Regular Expressions

Professor: Suman Saha

Regular Expression

Pattern matching

- What happens if, at a Unix/Linux shell prompt, you type

  ls *

  and press return?

- Suppose the current directory contains files called regfla.tex, regfla.aux, regfla.log, regfla.dvi, and regfla.aux. What happens if you type

  ls *.aux

  and press return?

Alphabets

An alphabet is specified by giving a finite set, Σ, whose elements are called symbols. For us, any set qualifies as a possible alphabet, so long as it is finite.

Examples:

- $\Sigma_1 = \{0,1,2,3,4,5,6,7,8,9\}$ – 10-element set of decimal digits.
- $\Sigma_2 = \{a,b,c,\ldots,x,y,z\}$ – 26-element set of lower-case characters of the English language
- $\Sigma_3 = \{S \mid S \subseteq \Sigma_1\}$ – 2^{10}-element set of all subsets of the alphabet of decimal digits.

Non-Example:

- $\mathbb{N} = \{0,1,2,3, \ldots\}$ –set of all non-negative whole numbers is not an alphabet, because it is infinite

String over an Alphabet

A string of length n (≥ 0) over an alphabet Σ is just an ordered n-tuple of elements of Σ, written without punctuation.

- Example: if $\Sigma = \{a, b, c\}$, then a, ab, aac, and $bbac$ are strings over Σ of lengths one, two, three and four respectively
- This string is called sentence or word

N.B. there is a unique string of length zero over Σ, called the null string (or empty string) and denoted ε

Regular Language

- A language is a set of strings over an alphabet. Thus
  - $\{a, ab, baa\}$ is a language over $\Sigma = \{a, b\}$
  - $\{0, 111\}$ is a language over $\Sigma = \{0, 1\}$
- The number of symbols in a string is called the length of the string. For a string w its length is represented by $|w|$.

- A Regular Language is a subset of all languages that can be defined by regular expressions

Regular Expression

- Each regular expression is a notation for a regular language.
- If A is a regular expression, then we write $L(A)$ to the language denoted by A
- *Single character*: 'c'
  - $L(\text{'c'}) = \{\text{"c"}\}$ (for any $c \in \Sigma$)
- *Concatenation*: AB (where A and B are regular expression)
  - $L(AB) = \{ab \mid a \in L(A) \text{ and } b \in L(B)\}$
  - Example: if $A = \text{'i'}$ and $B = \text{'f'}$ then $L(AB) = L(\text{'i' 'f'}) = \{\text{"if"}\}$
- *Union*:
  - $L(A \mid B) = L(A) \cup L(B)$
    $= \{s \mid s \in L(A) \text{ or } s \in L(B)\}$
  - Example: $L(\text{'if'} \mid \text{'then'} \mid \text{'else'}) = \{\text{"if"}, \text{"then"}, \text{"else"}\}$

Regular Expression

- So far, we do not have a notation for infinite languages
- Iteration: A^*
  - $L(A^*) = \{\text{“”} \mid L(A) \mid L(AA) \mid L(AAA) \mid \ldots\}$
  - Example: $L(0^*) = \{\text{“”}, \text{“0”}, \text{“00”}, \text{“000”}, \ldots\}$
- Epsilon: ε
  - $L(\varepsilon) = \{\text{“”}\}$
- If (A) is a regular expression same as the regular expression A

Top Hat

PennState

Notational Conveniences

- Regular Definition is a sequence of the definitions of the form

$$d_1 \quad \rightarrow r_1$$
$$d_2 \quad \rightarrow r_2$$

where d_i is a distinct name and r_i is a regular expression over symbols in

$$\Sigma \cup \{d_1, d_2, \ldots d_{i-1}\}$$

- Example

$$\begin{aligned} &\mathit{letter} && \rightarrow && A \mid B \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \\ &\mathit{digit} && \rightarrow && 0 \mid 1 \mid \ldots \mid 9 \\ &\mathit{id} && \rightarrow && \text{letter}(\text{letter} \mid \text{digit})^* \end{aligned}$$

Notational Conveniences

- One or more instances: $r^+ = rr^*$
- Zero or one instance: $r? = r \mid \varepsilon$
- Character classes:

$$[abc]=a \mid b \mid c$$
$$[a\text{-}z]=a \mid b \mid \ldots \mid z$$
$$[0\text{-}9]=0 \mid 1 \mid \ldots \mid 9$$

- Any single character denoted by dot sign: .
- Negated character class: [^aeiou]
- Number of repetition: [a - f]{3}

Precedence of RE

- The order is (high to low)
  - Closure (*)
  - Concatenation
  - Alternation

Example:

- ab | cd means (ab) | (cd)
- a | bc*d means (a | (b(c*)d))

Examples

- Keywords
  - if, while, for, ….
- Identifiers
- Integers
- Whitespace: non-empty sequence of blanks, newlines, tabs
  - (‘\n’ | ‘\t’ | ‘ ’)+

Examples

- Float

- String constants

Special Characters in RE

- If you want to match $1{+}2{=}3$, you need to use a backslash ($\backslash$) to escape the + as this character has a special meaning (Match one or more of the previous).

- To match the $1{+}2{=}3$ as one string you would need to use $1\backslash{+}2{=}3$

Apply Regular Expression

- Suppose our Σ is all ASCII characters.
- A regular expression for even number is

(+ | -)? [0-9]*[02468]

42

+1370

-3248

-9999912

Apply Regular Expression

- Suppose our $\Sigma = \{a, @, .\}$ where a represents "some letter."
- A regular expression for email addresses is

$$a^+ (.a^+)^* @ a^+ (.a^+)^+$$

szk461@cse.psu.edu

first.middle.last@mail.site.org

my.president@whitehouse.gov

Build a Regular Expression

- $L = \{w \mid w$ is a binary string which does not contain two consecutive 0s or two consecutive 1s anywhere$\}$
  - E.g., $w = 01010101$ is in L, while $w = 10010$ is not in L
- Goal: Build a regular expression for L
- Four cases for w:
  - Case A: w starts with 0 and |w| is even
  - Case B: w starts with 1 and |w| is even
  - Case C: w starts with 0 and |w| is odd
  - Case D: w starts with 1 and |w| is odd

- Regular expression for four cases:
  - Case A: $(01)^*$
  - Case B: $(10)^*$
  - Case C: $0(10)^*$
  - Case D: $1(01)^*$

- Since L is the union of all 4 cases
  - Reg Exp for $L = (01)^* \mid (10)^* \mid 0(10)^* \mid 1(01)^*$
- If we introduce ε then the regular expression can be simplified to:
  - Reg Exam for $L = (\varepsilon \mid 1)\,(01)^*\,(\varepsilon \mid 0)$

Build a Regular Expression

- $\Sigma = \{a, b, c\}$, a string that has a symbol in the middle that is neither its start not its end symbol, and its and start symbols are different, for e.g., $\mathrm{abbbbbc}$ or bcccca

- Given an alphabet $\Sigma = \{a, b\}$, a string with a’s followed by b’s with both the number of a’s and b’s being equal.

Reading and Exercises

PennState

Reading

- Chapter: 2.1 (Michael Scott Book)

Exercises

- Exercises: 2.1, and 2.3 (Michael Scott Book)