Theory of Automata
Regular Expressions

Week 2

Contents

Regular Expressions

- RE is the sequence of characters or symbols that represent a finite or infinite set of text strings.
- *Pattern-matching* is the process of checking whether a text string conforms to a set of characteristics defined by patterns such as regular expressions.
- A regular expression is a set of pattern matching rules encoded in a stringA regular expression is a set of pattern matching rules encoded in a string according to certain syntax rules. Although the syntax is somewhat complex it is very powerful and allows much more useful pattern matching than say simple wildcards like ? and *.

Regular Expression

- A regular expression (sometimes abbreviated to "regex") is a way for a computer user or programmer to express how a computer program should look for a specified pattern in text and then what the program is to do when each pattern match is found.
- For example, a regular expression could tell a program to search for all text lines that contain the word "Windows 95" and then to print out each line in which a match is found or substitute another text sequence (for example, just "Windows") where any match occurs.
- The best known tool for specifying and handling the incidence of regular expressions is grepThe best known tool for specifying and handling the incidence of regular expressions is grep, a utility found in Unix-based operating systems and also offered as a separate utility program for Windows and other operating systems.

Language-Defining Symbols

- We now introduce the use of the Kleene star, applied not to a set, but directly to the letter x and written as a superscript: x*.
- This simple expression indicates some sequence of x's (may be none at all):

  $\mathbf{x}^* = \Lambda$ or x or x^2 or x^3...

  $= x^n$ for some $n = 0, 1, 2, 3, \ldots$

- Letter **x** is intentionally written in boldface type to distinguish it from an alphabet character.

- We can think of the star as an unknown power. That is, **x*** stands for a string of x's, but we do not specify how many, and it may be the null string .

R.E. Continued…

- The notation x* can be used to define languages by writing, say L_4 = language (x*)

- Since x* is any string of x's, L_4 is then the language of all possible strings of x's of any length (including Λ).

- *We should not confuse x* (which is a **language-defining symbol**) with L_4 (which is the **name** we have given to a certain language).*

R.E. Continued…

- Given the alphabet = {a, b}, suppose we wish to define the language L that contains all words of the form: one ***a*** followed by some number of *b*'s (maybe no *b*'s at all); that is
- L = {a, ab, abb, abbb, abbbb, …}
- Using the language-defining symbol, we may write

  L = language (ab*)
- This equation obviously means that L is the language in which the words are the concatenation of an initial a with some or no b's.
- *From now on, for convenience, we will simply say* ***some*** *b**'s** to mean* ***some or no*** *b**'s**. When we want to mean* ***some positive number of*** *b**'s**, we will explicitly say so.*

R.E. Continued…

- We can apply the Kleene star to the whole string ab if we want:

  $(ab)^* = \Lambda$ or ab or abab or ababab…

- Observe that

  $(ab)^* \neq a^*b^*$

  - because the language defined by the expression on the left contains the word abab, whereas the language defined by the expression on the right does not.

R.E. Continued…

- If we want to define the language L1 = {x, xx, xxx, …} using the language-defining symbol, we can write

  L1 = language(xx*)

  which means that each word of L1 must start with an x followed by some (or no) x's.

- Note that we can also define L1 using the notation + (as an exponent) introduced in Chapter 2:

  L1 = language(x^+)

  – which means that each word of L1 is a string of some positive number of x's.

Alternation, Either/OR, Disjunction, Plus Sign

- Let us introduce another use of the plus sign. By the expression

$$x + y$$

where x and y are strings of characters from an alphabet, we mean **either** x **or** y.

- Care should be taken so as not to confuse this notation with the notation + (as an exponent) or with sign for arithmetic addition.

Example

- Consider the language T over the alphabet

  $\Sigma = \{a; b; c\}$:

- T = {a; c; ab; cb; abb; cbb; abbb; cbbb; abbbb; cbbbb; …}
- In other words, all the words in T begin with either an a or a c and then are followed by some number of b's.
- Using the above plus sign notation, we may write this as

  T = language((a+ c)b*)

Example

- Consider a finite language L that contains all the strings of a's and b's of length three exactly:

  L = {aaa, aab, aba, abb, baa, bab, bba, bbb}
- Note that the first letter of each word in L is either an a or a b; so are the second letter and third letter of each word in L.
- Thus, we may write

  $$L = \text{language}((a+b)(a+b)(a+b))$$
- or for short,

  $$L = \text{language}((a+b)^3)$$

Example

- In general, if we want to refer to the set of all possible strings of a's and b's of any length whatsoever, we could write

  language((a+ b)*)

- This is the set of **all possible strings** of letters from the alphabet Σ = {a, b}, **including the null string**.

- This is powerful notation. For instance, we can describe all the words that begin with first an ***a***, followed by anything (i.e., as many choices as we want of either a or b) as

  a(a + b)*

Formal Definition of Regular Expressions

- The set of **regular expressions** is defined by the following rules:
- **Rule 1:** Every letter of the alphabet Σ can be made into a regular expression by writing it in **boldface, Λ** itself is a regular expression.
- **Rule 2:** If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, then so are:
  - **(i)** $\mathbf{(r_1)}$
  - **(ii)** $\mathbf{r_1r_2}$
  - **(iii)** $\mathbf{r_1 + r_2}$
  - **(iv)** $\mathbf{r_1*}$
- **Rule 3:** Nothing else is a regular expression.
- Note: If r_1 = aa + b then when we write r_1*, we really mean (r_1)*, that is r_1* = (r_1)* = (aa + b)*

Example

- Consider the language defined by the expression

  (a + b)*a(a + b)*

- At the beginning of any word in this language we have *(a + b)**, which is any string of *a*’s and *b*’s, then comes an *a*, then another any string.

- For example, the word abbaab can be considered to come from this expression by 3 different choices:

  (Λ)a(bbaab) or (abb)a(ab) or (abba)a(b)

Example contd.

- This language is the set of all words over the alphabet Σ = {a, b} that have at least one a.
- The only words left out are those that have only b's and the word Λ.

These left out words are exactly the language defined by the expression b*.

- If we combine this language, we should provide a language of all strings over the alphabet Σ = {a, b}. That is,

$$(a + b)^* = (a + b)^*a(a + b)^* + b^*$$

Example

- Write RE to define the language of all words that have at least two a's :

  $(a + b)^*a(a + b)^*a(a + b)^*$

- Another expression that defines all the words with at least two a's is

  $b^*ab^*a(a + b)^*$

- Hence, we can write

  $(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$

  where by the equal sign we mean that these two expressions are **equivalent** in the sense that they describe the same language.

Example

- The language of all words that have at least one **a** and at least one **b** is somewhat trickier. If we write

  $$(a + b)^*a(a + b)^*b(a + b)^*$$

  then we are requiring that an ***a*** must precede a ***b*** in the word. Such words as ba and bbaaaa are not included in this language.

- Since we know that either the ***a*** comes before the ***b*** or the ***b*** comes before the ***a***, we can define the language by the expression

  $$(a + b)^*a(a + b)^*b(a + b)^* + (a + b)^*b(a + b)^*a(a + b)^*$$

- Note that the only words that are omitted by the first term $(a + b)^*a(a + b)^*b(a + b)^*$ are the words of the form some b's followed by some a's. They are defined by the expression bb^*aa^*

Example

- We can add these specific exceptions. So, the language of all words over the alphabet $\Sigma = \{a, b\}$ that contain at least one ***a*** and at least one ***b*** is defined by the expression:

  $$(a + b)a(a + b)b(a + b) + bb^*aa^*$$

- Thus, we have proved that

$$(a + b)^*a(a + b)^*b(a + b)^* + (a + b)^*b(a + b)^*a(a + b)^*$$

$$= (a + b)^*a(a + b)^*b(a + b)^* + bb^*aa^*$$

Example

- In the above example, the language of all words that contain both an a and a b is defined by the expression

  (a + b)*a(a + b)*b(a + b)* + bb*aa*

- The only words that do not contain both an a and a b are the words of all a's, all b's, or Λ.

- When these are included, we get everything. Hence, the expression

  (a + b)*a(a + b)*b(a + b)* + bb*aa* + a* + b*

defines all possible strings of a's and b's, including (accounted for in both a and b).

- Thus

$$(a + b)^* = (a + b)^*a(a + b)^*b(a + b)^* + bb^*aa^* + a^* + b^*$$

Example

- The following equivalences show that we should not treat expressions as algebraic polynomials:

$$(a + b)^* = (a + b)^* + (a + b)^*$$
$$(a + b)^* = (a + b)^* + a^*$$
$$(a + b)^* = (a + b)^*(a + b)^*$$
$$(a + b)^* = a(a + b)^* + b(a + b)^* + \Lambda$$
$$(a + b)^* = (a + b)^*ab(a + b)^* + b^*a^*$$

- The last equivalence may need some explanation:
  - The first term in the right hand side, $(a + b)^*ab(a + b)^*$, describes all the words that contain the substring ab.
  - The second term, b^*a^* describes all the words that do not contain the substring ab (i.e., all a's, all b's, Λ, or some b's followed by some a's).

Example

- Let V be the language of all strings of a's and b's in which either the strings are all b's, or else an a followed by some b's. Let V also contain the word Λ. Hence,

  V = {Λ, a, b, ab, bb, abb, bbb, abbb, bbbb, ...}
- We can define V by the expression

  $$b^* + ab^*$$

  where Λ is included in b*.
- Alternatively, we could define V by

  $$(\Lambda + a)b^*$$

which means that in front of the string of some b's, we have either an a or nothing.

Example contd.

- Hence,

  $(\Lambda + a)b^* = b^* + ab^*$

- Since $b^* = \Lambda b^*$, we have

  $(\Lambda + a)b^* = b^* + ab^*$

which appears to be distributive law at work.

- However, we must be extremely careful in applying distributive law. Sometimes, it is difficult to determine if the law is applicable.

Product Set

- If S and T are sets of strings of letters (whether they are finite or infinite sets), we define the **product set** of strings of letters to be

ST = {all combinations of a string from S
concatenated with a string from T in that order}

Example

- If S = {a, aa, aaa} and T = {bb, bbb} then

ST = {abb, abbb, aabb, aabbb, aaabb, aaabbb}

  - Note that the words are not listed in lexicographic order.

- Using regular expression, we can write this example as

(a + aa + aaa)(bb + bbb)

= abb + abbb + aabb + aabbb + aaabb + aaabbb

Example

- If M = {Λ, x, xx} and N = {Λ, y, yy, yyy, yyyy, …} then
- MN ={Λ, y, yy, yyy, yyyy,…x, xy, xyy, xyyy, xyyyy, …xx, xxy, xxyy, xxyyy, xxyyyy, …}

- Using regular expression

$(\Lambda + x + xx)(y^*) = y^* + xy^* + xxy^*$

Languages Associated with Regular Expressions

Definition

- The following rules define the **language associated** with any regular expression:

- **Rule 1:** The language associated with the regular expression that is just a single letter is that one-letter word alone, and the language associated with Λ is just {Λ}, a one-word language.

- **Rule 2:** If r_1 is a regular expression associated with the language L_1 and r_2 is a regular expression associated with the language L_2, then:

  (i) The regular expression $(r_1)(r_2)$ is associated with the product L_1L_2, that is the language L_1 times the language L_2:

$$\text{language}(r_1r_2) = L_1L_2$$

Definition contd.

- Rule 2 (cont.):

  (ii) The regular expression $r_1 + r_2$ is associated with the language formed by the union of L_1 and L_2:

  language($r_1 + r_2$) = $L_1 + L_2$

  (iii) The language associated with the regular expression $(r_1)^*$ is L_1^*, the Kleene closure of the set L_1 as a set of words:

  language(r_1^*) = L_1^*

Finite Languages Are Regular

Theorem 5

- **If L is a finite language (a language with only finitely many words), then L can be defined by a regular expression. In other words, all finite languages are regular.**
- *Proof*
- Let L be a finite language. To make one regular expression that defines L, we turn all the words in L into boldface type and insert plus signs between them.

- For example, the regular expression that defines the language

  L = {baa, abbba, bababa} is baa + abbba + bababa

- This algorithm only works for finite languages because an infinite language would become a regular expression that is infinitely long, which is forbidden.

How Hard It Is To Understand A Regular Expression

Let us examine some regular expressions and see if we could understand something about the languages they represent.

Example

- Consider the expression

  (a + b)*(aa + bb)(a + b)* =(arbitrary)(double letter)(arbitrary)

- This is the set of strings of a's and b's that at some point contain a double letter.

Let us ask, "What strings do not contain a double letter?" Some examples are

Λ; a; b; ab; ba; aba; bab; abab; baba; ...

Example contd.

- The expression (ab)* covers all of these except those that begin with b or end with a. Adding these choices gives us the expression:

$$(\Lambda + b)(ab)^*(\Lambda + a)$$

- Combining the two expressions gives us the one that defines the set of all strings

$(a + b)^*(aa + bb)(a + b)^* + (\Lambda + b)(ab)^*(\Lambda + a)$

Examples

- Note that

$$(a + b^*)^* = (a + b)^*$$

since the internal * adds nothing to the language. However,

$$(aa + ab^*)^* \neq (aa + ab)^*$$

since the language on the left includes the word *abbabb*, whereas the language on the right does not. (The language on the right cannot contain any word with a double b.)

Example

- Consider the regular expression: (a*b*)*.

- The language defined by this expression is all strings that can be made up of factors of the form a*b*.

- Since both the single letter a and the single letter b are words of the form a*b*, this language contains all strings of a's and b's. That is,

$$(a^*b^*)^* = (a + b)^*$$

- This equation gives a big doubt on the possibility of finding a set of algebraic rules to reduce one regular expression to another equivalent one.

Introducing EVEN-EVEN

- Consider the regular expression

  $E = [aa + bb + (ab + ba)(aa + bb)^*(ab + ba)]^*$

- This expression represents all the words that are made up of *syllables* of three types:

  $type_1 = aa$

  $type_2 = bb$

  $type_3 = (ab + ba)(aa + bb)^*(ab + ba)$

- Every word of the language defined by E contains an **even number of** a**'s and an even number of** b**'s**.

- All strings with an **even number of** a**'s and an even number of** b**'s** belong to the language defined by E.

Algorithms for EVEN-EVEN

- We want to determine whether a long string of a’s and b’s has the property that the number of a’s is even and the number of b’s is even.

- Algorithm 1: Keep two binary flags, the a-flag and the b-flag. Every time an a is read, the a-flag is reversed (0 to 1, or 1 to 0); and every time a b is read, the b-flag is reversed. We start both flags at 0 and check to be sure they are both 0 at the end.

- Algorithm 2: Keep only one binary flag, called the $type_3$-flag. We read letter in two at a time. If they are the same, then we do not touch the $type_3$-flag, since we have a factor of $type_1$ or $type_2$. If, however, the two letters do not match, we reverse the $type_3$-flag. If the flag starts at 0 and if it is also 0 at the end, then the input string contains an even number of a’s and an even number of b’s.

EVEN-EVEN

- If the input string is

aaabbbbaabbbbbbbababbbbaaa

Then by factoring in sub-strings of two letters each:

(aa)(ab)(bb)(ba)(ab)(bb)(bb)(bb)(ab)(ab)(bb)(ba)(aa)

0 1 1 0 1 1 1 1 0 1 1 0 0

by Algorithm 2, the $type_3$-flag is reversed 6 times and ends at 0.

- We give this language the name EVEN-EVEN. so, EVEN-EVEN ={Λ, aa, bb, aaaa, aabb, abab, abba, baab, baba, bbaa, bbbb, aaaaaa, aaaabb, aaabab, …}

Ex-1

- Find a regular expression for the set A of binary strings which have no substring 001.
- Solution. A string x in this set has no substring 00, except that it may have a suffix 0^K for k > 2.
- The set of strings with no substring 00 can be represented by the regular expression

$(01 + 1)^*(\lambda+0)$

- Therefore, set A has a regular expression

$(01 + 1)^*(\lambda + 0 + 000^*) = (01 + 1)^*0^*$

Ex-2

- Find a regular expression for the set B of all binary strings with at most one pair of consecutive 0 's and at most one pair of consecutive **1**s.
- Solution. A string x in B may have one of the following forms:
- (1) λ
- (2) $u_1 0$ (4) $u_1 0\, 0 v_1$ (6) $u_1 0\, 0 w_1 11 v_0$
- (3) $u_0 1$ (5) $u_0 11 v_0$ (7) $u_0 11 w_0 00 v_1$
- where u_0 , u_1 , v_0 , v_1 , w_0 , w_1 are strings with no substring 00 or 11, and u_0 ends with 0, u_1 ends with 1, v_0 begins with 0, v_1 begins with 1, w_0 begins with 0 and ends with 1, and w_1 begins with 1 and ends with 0.
- Now, observe that these types of strings can be represented by simple regular expressions: