REGULAR EXPRESSIONS AND FINITE-STATE AUTOMATA

1)

- Let the alphabet ∑ = {a, b}.
- a. Define a language L1 over ∑ to be the set of all strings that begin with the character **a** and have length at most three characters. Find L1.
- b. A **palindrome** is a string that looks the same if the order of its characters is reversed. For instance, aba and baab are palindromes. Define a language L2 over to be the set of all palindromes obtained using the characters of . Write ten elements of L2.

Solution

- a. L1 = {a, aa, ab, aaa, aab, aba, abb}
- b. L2 contains the following ten strings (among infinitely many others):

ϵ, a, b, aa, bb, aaa, bab, abba, babaabab, abaabbbbbaaba

2)

- Let ∑ be an alphabet. For each nonnegative integer n, let
- ∑ n = the set of all strings over ∑ that have length n,
- ∑ + = the set of all strings over ∑ that have length at least 1, and
- ∑ ∗ = the set of all strings over ∑.

- Let ∑ = {a, b}.
- a. Find ∑ 0, ∑ 1, ∑ 2, and ∑ 3.
- b. Let A = ∑ 0 ∪ ∑ 1 and B = ∑ 2 ∪ ∑ 3. Use words to describe A, B, and A ∪ B.
- c. Describe a systematic way of writing the elements of ∑ +. What change needs to be made to obtain the elements of ∑ ∗?

Solution

- a. ∑0 = {ϵ }, ∑ 1 = {a, b}, ∑ 2 = {aa, ab, ba, bb}, and ∑ 3 = {aaa, aab, aba, abb, baa, bab, bba, bbb}
- b. A is the set of all strings over ∑ of length at most 1.

B is the set of all strings over ∑ of length 2 or 3.

A ∪ B is the set of all strings over ∑ of length at most 3.

c. Elements of ∑ + can be written systematically by writing all the strings of length 1, then all the strings of length 2, and so forth.

∑ +: a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa,...

Of course the process of writing the strings in ∑ + would continue forever, because ∑ + is an infinite set. The only change that needs to be made to obtain ∑ ∗ is to place the null string at the beginning of the list. ■

Activate
Go to Setti

3)

- In 1 and 2 let ∑ = {x, y} be an alphabet.

1. a. Let L1 be the language consisting of all strings over ∑ that are palindromes and have length ≤ 4. List the elements of L1 between braces.

b. Let L2 be the language consisting of all strings over ∑ that begin with an x and have length ≤ 3. List the elements of L2.

a)
$$L_1 = \{x, y, xx, yy, xxx, yyy, xyx, yxy, xxxx, yyyy, xyyx, yxxy\}$$

b)
$$L_2 = \{x, xx, xy, xxx, xxy, xyx, xyy\}$$

4)

b. List between braces the elements of ∑ 4, the set of strings of length 4 over ∑.

$$\Sigma^4 = \{xxxx, xxxy, xxyx, xxyy, xyxx, xyxy, xyyx, xyyy, yxxx, yxxy, yxyx, yxyy, yyxx, yyxy, yyyx, yyyy\}$$

Polish Notation

(a + b)_ infix notation_ operator such as + sits between the two quantities.
(+ab)_ **prefix notation**_binary operator precedes the quantities on which it acts
(ab+)_ **postfix notation**_ binary operator follows the quantities on which it acts.

A great advantage of these notations is that they eliminate the need for parentheses in writing arithmetic expressions.

Ex: *For instance, in postfix (or reverse Polish) notation, the expression 8 4 + 6 / is evaluated from left to right(Same as (8+4)/6).*

Infix: **(a + b)·c** Postfix: **ab + c ·**

- a. If the expression ab·cd ·+ in postfix notation is converted to infix notation, what is the result?
- b. Let ∑ = {4, 1, +, −}, and let L = the set of all strings over ∑ obtained by writing either a 4 or a 1 first, then either a 4 or a 1, and finally either a + or a −. List all elements of L between braces, and evaluate the resulting expressions

$$L = \{44+, 44-, 41+, 41-, 14+, 14-, 11+, 11-\}$$

Evaluation: 4+4=8, 4-4=0, etc.

LANGUAGES CAN BE COMBINED TO FORM NEW LANGUAGES

Let ∑ be an alphabet. Given any strings x and y over ∑.

The **concatenation** of L and L ' , denoted L L ' , is: **L L ' = {x y | x ∈ L and y ∈ L ' }**

The **union** of L and L ' , denoted L ∪ L ' , is: **L ∪ L ' = {x | x ∈ L or x ∈ L ' }**

The **Kleene closure** of L, denoted L∗, is:

L∗ = {x | x is a concatenation of any finite number of strings in L}

Note **that ϵ is in L∗** because it is regarded as a concatenation of zero strings in L.

1)

- Let L1 be the set of all strings consisting of an even number of a's (namely, ε, aa, aaaa, aaaaaa, . . .), and let L2 = {b, bb, bbb}.

Find L1L2, L1 ∪ L2, and (L1 ∪ L2)∗. Note that the null string ϵ is in L1 because 0 is an even number.

I)L1L2 = the set of all strings that consist of an even number of a's followed by b or by bb or by bbb.

II)L1 ∪ L2 = the set that includes the strings b, bb, bbb and any strings consisting of an even number of a's.

III)(L1 ∪ L2)∗ = the set of all strings of a's and b's in which every occurrence of a is in a block consisting of an even number of a's.

The Language Defined by a Regular Expression

The regular expression r ∗ is called the Kleene closure of r

- Given an alphabet ∑, the following **are regular expressions over ∑ :**
- **I. BASE:** $\emptyset, \epsilon$, and each individual symbol in ∑ are regular expressions over ∑.
- **II. RECURSION** : If r and s are regular expressions over ∑, then the following are also regular expressions over ∑ :

(i) (r s) (ii) (r |s) (iii) (r ∗)

If the alphabet ∑ happens to include symbols—such as (|) ∗—special provisions have to be made to avoid ambiguity. An escape character, usually a **backslash**, is added before the potentially ambiguous symbol

For instance, a left parenthesis would be written as \(and the backslash itself would be written as \\.

To eliminate parentheses, **an order of precedence** for the operations used to define regular expressions has been introduced.

The **highest is** ∗, concatenation is next, and | is the lowest.

It is also customary to **eliminate the outer set of parentheses** in a regular expression, because doing so does not produce ambiguity.

(a((bc) ∗)) = a(bc) ∗ and (a |(bc)) = a | bc

- a. Add parentheses to make the order of precedence clear in the following expression: ab∗ | b∗a.
- b. Use the convention about order of precedence to eliminate the parentheses in the following expression: ((a |((b∗)c))(a∗)).

Solution

- a. ((a(b∗))|((b∗)a))
- b. (a | b∗c)a∗

Language defined by regular expression

- For any finite alphabet ∑, the function L that associates a language to each regular expression over ∑ is defined by (I) and (II) below. For each such regular expression r, L(r) is called the **language defined by r.**
- I. **BASE**: L(∅) = ∅, L(ϵ) = {ϵ}, L(a) = {a} for every a in ∑.
- II. **RECURSION**: If L(r) and L(r') are the languages defined by the regular expressions r and r ' over ∑, then
- (i) L(rr ') = L(r)L(r ') (ii) L(r |r ') = L(r) ∪ L(r ') (iii) L(r ∗) = (L(r))∗

Example: Let ∑ = {a, b}, and consider the language defined by the regular expression **(a | b)**∗. Use set notation to find this language, and describe it in words.
L((a | b)∗)=L(a|b)*=L({a}∪{b})*={a,b}*=∑*= set of all strings that start by characters a or b.

Note that **concatenating strings** and **taking unions** of sets are both associative operations. Thus for any regular expressions r,s and t,

L((r s)t) = L(r(st)), Since **L((r|s)|t)=L(r|(s|t))**

The expression a∗ matches only strings consisting entirely of the character aaa, including the empty string (e.g., "", "a", "aa", "aaa", etc.)

- In each of (a) and (b), let∑= {a, b} and consider the language L over ∑ defined by the given regular expression.
- a. The regular expression is a∗b(a | b)∗. Write five strings that belong to L.
- b. The regular expression is a∗ |(ab)∗. Indicate which of the following strings belong to L:

a b aaaa abba abababb

Solution

a. The strings b, ab, abbb, abaaa, and ababba are five strings from the infinitely many in L.

b. b. The following strings are the only ones listed that belong to L: a, aaaa, and abababb. The string b does not belong to L because it is neither a string of a's nor a string of possibly repeated ab's. The string abba does not belong to L because any two b's that might occur in a string of L are separated by an a

---

- Let ∑ = {0, 1}. Use words to describe the languages defined by the following regular expressions over ∑ .
- a. 0∗1∗ | 1∗0∗
- b. 0(0 | 1)∗

a. The strings in this language consist either of a string of 0's followed by a string of 1's or of a string of 1's followed by a string of 0's. However, in either case the strings could be empty, which means that ε is also in the language.

b. The strings in this language have to start with a 0. The 0 may be followed by any finite number (including zero) of 0's and 1's in any order. Thus the language is the set of all strings of 0's and 1's that start with a 0. ■

Example: Use set notation to derive the language defined by the given regular expression.
Assume ∑ = {a, b, c}
ϵ | ab ___ L= {ϵ,ab}
∅ | ___ L = ∅
(a | b)c ___ L={ac,bc}

A Regular Expression That Defines a Language

This chapter is just full of exercises.

Practical Uses of Regular Expressions

- **Character classes** are also allowed to include more than one range of characters. For instance, **[A − C x − z]** stands for **(A | B | C | x | y | z)**.
- Other commonly used shorthands are **[ABC]** to stand for **(A | B | C)**.
- If ∑ **= {A, B,C}**, then A.C stands for **(AAC | ABC | ACC)**.
- **[ˆ D − Z][0 − 9][0 − 9]** ∗_ stands for any string starting with a letter of the alphabet different from D to Z, followed by any positive number of digits from 0 to 9. Examples are B3097,C0046, and so forth.
- If r is a regular expression, the notation **r+ denotes** the concatenation of r with itself any positive finite number of times:
    - **r+ = rr** ∗
- . If r is a regular expression, then **r? = (ϵ |r),** either zero occurrences or exactly one occurrence of r.
- Following are true when **m ≤ n**, **m > 0**, **n > 0**.
    - **r{n}** denotes the concatenation of r with itself exactly n times.
    - **r{m, n}** denotes the concatenation of r with itself anywhere from **m through n times**.

  Examples:

  [0 − 9][0 − 9][0 − 9] - [0 − 9][0 − 9][0 − 9][0 − 9] = **[0-9]{3} - [0-9]{4}**

  Rest of this chapter is **full of exercises**.