (∪ ). Since writing (+ ) has no meaning in itself there should be no confusion but only brings about perhaps a better notation for the understanding of Union as an OR logical operation.

Formally, the relation between regular expressions and the languages they represent is established by a function L, such that if  is any regular expression, then the notation L() is the language represented by .

That is, L is a function from strings to languages. The function L is formally defined later in this chapter.

**NOTATION SUMMARY/EXAMPLES:**

|  |  |  |
| --- | --- | --- |
|  | REGULAR LANGUAGE | REGULAR EXPRESSION |
| 1 | {** } | ** |
| 2 | {0} | 0 |
| 3 | {001} = {0}{0}{1} | 001 |
| 4 | {0, 1} = {0} ∪ {1} | 0 + 1 |
| 5 | {0, 01} = {0} ∪ {01} | 0 + 01 |
| 6 | {1, **}{001} | (1 + **)001 |
| 7 | {110}\*{0, 1} | (110)\*(0 + 1) |
| 8 | {0, 10}\*({11}\*∪ {001, ** })\* | (0 + 10)\*((11)\* + 001 +**)\* |
| 9 | {1}\*{10} | 1\*10 |
| 10 | {10, 11, 11010}\* | (10 + 11 + 11010)\* |

**OTHER EXAMPLES (Notation)**

1. Notice that 01 ∪ 10 = (01 + 10) = {01, 10}
2. Notice that (0 ∪ )1\* = 01\* ∪ 1\* = 01\* + 1\*
3. Notice that (0 ∪ ) (1 ∪ ) = {, 0, 1, 01} = ( + 0 + 1 + 01)
4. Notice that 1\*
5. Notice that \* = {  }
6. Notice that {∪ 0 = + 0

**Example:**

Expand the expression {10, 11}\*

{10, 11}\* = {, 10, 11, 1010, 1011, 1110, 1111, 101010, …}

Clearly in this case the alphabet is: = {0, 1}.

In writing regular expressions we can omit many parentheses if we assume that \* has higher precedence than concatenation or +, and that concatenation has higher precedence than +.

For example, ((0(1\*)) + 0) may be written 01\* + 0. We may also abbreviate the expression rr\* by r+.

When necessary to distinguish between a regular expression r and the language denoted by r, we use L(r) for the latter. When no confusion is possible we use r for both the regular expression and the language denoted by the regular expression.

**More Examples.**

00 is a regular expression representing the language {00}.

The expression (0 + 1)\* denotes *all* strings of 0's and l's. In fact, this is the universe for all languages derived from the alphabet {0, 1}.

Thus, (0 + l)\*00(0 + 1)\* denotes all strings of 0's and l's with at least two consecutive 0's.

The regular expression (1 + 10)+ denotes all strings of 0's and l's beginning with 1 and not having two consecutive 0's. In proof, it is an easy induction on i that (1 + 10)i does not have two consecutive 0's. Furthermore, given any string beginning with 1 and not having consecutive 0's, one can partition the string into l's, with a following 0 if there is one.

For example, 1101011 is partitioned 1-10-10-1-1. This partition shows that any such string is in (1 + 10)i, where i is the number of l's.

The regular expression (0 + )(l + 10)\* denotes all strings of 0's and l's whatsoever that do not have two consecutive 0's.

For some additional examples, (0 + l)\*011 denotes all strings of 0's and l's ending in 011.

Also, 0\*1\*2\* denotes any number of 0's followed by any number of l's followed by any number of 2's.

The language 00\*11\*22\* denotes those strings in 0\*1\*2\* with at least one of each symbol. We may use the shorthand 0+l+2+ for 00\*11\*22\*.

**Formal Definition of a Regular Language.**

1. Ø is a regular expression and the language is L(Ø) = Ø.
2. For each a ∈, the language is L(a) = {a}
3. If  and  are regular expressions then L(()) = L()L()
4. If  and  are regular expressions then L((∪ )) = L() ∪ L()
5. If  is a regular expression then L(\*) = (L() )\* = L()\*
6. Nothing is a regular unless defined by (1) thru (5) above.

**Note**

Statement 2 defines L() for each regular expression  that consists of a single symbol; if n > 1, then (3) through (5) define L() for regular expressions of length *n* in terms of L() for one or two regular expressions  of length n - 1 or less.

Thus, every *regular expression* is associated in this way with some *regular language*.

**Example**

What language is represented by (((a ∪ b)\*a))? (Note: It may be illuminating to consider union as a “sum”). We have the following:

L(((a ∪ b)\*a)) = L((a ∪ b)\*) L(a) by (3)

= L((a ∪ b)\*) {a} by (2)

= L((a ∪ b))\* {a} by (5)

= (L(a) ∪ L(b) )\* {a} by (4)

= ({a} ∪ {b} )\* {a} by (2) twice

= {a, b}\* {a}

= {w ∈ {a, b}\* : w ends with a}

**Example**

The expression (0\* ∪ (((0\*(1 ∪ (11)))((00\*)(1 ∪ (11)))\*)0\*)) represents the set of all strings over {0, 1} that do not include the substring 111.

**NOTE:**

To see it more clearly let’s look at it step by step:

(0\* ∪ (((0\*(1 ∪ (11))) ((00\*) (1 ∪ (11)))\* )0\*))

0\* ∪ (((0\*(1 ∪ (11))) ((00\*) (1 ∪ (11)))\* )0\*) now “for clarity” we switch to the “+” sign instead of the union sign “∪”:

0\* + ((0\*( 1 + 11 )) ((00\*)( 1 + 11 ))\*)0\*

0\* + ( 0\*( 1 + 11 ) ( 00\* ( 1 + 11 ))\*)0\*

0\* + 0\*( 1 + 11 ) ( 00\* ( 1 + 11 ))\*0\*

0\* + (0\*1 + 0\*11) (00\*1 + 00\*11 )\* 0\*

**Example**

What language is represented by (c\*(a ∪ (bc\*))\*) = c\*{a, 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 *L((c\*(a ∪ (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 substring *ac* and not beginning with c.

Any such string is in *L((a ∪ (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∈ L((c\*(a ∪ (bc\*))\*)).*

Remarks

• Every language that can be represented by a regular expression can be represented by infinitely many of them. For example,

 and ( ∪ Ø) always represent the same language;

so do (( ∪ ) ∪ ) and ( ∪ ( ∪ )); and

so do (())) and (()).

Since set union and concatenation are associative operations.

• We normally omit the extra ( and ) symbols in regular expressions; for example, we treat ( ∪  ∪ )\*as a regular expression even though "officially" it is not.

For another example, the regular expression which represents the set of all strings over {0, 1} that do not have the substring 111,

(0\* ∪ (((0\*(1 ∪ (11)))((00\*)(1 ∪(11)))\*)0\*))

might be rewritten as 0\* ∪ 0\*(1 ∪ 11)(00\* (1 ∪ 11))\*0\*.

• Thus we may say at one point that a\*b\* is the set of all strings consisting of some number of *a's* (0 or more a’s) followed by some number of b's (0 or more b’s) to be precise, we should have written {a}\*{b}\*.

**NOTE:** the set a\*b\* is not the same as (ab)\*. (To prove expand the \*)

• 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\*).

So, these regular expressions are one method for describing concisely certain infinite languages, the regular languages.

**DEFINITION OF REGULAR LANGUAGES**

The class of regular languages (or regular sets) over an alphabet  is the minimal set of languages containing Ø and the singleton sets {a}, for all a ∈ and is closed under the operations of *union*, *concatenation*, and *Kleene star*.

That is, the class *R* of regular languages is to have the following four properties.

1. Ø ∈ *R*;

2. If a ∈, then {a} ∈ *R*

3. If *A* and *B ∈R*, then A∪ *B*, *A°B* and *A\** (*or B\**) are members of *R*

4. If *S* is the set of languages containing *Ø* and all languages *{a},* with all *a ∈,* (as in statement 2) and closed under union, concatenation, and Kleene star (as in statement 3), then we have that: *R ⊆ S*.

Remarks

• A language is regular if and only if (iff) can be described by a regular expression.

• Unfortunately, we cannot describe by regular expressions some languages that have very simple descriptions by other means.

For example, {0n1n: n ≥ 1} is not a regular language. Any theory of the finite representation of languages will have to accommodate at least such simple languages as this.

• Thus, in general, regular expressions are an inadequate specification method.

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

L = {w ∈ \*: w has property P}.

The trouble is that, unless we somehow restrict the kinds of attributes that can be used in describing the property P, we may end up with some bizarre representations of languages.

We might try restricting P to ''well-defined mathematical properties" according to some suitable definitions of these terms.

However, from a computational standpoint, some descriptions may have a discomforting character: they do not seem to help us determine, given a string *w*, whether or not it belongs to *L*.

So, for computational purposes, a property *P* can be considered to be a suitable description of the strings in a language only if we are also given a systematic procedure for deciding given a string *w*, whether or not it has property *P*.

Thus, as a first step, we should concentrate on formalizing the notion of a *"systematic procedure"* or *algorithm*.

**Definition of Algorithm**

An algorithm may be described as a finite sequence of instructions,

precisely expressed, that when confronted with a question of some kind and carried out in the most literal-minded way, will invariably terminate, sooner or later, with the correct answer.

For example, the method taught in elementary school for multiplying two integers can be presented as an algorithm.

It is designed for answering n particular kind of question, "What

is the product of *m* and *n* " If written down in very dry and explicit language it could be carried out mechanically.

Finally, we know that, if carried out correctly, it will eventually give the right answer.

In general, we shall not insist that algorithms be written out in tedious detail; it turns out rarely to be controversial what can and cannot be turned into a fully explicit set of instructions.

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 ∈ {a, b}\*: w does not have the substring bbb}

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 an *a* is encountered in the input; add one every time a *b* 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”.

On the other hand, a regular expression such as

( ∪ b ∪ bb)(a ∪ ab ∪ abb)\*

may be viewed as a way of *generating members of that 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?).

The relation between language recognition devices and language generators, both of which are types of finite language specifications, is a major subject of the Theory of Computation.

The theory of computation begins with a question: What is a computer?

It is perhaps a silly question, as even a four-year-old knows that this thing we type on is a computer.

But these real computers are quite complicated, too much so to allow us to set up a manageable mathematical theory of them directly.

Instead we use an idealized computer called a computational model.

As with any model in science, a computational model may be accurate in some ways but perhaps not in others.

Thus, we will use several different computational models, depending on the features we want to focus on.

Soon we will study the simplest model, called the *finite state machine* (FSA) or *finite automaton* (FA).

**EXAMPLES (taken mostly from Sipser’s textbook):**

Given the alphabet  = {0, 1}\*,write a regular expression for the following regular languages:

L = {w | w has exactly a single 1}

Answer: 0\*10\*

L = {w | w has at least one 1}

Answer: \*1\*

L = {w | w contains the string 001 as a substring}

Answer: \*001\*

L = {w | w is a string of even length}

Answer: (\*

L = {w | w is a string of odd length}

Answer: (\*

L = {w | the length of w is a multiple of three}

Answer: (\*

L = {w | w starts and ends with the same symbol}

Answer: 0\*0 U 1\*1 U 0 U 1

L = {w | w ends in 0 or 11}

Answer: \*(0 U 11)

L = {w | w has an even number of 0’s followed by an odd number of 1’s}

Answer: (00)\*(11)\*1

L = {w | w has at least one pair of consecutive zeroes}

Answer: \*00\*

L = {w | w has no pair of consecutive zeroes}

Answer: (1\*011)\*(0 + ) + 1\*(0 + )

Another answer: (1 + 01)\*(0 + )

GENERALLY THERE ARE AN UNLIMITED NUMBER OF REGULAR EXPRESSIONS FOR ANY REGULAR LANGUAGE.

L = {w | w contains the substring 0101}

Answer: \*0101\*

L = {w | w has length at least 3 and its third symbol is a 0}

Answer: \*

L = {w | w is a string with an even number of 0’s}

Answer: 1\*(01\*01\*)\*

Harder exercises are the following:

L = {w | w does not contain the substring 110}

Answer: (0 U (10)\*)\*1\*

L = { w | w does not have the substring 111}

Answer: ( ∪ 1 ∪ 11)(0 ∪ 01 ∪ 011)\*