Theory of Automata

Shakir Ullah Shah

Lecture 3

Operation on Languages

- Union, intersection and difference --- same as on sets,
- Let Σ ={a,b}
- {a,ba,ab} ∩ {Λ, a, aa, aaa,…}=?

Operation on Languages

- Union, intersection and difference --- same as on sets,
- Let $\Sigma = \{a,b\}$
- $\{a,ba,ab\} \cap \{\Lambda, a, aa, aaa,\ldots\}=\{a\}$

Operation on Languages

- Union, intersection and difference --- same as on sets,
- Let $\Sigma = \{a,b\}$
- $\{a,ba,ab\} \cap \{\Lambda, a, aa, aaa, \ldots\} = \{a\}$
- Complement: Let $L = \{\Lambda, a, aa, aaa, ..\}$

Operation on Languages

- Union, intersection and difference ---same as on sets,
- Let Σ ={a,b}
- {a,ba,ab} ∩ {Λ, a, aa, aaa,...}={a}
- Complement: Let L={Λ, a,aa,aaa,..}
- $\overline{L}$={w: w includes all b's}
- Reverse: Let L={a,ba, abc}
- L^R ={a,ab,cba}

Arithmetic Expression (AE)

- Alphabet of valid arithmetic expression
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)\}$

Arithmetic Expression (AE)

- Alphabet of valid arithmetic expression
- Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)}

- (3 + 5) + 6)   2(/8 + 9)   (3 + (4-)8)

Arithmetic Expression (AE)

- Alphabet of valid arithmetic expression
- Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)}

- (3 + 5) + 6)   2(/8 + 9)   (3 + (4-)8)

- The first contains unbalanced parentheses; the second contains the forbidden substring /; the third contains the forbidden substring -).

Arithmetic Expression (AE)

- Alphabet of valid arithmetic expression
- Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)}

- (3 + 5) + 6)   2(/8 + 9)   (3 + (4-)8)

- The first contains unbalanced parentheses; the second contains the forbidden substring /; the third contains the forbidden substring -).

Recursive Definition of AE

- **Rule 1:** Any number (positive, negative, or zero) is in AE.

Recursive Definition of AE

- **Rule 1:** Any number (positive, negative, or zero) is in AE.

- **Rule 2:** If x is in AE, then so are

(i) (x)
*(ii) -*x (provided that x does not already start with a minus sign)

Recursive Definition of AE

- **Rule 1:** Any number (positive, negative, or zero) is in AE.

- **Rule 2:** If x is in AE, then so are

(i) (x)
(ii) -x (provided that x does not already start with a minus sign)

- **Rule 3:** If x and y are in AE, then so are

(i) x + y (if the first symbol in y is not + or -)
(ii) x - y (if the first symbol in y is not + or -)
(iii) x * y
(iv) x / y
(v) x ** y (our notation for exponentiation)

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:
- Is (2 + 4) OK?

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:
- Is (2 + 4) OK? Yes

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:
- Is (2 + 4) OK? Yes
- Is (9 - 3) OK?

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:
- Is (2 + 4) OK? Yes
- Is (9 - 3) OK? Yes

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:
- Is (2 + 4) OK? Yes
- Is (9 - 3) OK? Yes
- Is 7 * (9 - 3)/4 OK?

Recursive Definition of AE

- (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) – 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:
- Is (2 + 4) OK? Yes
- Is (9 - 3) OK? Yes
- Is 7 * (9 - 3)/4 OK? Yes, and so on.

Defining Languages by Another New Method Regular Expression (RE)

Recursive definition of Regular Expression(RE)

- Step 1: Every letter of Σ including Λ is a regular expression.
- Step 2: If R1 and R2 are regular expressions then
   1. (R1)
   2. R1 R2
   3. R1 + R2 and
   4. R1*

are also regular expressions.

- Step 3: Nothing else is a regular expression.

Regular Expression (RE)

- a*=

Regular Expression (RE)

- $a^* = \{\wedge, a, aa, aaa, aaaa, \ldots\}$

Regular Expression (RE)

- $a^* = \{\Lambda, a, aa, aaa, aaaa, \ldots\} = a^0, a^1, a^2, a^3, \ldots$

Regular Expression (RE)

- $a^* = \{\wedge, a, aa, aaa, aaaa, \ldots\} = a^0, a^1, a^2, a^3, \ldots$
- $a^+ = \{a, aa, aaa, aaaa, \ldots\} = a^1, a^2, a^3, \ldots$

Regular Expression (RE)

- $a^* = \{\Lambda, a, aa, aaa, aaaa, \ldots\} = a^0, a^1, a^2, a^3, \ldots$
- $a^+ = \{a, aa, aaa, aaaa, \ldots\} = a^1, a^2, a^3, \ldots$
- $b^+ = \{b, bb, bbb, bbbb, \ldots\}$

Regular Expression (RE)

- $a^* = \{\Lambda, a, aa, aaa, aaaa, \ldots\} = a^0, a^1, a^2, a^3, \ldots$
- $a^+ = \{a, aa, aaa, aaaa, \ldots\} = a^1, a^2, a^3, \ldots$
- $b^+ = \{b, bb, bbb, bbbb, \ldots\}$
- $L = \{a, ab, abb, abbb, abbbb, \ldots\}$

Regular Expression (RE)

- $a^* = \{\Lambda, a, aa, aaa, aaaa, \ldots\} = a^0, a^1, a^2, a^3, \ldots$
- $a^+ = \{a, aa, aaa, aaaa, \ldots\} = a^1, a^2, a^3, \ldots$
- $b^+ = \{b, bb, bbb, bbbb, \ldots\}$
- L = {a, ab, abb, abbb, abbbb, …}
- L = language (ab*)

Regular Expression (RE)

- $a^* = \{\Lambda, a, aa, aaa, aaaa, \ldots\} = a^0, a^1, a^2, a^3, \ldots$
- $a^+ = \{a, aa, aaa, aaaa, \ldots\} = a^1, a^2, a^3, \ldots$
- $b^+ = \{b, bb, bbb, bbbb, \ldots\}$
- L = {a, ab, abb, abbb, abbbb, …}
- L = language (ab*)
- L is the language in which the words are the concatenation of an initial a with some or no b's.

Regular Expression (RE)

- We can apply the Kleene star to the whole string ab if we want:
  $(ab)^* = \Lambda$ or ab or abab or ababab...
- Observe that
  $(ab)^*$? a^*b^*

Regular Expression (RE)

- We can apply the Kleene star to the whole string ab if we want:
  $(ab)^* = \Lambda$ or ab or abab or ababab…
- Observe that
  $(ab)^* \neq a^*b^*$

Regular Expression (RE)

- We can apply the Kleene star to the whole string ab if we want:
  (ab)* = Λ or ab or abab or ababab...
- Observe that
  (ab)* ≠ 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

Regular Expression (RE)

- a*+ b* ? (a+b)*

Regular Expression (RE)

- $a^* + b^* \neq (a+b)^*$

Regular Expression (RE)

- $a^* + b^* \neq (a+b)^*$
- Here a^*+b^* does not generate any string of concatenation of a and b, while $(a+b)^*$ generates such strings.
- $(a + b^*)^*$? $(a + b)^*$

Regular Expression (RE)

- $a^* + b^* \neq (a+b)^*$
- Here a^*+b^* does not generate any string of concatenation of a and b, while $(a+b)^*$ generates such strings.
- $(a + b^*)^* = (a + b)^*$

  since the internal * adds nothing to the language.

Regular Expression (RE)

- Plus sign:
- Let us introduce another use of the plus sign. Let $\Sigma=\{a,b\}$. By the expression

  $a + b$

  – means

Regular Expression (RE)

- Plus sign:
- Let us introduce another use of the plus sign. Let $\Sigma=\{a,b\}$. By the expression

  $a + b$

  - means **either** a **or** b.

Regular Expression (RE)

- Plus sign:
- Let us introduce another use of the plus sign. Let Σ={a,b}. By the expression

  a + b

  – means **either** a **or** b.
- Care should be taken so as not to confuse this notation with the notation + (as an exponent).

Regular Expression (RE)

- Plus sign:
- {ab,bc}=ab+bc

Regular Expression (RE)

- Plus sign:
- {ab,bc}=ab+bc
- {abb,bcb}=

Regular Expression (RE)

- Plus sign:
- {ab,bc}=ab+bc
- {abb,bcb}=(ab+bc).b
- {a,b}*=

Regular Expression (RE)

- Plus sign:
- {ab,bc}=ab+bc
- {abb,bcb}=(ab+bc).b
- {a,b}*=(a+b)*
- {ac,c}=

Regular Expression (RE)

- Plus sign:
- {ab,bc}=ab+bc
- {abb,bcb}=(ab+bc).b
- {a,b}*=(a+b)*
- {ac,c}=(a+ Λ).c
- {Λ,a,b,ab}=

Regular Expression (RE)

- Plus sign:
- {ab,bc}=ab+bc
- {abb,bcb}=(ab+bc).b
- {a,b}*=(a+b)*
- {ac,c}=(a+ Λ).c
- {Λ,a,b,ab}=(a+ Λ)(b+Λ)

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- $(a+b)(a+b)$
-

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- $(a+b)(a+b)$
- language of 2 length

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):
- language of 3 length

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- $(a+b)(a+b)$
- language of 2 length
- $(a+b)\ (a+b)(a+b)$:
- language of 3 length
- $(a+b)\ (a+b)\ (a+b)\ (a+b)$:

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):
- language of 3 length
- (a+b) (a+b) (a+b) (a+b):
- language of 4 length

Regular Expression (RE)

- Let Σ = {a, b}
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):
- language of 3 length
- (a+b) (a+b) (a+b) (a+b):
- language of 4 length
- (a+b)*:

Regular Expression (RE)

- Let Σ = {a, b}
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):
- language of 3 length
- (a+b) (a+b) (a+b) (a+b):
- language of 4 length
- (a+b)*: all strings including null

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):
- language of 3 length
- (a+b) (a+b) (a+b) (a+b):
- language of 4 length
- (a+b)*: all strings including null
- $(a+b)^+$:

Regular Expression (RE)

- Let $\Sigma = \{a, b\}$
- (a+b)(a+b)
- language of 2 length
- (a+b) (a+b)(a+b):
- language of 3 length
- (a+b) (a+b) (a+b) (a+b):
- language of 4 length
- $(a+b)^*$: all strings including null
- $(a+b)^+$: all strings without null

Regular Expression (RE)

- a(a+b)*:

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything

Regular Expression (RE)

- $a(a+b)^*$: begin with a followed by anything
- $b(a+b)^*$:

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b:

Regular Expression (RE)

- $a(a+b)^*$: begin with a followed by anything
- $b(a+b)^*$: begin with b followed by anything
- $(a+b)^*b$: end with b

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*:

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a
- (a+b)*aa(a+b)*:

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a
- (a+b)*aa(a+b)*: having double a

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a
- (a+b)*aa(a+b)*: having double a
- (a+b)*a(a+b)*a(a+b)*:

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a
- (a+b)*aa(a+b)*: having double a
- (a+b)*a(a+b)*a(a+b)*: having at least two a's

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a
- (a+b)*aa(a+b)*: having double a
- (a+b)*a(a+b)*a(a+b)*: having at least two a’s
- b*ab*a(a + b)*:

Regular Expression (RE)

- a(a+b)*: begin with a followed by anything
- b(a+b)*: begin with b followed by anything
- (a+b)*b: end with b
- (a+b)*a(a+b)*: having at least one a
- (a+b)*aa(a+b)*: having double a
- (a+b)*a(a+b)*a(a+b)*: having at least two a’s
- b*ab*a(a + b)*: having at least two a’s

Regular Expression (RE)

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

Regular Expression (RE)

- $((a+b)(a+b))^*$
- language L, of even length

Regular Expression (RE)

- ((a+b)(a+b))*
- language L, of even length
- (a+b)((a+b)(a+b))* or ((a+b)(a+b))*(a+b)

Regular Expression (RE)

- ((a+b)(a+b))*
- language L, of even length
- (a+b)((a+b)(a+b))* or ((a+b)(a+b))*(a+b)
- language L, of odd length

Regular Expression (RE)

- ((a+b)(a+b))*
- language L, of even length
- (a+b)((a+b)(a+b))* or ((a+b)(a+b))*(a+b)
- language L, of odd length
- a language may be expressed by more than one regular expressions, ***while*** given a regular expression there exist a unique language generated by that regular expression

Regular Expression (RE)

- starting with double a and ending in double b
- aa(a+b)*bb
- starting and ending with same letter
- a(a+b)*a + b(a+b)*b
- starting and ending with different letter
- a(a+b)*b+ b(a+b)*a
- ending with aa or bb
- (a+b)*(aa + bb)=(a+b)*aa +

Regular Expression (RE)

- Consider the regular expression
  $E = [aa + bb + (ab + ba)(aa + bb)^*(ab + ba)]^*$

Regular Expression (RE)

- 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:
  $type_1 = aa$
  $type_2 = bb$
  $type_3 = (ab + ba)(aa + bb)^*(ab + ba)$

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.

Algorithms for EVEN-EVEN

- If the input string is (aa)(ab)(bb)(ba)(ab)(bb)(bb)(bb)(ab)(ab)(bb)(ba)(aa) then, by Algorithm 2, the type$_3$-flag is reversed 6 times and ends at 0.

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

Regular Expression (RE)

- If r_1 =(aa + bb) and
- r_2 =(a + b) then

1. $r_1 + r_2$ =(aa + bb) + (a + b)
2. $r_1 r_2$ = (aa + bb) (a + b)

   = (aaa + aab + bba + bbb)
3. $(r_1)^*$ =(aa + bb)*