Theory of Automata Regular Expressions

Mr. Hafiz Tayyeb Javed

Week 3 Lecture 01

Concluding remarks on RE

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 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

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

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.

- 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, …}

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.

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

- 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: