CSC-257
Theory Of Computation
(BSc CSIT, TU)

Ganesh Khatri
kh6ganesh@gmail.com

Chapter 3 : Regular Expressions

- In previous lectures, we studied the languages in terms of machine like description-finite automata (DFA or NFA).
- Now we switch our attention to an algebraic description of languages, called regular expressions.
- Regular Expressions are those algebraic expressions used for representing regular languages, the languages accepted by finite automata.
- Regular expressions offer a declarative way to express the strings we want to accept.
- This is what the regular expressions offer that the automata do not.
- Many systems use regular expressions as input language. Some of them are
  - Search commands such as UNIX grep
  - Lexical analyzer generator such as LEX or FLEX. Lexical analyzer is a component of compiler that breaks the source program into logical unit called tokens
  - In programming languages for search operations.

Defining Regular expressions

- A regular expression is built up out of simpler regular expression using a set of defining rules
- Each regular expression 'r' denotes a language L(r)
- The defining rules specify how L(r) is formed by combining in various ways the languages denoted by the sub expressions of 'r'.
- **Method :** Let Σ be an alphabet, the regular expression over the alphabet Σ are defined inductively as follows;
- **Basic steps:**
  - Φ is a regular expression representing empty language.
  - Є is a regular expression representing the language of empty strings. i.e. {Є}
  - if 'a' is a symbol in Σ, then 'a' is a regular expression representing the language {a}

Defining Regular expressions

- Now the following operations over basic regular expression define the complex regular expression as:
- if ‘r’ and ‘s’ are the regular expressions representing the language L(r) and L(s) then –
  - r U s is a regular expression denoting the language L(r) U L(s).
  - r . s is a regular expression denoting the language L(r) . L(s).
  - r * is a regular expression denoting the language (L(r))*.
  - (r) is a regular expression denoting the language (L(r)). (this denote the same language as the regular expression ‘r’ denotes
- Note: any expression obtained from Φ, Є, a using above operation and parenthesis where required is a regular expression.

Regular Operators

- Basically, there are three operators that are used to generate the languages that are regular
- **Union (U / +) :** If L1 and L2 are any two regular languages then
  - L1 U L2 is the set of strings that are in either L or M, or both
  - i. e. L1 U L2 = { s | s ∈ L1, or s ∈ L2 }
  - For Example : if L1 = {00, 11} and L2 = (Є, 10}
  - Then L1 U L2 = { Є, 00, 11, 10 }
- **Concatenation (.) :** If L1 and L2 are any two regular languages then,
  - L1 . L2 = {l1 . l2 | l1 ∈ L1 and l2 ∈ L2}
  - For example : L1 = {00, 11} and L2 = {Є, 10}
  - then L1 . L2 = { 00, 11, 0010, 1110 }
  - L2 . L1 = { 1000, 1011, 00, 11}
  - So L1 . L2 != L2.L1

Regular Operators

- **Kleene Closure (*):**
  - If L is any regular Language then,
  - L* = Li = L0 U L1 U L2 U…………
  - **Example :**
  - Let L = { aa, b } then,
  - L0 = { Є }
  - L1 = { aa, b }
  - L2 = { aaaa, aab, baa, bb }
  - …
  - …
  - L* = L0 U L1 U L2 U…………
  - = all strings that can be obtained by concatenating 0 or more copies of aa and b

Precedence of regular operator

- The star operator is of highest precedence. i.e it applies to its left well formed RE.
- Next precedence is taken by concatenation operator.
- Finally, unions are taken

Precedence of regular operator

- **Example :** Write a RE for the set of strings that consists of alternating 0's and 1's over {0,1}
- **Solution :**
  - First part: we have to generate the language { 01, 0101, 0101,…………………}
  - Second part we have to generate the language { 10, 1010, 101010……………. }
  - So lets start first part.
  - Here we start with the basic regular expressions 0 and 1 that represent the language {0} and {1} respectively.
  - Now if we concatenate these two RE, we get the RE 01 that represent the language {01}.
  - Then to generate the language of zero or more occurrence of 01, we take Kleene closure. i.e. the RE (01)* represent the language { 01, 0101,…………. }
  - Similarly, the RE for second part is (10)*
  - Now finally, we take union of above two first part and second part to get the required RE. i.e. the RE (01)* + (10)* represents the given language

Regular Language

- Let Σ be an alphabet, the class of regular language over Σ is defined inductively as;
    - Φ is a regular language representing empty language
    - {Є} is a regular language representing language of empty strings
    - For each a ε Σ, {a} is a regular language
    - If L1, L2…………. Ln are regular languages, then so is L1 U L2 U ………. U Ln
    - If L1, L2, L3, ………….. Ln are regular languages, then so is L1 . L2 . L3………Ln
    - If L is a regular language, then so is L*
- **Note :** strictly speaking, a regular expression E is just an expression, not a language. We should use L(E) when we want to refer to the language that E denotes. However it is to common to refer to say E when we really mean L(E)

Applications of Regular Languages

- **Validation :**
  - Determining that a string complies with a set of formatting constraints.
  - Like email address validation, password validation etc.
- **Search and Selection :**
  - Identifying a subset of items from a larger set on the basis of a pattern match.
- **Tokenization :**
  - Converting a sequence of characters into words, tokens (like keywords, identifiers) for later interpretation

Algebraic Rules/Laws for Regular Expressions

- **Commutativity :**
  - Commutative of operator means we can switch the order of its operands and get the same result.
  - The union of regular expression is commutative but concatenation of regular expression is not commutative.
  - i.e. if r and s are regular expressions representing like languages L(r) and L(s) then, $r + s = s + r$ ($r \cup s = s \cup r$) but $r \,.\, s \neq s \,.\, r$
- **Associativity :**
  - The unions as well as concatenation of regular expressions are associative.
  - i.e. if t, r, s are regular expressions representing regular languages L(t), L(r) and L(s) then,
    $t + (r + s) = (t + r) + s$ and
    $t \,.\, (r \,.\, s) = (t \,.\, r) \,.\, s$

Algebraic Rules/Laws for Regular Expressions

- **Distributive law :**
  - For any regular expressions r, s, t representing regular languages L(r), L(s) and L(t) then,
    r(s + t) = rs + rt ------ left distribution
    (s + t)r = sr + tr ------ right distribution
- **Identity law :**
  - for any regular expression r representing regular expression L(r),
  - ϕ is identity for union. i.e. r + ϕ = ϕ + r = r (ϕ U r = r)
  - Є is identity for concatenation. i.e. Є . r = r = r . Є
- **Annihilator :**
  - An annihilator for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is annihilator.
  - ϕ is annihilator for concatenation. i.e. ϕ . r = r . ϕ = ϕ

Algebraic Rules/Laws for Regular Expressions

- **Idempotent Law of Union :**
  - For any regular expression r representing the regular language L(r), $r + r = r$
  - This is the idempotent law of union
- **Law of Closure :**
  - for any regular expression r representing the regular language L(r),
  - $(r^*)^* = r^*$
  - Closure of $\phi = \phi^* = \epsilon$
  - Closure of $\epsilon = \epsilon^* = \epsilon$
  - Positive closure of r, $r^+ = rr^*$

Regular Expressions : Examples

- **Consider Σ = {0, 1}, then some regular expressions over Σ are :**
  - 0*10* is RE that represents language { w|w contains a single 1 }
  - Σ*1Σ* is RE for language {w|w contains at least single 1 }
  - Σ*001 Σ* is RE for { w|w contains the string 001 as substring }
  - (ΣΣ)* or ((0+1)*.(0+1)*) is RE for { w|w is string of even length }
  - 1*(01*01*)* is RE for { w|w is string containing even number of zeros }
  - 0*10*10*10* is RE for { w|w is a string with exactly three 1's }
  - (1+0)*.001.(1+0)*+(1+0)*.(100).(1+0)* = { w|w is a string with either 001 or 100 as substring }
  - 1*.(0+Є).1*.(0+Є).1* = { w|w is a string having at most two 0's }
  - (1+0)*.(11)+ = { w|w is a string ending with 11 }
  - (Alphabet + _)(Alphabet + digit + _)* = Regular expression that denotes the C identifiers (C identifiers always start with an alphabet or underscore)