CMSC 330: Organization of Programming Languages

Theory of Regular Expressions

The Theory Behind r.e.’s

- That’s it for the basics of Ruby
  - If you need other material for your project, come to office hours or check out the documentation

- Next up: How do r.e.’s really work?
  - Mixture of a very practical tool (string matching with r.e.’s) and some nice theory
  - A great computer science result

A Few Questions about Regular Expressions

- What does a regular expression represent?
  - Just a set of strings

- What are the basic components of r.e.’s?
  - E.g., we saw that $e^*$ is the same as $ee^*$

- How are r.e.’s implemented?
  - We’ll see how to build a structure to parse r.e.’s

Definition: Alphabet

- An alphabet is a finite set of symbols
  - Usually denoted $\Sigma$

- Example alphabets:
  - Binary: $\Sigma = \{0, 1\}$
  - Decimal: $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - Alphanumeric: $\Sigma = \{0-9, a-z, A-Z\}$

Definition: String

- A string is a finite sequence of symbols from $\Sigma$
  - $\varepsilon$ is the empty string ("" in Ruby)
  - $|s|$ is the length of string $s$
    - $|\text{Hello}| = 5$, $|\varepsilon| = 0$
  - Note: $\emptyset$ is the empty set (with 0 elements); $\emptyset \neq \{ \varepsilon \}$

- Example strings:
  - $0101 \in \Sigma = \{0, 1\}$ (binary)
  - $0101 \in \Sigma = \text{decimal}$
  - $0101 \in \Sigma = \text{alphanumeric}$

Definition: Concatenation

- Concatenation is indicated by juxtaposition
  - If $s_1 =$ super and $s_2 =$ hero, then $s_1s_2 =$ superhero
  - Sometimes also written $s_1 \cdot s_2$
  - For any string $s$, we have $se = es = s$
  - You can concatenate strings from different alphabets, then the new alphabet is the union of the originals:
    - If $s_1 \in \Sigma_1 = \{s,u,p,e,r\}$ and $s_2 \in \Sigma_2 = \{h,e,r,o\}$, then $s_1s_2 \in \Sigma_3 = \{e,h,u,o,p,r,s,u\}$
Definition: Language

- A *language* is a set of strings over an alphabet.
- Example: The set of phone numbers over the alphabet \( \{0, 1, 2, 3, 4, 5, 6, 7, 9, (, ), -\} \)
  - Give an example element of this language \[ (123) 456-7890 \]
  - Are all strings over the alphabet in the language? **Yes**
  - Is there a Ruby regular expression for this language?
    \[ /\d{3,3}\d{3,3}-\d{4,4}/ \]
- Example: The set of all strings over the alphabet \( \{a, b, c\} \)
  - Often written \( \Sigma^* \)

Languages (cont’d)

- Example: The set of strings of length 0 over the alphabet \( \{a, b, c\} \)
  - \( \{s \mid s \in \Sigma^* \text{ and } |s| = 0\} = \emptyset \)
- Example: The set of all valid Ruby programs
  - Is there a Ruby regular expression for this language?
    No. Matching brackets so they are balanced is impossible.
    \[ {\{\} \text{ or } (\text{)} \text{ or, in general, } (\text{)})} \]
- Can r.e.’s represent all possible languages?
  - The answer turns out to be no!
  - The languages represented by regular expressions are called, appropriately, the *regular languages*.

Operations on Languages

- Let \( \Sigma \) be an alphabet and let \( L, L_1, L_2 \) be languages over \( \Sigma \).
- Concatenation \( L_1L_2 \) is defined as
  - \( L_1L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\} \)
  - Example: \( L_1 = \{"hi", "bye"\}, L_2 = \{"1", "2"\} \)
    - \( L_1L_2 = \{"hi1", "hi2", "bye1", "bye2"\} \)
- Union is defined as
  - \( L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\} \)
  - Example: \( L_1 = \{"hi", "bye"\}, L_2 = \{"1", "2"\} \)
    - \( L_1 \cup L_2 = \{"hi", "bye", "1", "2"\} \)

Operations on Languages (cont’d)

- Define \( L^n \) inductively as
  - \( L^0 = \\emptyset \)
  - \( L^n = LL^{n-1} \text{ for } n > 0 \)
- In other words,
  - \( L^1 = LL_0 = L\emptyset = L \)
  - \( L^2 = LL^1 = LL \)
  - \( L^3 = LL^2 = LLL \)
  - ...

Examples of \( L^n \)

- Let \( L = \{a, b, c\} \)
- Then
  - \( L^0 = \emptyset \)
  - \( L^1 = \{a, b, c\} \)
  - \( L^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\} \)

Operations on Languages (cont’d)

- *Kleene closure* is defined as
  \( L^* = \bigcup_{i=0}^{\infty} L^i \)
- In other words...
  \( L^* \) is the language (set of all strings) formed by concatenating together zero or more strings from \( L \).
Definition of Regexps

Given an alphabet $\Sigma$, the regular expressions over $\Sigma$ are defined inductively as

- ... 

<table>
<thead>
<tr>
<th>regular expression</th>
<th>denotes language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$\Sigma$</td>
</tr>
<tr>
<td>each element $a \in \Sigma$</td>
<td>${a}$</td>
</tr>
</tbody>
</table>

Definition of Regexps (cont’d)

- Let $A$ and $B$ be regular expressions denoting languages $L_A$ and $L_B$, respectively

<table>
<thead>
<tr>
<th>regular expression</th>
<th>denotes language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AB$</td>
<td>$L_A \cdot L_B$</td>
</tr>
<tr>
<td>$(A</td>
<td>B)$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$L_A^*$</td>
</tr>
</tbody>
</table>

- There are no other regular expressions over $\Sigma$
- We use $()$’s as needed for grouping

The Language Denoted by an r.e.

- For a regular expression $e$, we will write $\llbracket e \rrbracket$ to mean the language denoted by $e$
  - $\llbracket a \rrbracket = \{a\}$
  - $\llbracket (a|b) \rrbracket = \{a, b\}$

- If $s \in \llbracket re \rrbracket$, we say that $re$ accepts, describes, or recognizes $s$.

Example 1

- All strings over $\Sigma = \{a, b, c\}$ such that all the $a$’s are first, the $b$’s are next, and the $c$’s last
  - Example: $aaabbbbccc$ but not $abcb$
  - Regexp: $a^*b^*c^*$
    - This is a valid regexp because:
      - $a$ is a regexp ($\llbracket a \rrbracket = \{a\}$)
      - $a^*$ is a regexp ($\llbracket a^* \rrbracket = \{a, aa, \ldots\}$)
      - Similarly for $b^*$ and $c^*$
      - So $a^*b^*c^*$ is a regular expression
      (Remember that we need to check this way because regular expressions are defined inductively.)

Which Strings Does $a^*b^*c^*$ Recognize?

- $aabbcc$
  - Yes; $aa \in \llbracket a^* \rrbracket$, $bbb \in \llbracket b^* \rrbracket$, and $cc \in \llbracket c^* \rrbracket$, so entire string is in $\llbracket a^*b^*c^* \rrbracket$
- $abb$
  - Yes, $abb = abbx$, and $\varepsilon \in \llbracket c^* \rrbracket$
- $ac$
  - Yes
- $\varepsilon$
  - Yes
- $aacbc$
  - No
- $abcd$
  - No $\Rightarrow$ outside the language

Example 2

- All strings over $\Sigma = \{a, b, c\}$
- Regexp: $(a|b|c)^*$
  - Other regular expressions for the same language?
    - $(a|b)a^*$
    - $(a|b)b^*|c|^*$
    - $(a|b|c)^*$
    - $(a|b|c|a|b|c)^*$
    - $(a|b|c)^*|abc$
    - etc.
Example 3

- All whole numbers containing the substring 330
- Regular expression: \((0|1)\ldots(9)\*330(0|1)\ldots(9)\*\)
- What if we want to get rid of leading 0’s?
- \((1)\ldots(9)(0|1)\ldots(9)\*330(0|1)\ldots(9)\* | 330(0|1)\ldots(9)\*\)
- Any other solutions?
- Challenge: What about all whole numbers not containing the substring 330?
- Is it recognized by a regexp? Yes. We’ll see how to find it later...

Example 4

- What is the English description for the language that \((10|0)\*(10|1)\*\) denotes?
  - \((10|0)\*)
    - 0 may appear anywhere
    - 1 must always be followed by 0
  - \((10|1)\*)
    - 1 may appear anywhere
    - 0 must always be preceded by 1
  - Put together, all strings of 0’s and 1’s where every pair of adjacent 0’s precedes any pair of adjacent 1’s

Example 5

- What language does this regular expression recognize?
  - \((( (1|c)(0|1)\ldots(9) | (2(0|1)\ldots(3)) ) \cdot (0|1)\ldots(5)(0|1)\ldots(9)\)
- All valid times written in 24-hour format
  - 10:17
  - 23:59
  - 0:45
  - 8:30

Two More Examples

- \((000|00|1)\*)
  - Any string of 0’s and 1’s with no single 0’s
- \((00|0000)\*)
  - Strings with an even number of 0’s
  - Notice that some strings can be accepted more than one way
  \(000000 = 00\cdot 00 = 00 - 0000 = 000000\)
  - How else could we express this language?
  \(\{00\}^*\)
  \(\{00\}000000\*)\]
  \(\{00\}0000(000000)^*\)
  \(\text{etc...}\)

Regular Languages

- The languages that can be described using regular expressions are the regular languages or regular sets
- Not all languages are regular
  - Examples (without proof):
    - The set of palindromes over \(\Sigma\)
    - \((a^k b^n | n > 0)\)
    - \((a^n | a = \text{sequence of } n \ a's)\)
- Almost all programming languages are not regular
  - But aspects of them sometimes are (e.g., identifiers)
  - Regular expressions are commonly used in parsing tools
Ruby Regular Expressions

- Almost all of the features we've seen for Ruby r.e.'s can be reduced to this formal definition
  - /Ruby/ – concatenation of single-character r.e.'s
  - /(Ruby)(Regular)/ – union
  - /(Ruby)^/ – Kleene closure
  - /(Ruby)+/ – same as (Ruby)(Ruby)*
  - /(Ruby)?/ – same as (\((\epsilon)) (\epsilon \text{ is } \epsilon)
  - /[a-z]/ – same as (a|b|c|...|z)
  - /[^0-9]/ – same as (a|b|c|...|z) for a,b,c,... \in \Sigma - \{0..9\}
  - ^, $ – correspond to extra characters in alphabet

Implementing Regular Expressions

- We can implement regular expressions by turning them into a finite automaton
  - A "machine" for recognizing a regular language

Example

- Machine starts in start or initial state
- Repeat until the end of the string is reached:
  - Scan the next symbol s of the string
  - Take transition edge labeled with s
- The string is accepted if the automaton is in a final or accepting state when the end of the string is reached

Example

- What Language is This?
  - All strings over \{0, 1\} that end in 1
  - What is a regular expression for this language?
    - (01)^1

Example

- What Language is This?
  - All strings over \{0, 1\} that end in 1
  - What is a regular expression for this language?
    - (01)^1
Formal Definition

- A deterministic finite automaton (DFA) is a 5-tuple \((\Sigma, Q, q_0, F, \delta)\) where
  - \(\Sigma\) is an alphabet
  - the strings recognized by the DFA are over this set
  - \(Q\) is a nonempty set of states
  - \(q_0 \in Q\) is the start state
  - \(F \subseteq Q\) is the set of final states
  - How many can there be?
  - \(\delta : Q \times \Sigma \to Q\) specifies the DFA’s transitions
  - What’s this definition saying that \(\delta\) is?

More on DFAs

- A finite state automata can have more than one final state:

Our Example, Formally

- \(\Sigma = \{0, 1\}\)
- \(Q = \{S_0, S_1\}\)
- \(q_0 = S_0\)
- \(F = \{S_1\}\)
- \(\delta:\)
  - \(\begin{array}{c|cc}
  \Sigma & 0 & 1 \\
  \hline
  S_0 & S_0 & S_1 \\
  S_1 & S_0 & S_1 \\
  \end{array}\)

Another Example

<table>
<thead>
<tr>
<th>string</th>
<th>state at end</th>
<th>accepts?</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabbcc</td>
<td>S2</td>
<td>Y</td>
</tr>
<tr>
<td>aacc</td>
<td>S2</td>
<td>Y</td>
</tr>
<tr>
<td>bacc</td>
<td>S2</td>
<td>Y</td>
</tr>
<tr>
<td>aabbb</td>
<td>S1</td>
<td>Y</td>
</tr>
<tr>
<td>aa</td>
<td>S0</td>
<td>Y</td>
</tr>
<tr>
<td>(\varepsilon)</td>
<td>S0</td>
<td>Y</td>
</tr>
<tr>
<td>acba</td>
<td>S3</td>
<td>N</td>
</tr>
</tbody>
</table>

(a,b,c notation shorthand for three self loops)

Another Example (cont’d)

What language does this DFA accept? \(a^*b^*c^*\)

Shorthand Notation

- If a transition is omitted, assume it goes to a dead state that is not shown

Language?

Strings over \(\{0,1,2,3\}\) with alternating even and odd digits, beginning with odd digit
What Lang. Does This DFA Accept?

a*b*c* again, so DFAs are not unique

Practice

Give the English descriptions and the DFA or regular expression of the following languages:

• (0|1)(0|1)(0|1)(0|1)(0|1)*
  all strings with length a multiple of 5
• (01)*|(10)*|(01)*0|(10)*1
  all alternating binary strings

Practice

Give the regular expressions and DFAs for the following languages:

• You and your neighbors’ names
• All valid DNA strings (including only ACGT and appearing in multiples of 3)
• All binary strings containing an even length substring of all 1’s
• All binary strings containing exactly two 1’s
• All binary strings that start and end with the same number