02-0: Alphabets & Strings

- An alphabet \( \Sigma \) is a finite set of symbols
  - \( \Sigma_1 = \{a, b, \ldots, z\} \)
  - \( \Sigma_2 = \{0, 1\} \)
- A string is a finite sequence of symbols from an alphabet
  - fire, truck are both strings over \( \{a, \ldots, z\} \)
- length of a string is the number of symbols in the string
  - \( |\text{fire}| = 4, |\text{truck}| = 5 \)

02-1: Alphabets & Strings

- The empty string \( \epsilon \) is a string of 0 characters
  - \( |\epsilon| = 0 \)
- \( \circ \) is the concatenation operator
  - \( w_1 = \text{fire}, w_2 = \text{truck} \)
  - \( w_1 \circ w_2 = \text{firetruck} \)
  - \( w_2 \circ w_1 = \text{truckfire} \)
  - \( w_2 \circ w_2 = \text{trucktruck} \)
- Often drop the \( \circ \): \( w_1 w_2 = \text{firetruck} \)
- For any string \( w \), \( w \epsilon = w \)

02-2: Concatenation & Reversal

- We can concatenate a string with itself:
  - \( w^1 = w \)
  - \( w^2 = ww \)
  - \( w^3 = www \)
- By definition, \( w^0 = \epsilon \)
- Can reverse a string: \( w^R \)
  - \( \text{truck}^R = \text{kcurt} \)

02-3: Formal Language

- A formal language (or just language) is a set of strings
  - \( L_1 = \{a, aa, abba, bbba\} \)
  - \( L_2 = \{\text{car, truck, goose}\} \)
  - \( L_3 = \{1, 11, 111, 1111, 11111, \ldots\} \)
- A language can be either finite or infinite
02-4: **Language Concatenation**

- We can concatenate languages as well as strings
- \( L_1 L_2 = \{ wv : w \in L_1 \land v \in L_2 \} \)
- \( \{a, ab\} \{bb, b\} = \)

02-5: **Language Concatenation**

- We can concatenate languages as well as strings
- \( L_1 L_2 = \{ wv : w \in L_1 \land v \in L_2 \} \)
- \( \{a, ab\} \{bb, b\} = \{abb, ab, abbb\} \)
- \( \{a, ab\} \{a, ab\} = \)

02-6: **Language Concatenation**

- We can concatenate languages as well as strings
- \( L_1 L_2 = \{ wv : w \in L_1 \land v \in L_2 \} \)
- \( \{a, ab\} \{bb, b\} = \{abb, ab, abbb\} \)
- \( \{a, ab\} \{a, ab\} = \{aa, aab, aba, abab\} \)
- \( \{a, aa\} \{a, aa\} = \)

02-7: **Language Concatenation**

- We can concatenate languages as well as strings
- \( L_1 L_2 = \{ wv : w \in L_1 \land v \in L_2 \} \)
- \( \{a, ab\} \{bb, b\} = \{abb, ab, abbb\} \)
- \( \{a, ab\} \{a, ab\} = \{aa, aab, aba, abab\} \)
- \( \{a, aa\} \{a, aa\} = \{aa, aaa, aaaa\} \)

What can we say about \( |L_1 L_2| \), if we know \( |L_1| = m \) and \( |L_2| = n \)?

02-8: **Language Concatenation**

- We can concatenate a language with itself, just like strings
- \( L^1 = L, L^2 = LL, L^3 = LLL, \) etc.
- What should \( L^0 \) be, and why?

02-9: **Language Concatenation**

- We can concatenate a language with itself, just like strings
- \( L^1 = L, L^2 = LL, L^3 = LLL, \) etc.
- \( L^0 = \{\epsilon\} \)
- \( \{\} \) is the empty language
• \{\epsilon\} is the trivial language

• Kleene Closure (L*)
  
  • \(L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots\)

02-10: Regular Expressions

• Regular expressions are a way to describe formal languages

• Regular expressions are defined recursively
  
  • Base case – simple regular expressions
  
  • Recursive case – how to build more complex regular expressions from simple regular expressions

02-11: Regular Expressions

• \(\epsilon\) is a regular expression, representing \(\{\epsilon\}\)

• \(\emptyset\) is a regular expression, representing \(\{\}\)

• \(a \in \Sigma\), a is a regular expression representing \(\{a\}\)

• if \(r_1\) and \(r_2\) are regular expressions, then \((r_1 r_2)\) is a regular expression
  
  • \(L[r_1 r_2] = L[r_1] \circ L[r_2]\)

• if \(r_1\) and \(r_2\) are regular expressions, then \((r_1 + r_2)\) is a regular expression
  
  • \(L[r_1 + r_2] = L[r_1] \cup L[r_2]\)

• if \(r\) is a regular expression, then \((r^*)\) is a regular expression
  
  • \(L[r^*] = (L[r])^*\)

02-12: Regular Expressions

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\epsilon)</td>
<td>(L[\epsilon] = {\epsilon})</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(L[\emptyset] = {})</td>
</tr>
<tr>
<td>(a \in \Sigma)</td>
<td>(L[a] = {a})</td>
</tr>
<tr>
<td>((r_1 r_2))</td>
<td>(L[r_1 r_2] = L[r_1] L[r_2])</td>
</tr>
<tr>
<td>((r_1 + r_2))</td>
<td>(L[r_1 + r_2] = L[r_1] \cup L[r_2])</td>
</tr>
<tr>
<td>((r^*))</td>
<td>(L[r^<em>] = (L[r])^</em>)</td>
</tr>
</tbody>
</table>

02-13: Regular Expressions

• \(((a+b)(b^*))a\)

• \(((a((a+b)^*))a\)

• \(((a^*)(b^*))\)

• \(((ab)^*)\)

02-14: Regular Expressions

• \(((a+b)(b^*))a\)
• \{aa, ba, aba, bba, abba, abbbba, abbba, bbbba, \ldots\}
• \(((a((a+b)^*))a)\)
  • \{aa, aaa, aba, aaaa, aaba, abaa, abba, \ldots\}
• \(((a^*)(b^*))\)
  • \{\varepsilon, a, b, aa, ab, bb, aaa, aab, abb, bbb, \ldots\}
• \(((ab)^*)\)
  • \{\varepsilon, ab, abab, ababab, abababab, \ldots\}

02-15: Regular Expressions

• All those parenthesis can be confusing
  • Drop them!!
  • \(((ab)b)a\) becomes abba
• What about a+bb*a – what’s the problem?

02-16: Regular Expressions

• All those parenthesis can be confusing
  • Drop them!!
  • \(((ab)b)a\) becomes abba
• What about a+bb*a – what’s the problem?
  • Ambiguous!
  • a+(b(b^*))a, (a+b)(b^*)a, (a+(bb))*a ?

02-17: r.e. Precedence

From highest to Lowest:

Kleene Closure *
Concatenation
Alternation +

ab^*c+e = (a(b^*)c) + e

(We will still need parentheses for some regular expressions: (a+b)(a+b)) 02-18: Regular Expressions

• Intuitive Reading of Regular Expressions
  • Concatenation == “is followed by”
  • + == ”or”
  • * == ”zero or more occurrences”
• \((a+b)(a+b)(a+b)\)

• \((a+b)^*\)

• \(aab(aa)^*\)

02-19: Regular Expressions

• All strings over \(\{a,b\}\) that start with an a

02-20: Regular Expressions

• All strings over \(\{a,b\}\) that start with an a
  • \(a(a+b)^*\)

• All strings over \(\{a,b\}\) that are even in length

02-21: Regular Expressions

• All strings over \(\{a,b\}\) that start with an a
  • \(a(a+b)^*\)

• All strings over \(\{a,b\}\) that are even in length
  • \(((a+b)(a+b))^*\)

• All strings over \(\{0,1\}\) that have an even number of 1’s.

02-22: Regular Expressions

• All strings over \(\{a,b\}\) that start with an a
  • \(a(a+b)^*\)

• All strings over \(\{a,b\}\) that are even in length
  • \(((a+b)(a+b))^*\)

• All strings over \(\{0,1\}\) that have an even number of 1’s.
  • \(0^*(10^*10^*)^*\)

• All strings over a, b that start and end with the same letter

02-23: Regular Expressions

• All strings over \(\{a,b\}\) that start with an a
  • \(a(a+b)^*\)

• All strings over \(\{a,b\}\) that are even in length
  • \(((a+b)(a+b))^*\)

• All strings over \(\{0,1\}\) that have an even number of 1’s.
  • \(0^*(10^*10^*)^*\)
• All strings over a, b that start and end with the same letter
  • (a+b)*a + b(a+b)*b + a + b

02-24: **Regular Expressions**

• All strings over \{0, 1\} with no occurrences of 00

02-25: **Regular Expressions**

• All strings over \{0, 1\} with no occurrences of 00
  • 1*(011*)*(0+1*)

• All strings over \{0, 1\} with exactly one occurrence of 00

02-26: **Regular Expressions**

• All strings over \{0, 1\} with no occurrences of 00
  • 1*(011*)*(0+1*)

• All strings over \{0, 1\} with exactly one occurrence of 00
  • 1*(011*)*00(11*0)*1*

• All strings over \{0, 1\} that contain 101

02-27: **Regular Expressions**

• All strings over \{0, 1\} with no occurrences of 00
  • 1*(011*)*(0+1*)

• All strings over \{0, 1\} with exactly one occurrence of 00
  • 1*(011*)*00(11*0)*1*

• All strings over \{0, 1\} that contain 101
  • (0+1)*101(0+1)*

• All strings over \{0, 1\} that do not contain 01

02-28: **Regular Expressions**

• All strings over \{0, 1\} with no occurrences of 00
  • 1*(011*)*(0+1*)

• All strings over \{0, 1\} with exactly one occurrence of 00
  • 1*(011*)*00(11*0)*1*

• All strings over \{0, 1\} that contain 101
  • (0+1)*101(0+1)*

• All strings over \{0, 1\} that do not contain 01
02-29: Regular Expressions

- All strings over \{:/, "*", a, . . . , z\} that form valid C comments
  - Use quotes to differentiate the "*" in the input from the regular expression *
  - Use [a-z] to stand for (a + b + c + d + . . . + z)

02-30: Regular Expressions

- All strings over \{:/, "*", a, . . . , z\} that form valid C comments
  - Use quotes to differentiate the "*" in the input from the regular expression *
  - Use [a-z] to stand for (a + b + c + d + . . . + z)
  - \(/"*/([a-z]+/)*/\)
  - \("*/\)\)*(\("*/\)([a-z]+/)*/\)\)*(\("*/\)\)*/\)

- This exact problem (finding a regular expression for C comments) has actually been used in an industrial context.

02-31: Regular Languages

- A language is **regular** if it can be described by a regular expression.
- The **Regular Languages**\(\(L_{REG}\)\) is the set of all languages that can be represented by a regular expression
  - Set of set of strings
- Raises the question: Are there languages that are not regular?
  - Stay tuned!