Finite Automata

- Deterministic Finite Automata
- Languages decided by a DFA – Regular Languages
- Closure Properties of regular languages
- Non-Deterministic Finite Automata, DFA= NFA
- Regular Expression: Computation as Description
- DFA=NFA=RegExp, Generalized NFA
- Non-Regular Languages, The Pumping Lemma
- Minimizing DFA

Imdad ullah Khan

Regular Expression

Regular Expression

Regular Expression is a way to describe DFA-computable problems

Recall Computational Problem = Language Recognition

Regular expression enables simple algebraic description rather than a machine representation of computation (at least for a large class of computation problems, namely regular languages)

What is the complexity of describing the strings in the language?

Regular expressions describe regular languages or problems computable by DFA (that is why DFA-recognizable languages are called regular languages)

Regular Expression (Syntax)

Inductive Definition of Regular Expression

A regular expression (recursively) describes a language by combining simple languages using the regular operations

Let Σ be an alphabet. Regular expressions (regexp) over Σ are defined as

- For all $\sigma \in \Sigma$, σ is a regexp
- ϵ is a regexp
- $\emptyset$ is a regexp
- If R_1 and R_2 are regexps, then
  - $(R_1 \circ R_2)$ is a regexp ▷ concatenation, also denoted as $(R_1 R_2)$
  - $(R_1 \cup R_2)$ is a regexp ▷ union, also denoted as $(R_1 + R_2)$
  - $(R_1)^*$ is a regexp ▷ star

Examples: ϵ, 1, $\emptyset$, $(0+1) \circ 0$, $(1+0)^*$, $((0)^* \circ 1) + (10)$

Regular Expression (Semantics)

Meaning of Regular Expressions

Regular expressions describe languages

- For $\sigma \in \Sigma$, the regexp σ represents the language $\{\sigma\}$
- the regexp ϵ represents the language $\{\epsilon\}$
- the regexp $\emptyset$ represents the language $\emptyset$
- If R_1 and R_2 are regexps representing languages L_1 and L_2 then
  - $(R_1 \circ R_2) = (R_1 R_2)$ represents the language $L_1 \circ L_2$
  - $(R_1 + R_2) = (R_1 \cup R_2)$ represents the language $L_1 \cup L_2$
  - $(R_1)^*$ represents the language L_1^*

Examples: $(111) + (01 + 1^*0)$ represents the language $\{111, 01, 0, 10, 110, 1110, 11110, \ldots\}$

Language described by Regexp

Let $L(R)$ be the language described/represented by regexp R

A string $w \in \Sigma^*$ matches R (is recognized/accepted by R) if $w \in L(R)$

$111, 01, 0, 10, 110, 1110$ all match $(111) + (01 + 1^*0)$

$1111, 00, 011$ do not match $(111) + (01 + 1^*0)$

$0, 11000, 101010, 100, 1111110$ all match $(1 + 0)^*0$

$01101, 1010101, 1001, 11111101$ do not match $(1 + 0)^*0$

We omit parenthesis and use ops. precedence $*$ then $\circ$ then $+$

$$(R_1(R_2)^*) + R_3 = R_1 R_2^* + R_3$$

Language described by Regexp

Let $L(R)$ be the language represented by regexp R $(\Sigma = \{a, b\})$

- $L(R) = \{w : w \text{ has exactly one } b\}$
  $$R = a^*ba^*$$
- $L(R) = \{w : w \text{ starts with } bb\}$
  $$R = bb(a+b)^*$$
- $L(R) = \{w : w \text{ ends with } ab\}$
  $$R = (a+b)^*ab$$
- $L(R) = \{w : w \text{ contains } baa\}$
  $$R = (a+b)^*baa(a+b)^*$$
- $L(R) = \{w : w \text{ has } b \text{ at every odd position}\}$
  $$R = (b(a+b))^*(b+\epsilon)$$
- $L(R) = \{w : w \text{ has length} \geq 3 \text{ and third symbol is } b\}$
  $$R = (a+b)(a+b)b(a+b)^*$$

Language described by Regexp

Let $L(R)$ be the language represented by regexp R $\quad (\Sigma = \{a, b\})$

- $L(R) \; = \; \{w : w$ has equal number of occurrences of ab and $ba\}$

  Claim: w in $L(R)$ has length ≤ 1 or starts and ends with same char

  $$R \; = \; \epsilon + a + b + a(a+b)^*a + b(a+b)^*b$$

- $L(R) \; = \; \{w : w$ has odd number of b's $\}$

  $$R \; = \; a^*ba^*(a^*ba^*ba^*)^*$$

- $L(R) \; = \; \Sigma^*$

  $$R \; = \; (a+b)^*$$

- $L(R) \; = \; \{w : w$ has even length and b's at odd positions$\}$

  $$R \; = \; (b(a+b))^*$$

Language described by Regexp

Let $L(R)$ be the language represented by regexp R $\quad (\Sigma = \{a, b\})$

- $L(R) \ = \ \{w : w \text{ contains substring } ab \text{ or } ba \ \}$

$$R \ = \ \big((a+b)^*ab(a+b)^*\big) + \big((a+b)^*ba(a+b)^*\big)$$

- $L(R) \ = \ \{w : w \text{ does not contains substring } ab \text{ or } ba \ \}$

  complement of the above language

  How to write complement?

$$R \ = \ a^* \cup b^*$$

- $L(R) = \{w :$
  w has no more than 2 consecutive a's or 2 consecutive b's$\}$

$$R \ = \ (\epsilon + a + aa)\big((b+bb)(a+aa)\big)^*(\epsilon + b + bb)$$

Language described by Regexp

Let $L(R)$ be the language represented by regexp R

- $R = \emptyset^*$

$$L(R) = \{\epsilon\}$$

- $(\Sigma = \{a, b\})$ $R = a^*b^*$

$$L(R) = \{a^m b^n : m \geq 0, n \geq 0\}$$

- $R = DD^* \cdot D^* + D^* \cdot DD*$ $\qquad \triangleright D = \{0, 1, \ldots, 9\}$

$L(R) =$ decimal numbers, requiring a digit before or after decimal

Regexps in Real World

Regular expressions are commonly used to specify syntax

- For (portion) of programming languages
- For commands in command line interface
- In Text Editors (particularly for searching) ▷ grep

```
"regex">^.*\b(one|two|three)\b.*$    //one,two,three

//for deleting duplicate lines from text file...

"regex">^(.*)(\r?\n\1)+$
//For checking digits..

^[0-9]
//for checking alphabets...

^[a-z]
"regex">//REgular Expression for some Credit cards v

"regex">
//"Visa"

^[4]([0-9]{15}$|[0-9]{12}$)
//"MasterCard"
```

```
$ grep 'Smith' people
Personal        J.Smith         25000
Personal        E.Smith         25400
$ grep '^Personal' people> npeople
$ cat npeople
Personal        J.Smith         25000
Personal        E.Smith         25400
Personal        F.Jones         25000
Personal        J.Maler         33000
$ grep '500$' people>>npeople
$ cat npeople
Personal        J.Smith         25000
Personal        E.Smith         25400
Personal        F.Jones         25000
Personal        J.Maler         33000
(Admin)         R.Bron          30500
Groundfloor     T.Smythe        30500
$ grep $(hostname) /etc/hosts
127.0.0.1       mx
$ egrep '(lp|kdm:)' /etc/passwd
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
kdm:x:1000:1000::/home/kdm:/bin/bash
$
```