CMSC 330: Organization of Programming Languages

Theory of Regular Expressions

Introduction

- 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 regular expressions (REs) really work?
  - Mixture of a very practical tool (string matching with REs) and some nice theory
  - A great computer science result

A Few Questions About REs

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

- What are the basic components of REs?
  - E.g., we saw that e+ is the same as ee*

- How are REs implemented?
  - We’ll see how to build a structure to parse REs

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 $s_1 = \text{super}$ and $s_2 = \text{hero}$, then $s_1s_2 = \text{superhero}$.
  - Sometimes also written $s_1; s_2$.
  - For any string $s$, we have $s \varepsilon = \varepsilon s = s$.
  - You can concatenate strings from different alphabets, then the new alphabet is the union of the originals:
    - If $s_1 = \text{super} \in \Sigma_1 = \{\text{s,u,p,e,r}\}$ and $s_2 = \text{hero} \in \Sigma_2 = \{\text{h,e,r,o}\}$, then $s_1s_2 = \text{superhero} \in \Sigma_3 = \{\text{e,h,o,p,r,s,u}\}$

Definition: Language (cont.)

- Example: The set of strings of length 0 over the alphabet $\Sigma = \{a, b, c\}$
  - $|s| \in \Sigma^*$ and $|s| = 0 \implies \varepsilon \in \Sigma^*$
- Example: The set of all valid Ruby programs
  - Is there a Ruby regular expression for this language? No. Matching (an arbitrary number of) brackets so that they are balanced is impossible. $\{ \{ \ldots \} \}$

- Can REs 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 (cont.)

- Define $L^n$ inductively as
  - $L^0 = \{ \varepsilon \}$
  - $L^n = LL^{n-1}$ for $n > 0$

- In other words,
  - $L^1 = LL^0 = L\{\varepsilon\} = L$
  - $L^2 = LL^1 = LL$
  - $L^3 = LL^2 = LLL$
  - $\ldots$

A language is a set of strings over an alphabet.

- Example: The set of phone numbers over the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 9, (, ), -\}$
- Example: The set of phone numbers over the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 9, (, ), -\}$

- Give an example element of this language
  - 123-456-7890

- Are all strings over the alphabet in the language?
  - No

- 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 $\Sigma$
  - Often written $\Sigma^*$

Can REs represent all possible languages?

- The answer turns out to be no!
- Appropriately, the regular languages $L, L_1, L_2 \subseteq \Sigma^*$

Examples of $L^n$

- Let $L = \{a, b, c\}$
- Then
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{a, b, c\}$
  - $L^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
Operations on Languages (cont.)

- Kleene closure is defined as
  \[ L^* = \bigcup_{i \geq 0} L^i \]

- In other words...
  \[ L^* \] is the language (set of all strings) formed by concatenating together zero or more strings from \( L \).

Definition: Regular Expressions

- Given an alphabet \( \Sigma \), the regular expressions over \( \Sigma \) are defined inductively as

\[
\begin{array}{|c|c|}
\hline
\text{regular expression} & \text{denotes language} \\
\hline
\emptyset & \emptyset \\
\varepsilon & \{ \varepsilon \} \\
\sigma & \{ \sigma \} \\
\hline
\end{array}
\]

Constants

Definition: Regular Expressions (cont.)

- 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 L_B )</td>
</tr>
<tr>
<td>( (A</td>
<td>B) )</td>
</tr>
<tr>
<td>( A^* )</td>
<td>( L_A^* )</td>
</tr>
</tbody>
</table>

Precedence

- Order in which operators are applied
  - In arithmetic
    - Multiplication \( \times \) > addition \( + \)
    - \( 2 \times 3 + 4 = (2 \times 3) + 4 = 10 \)
  - In regular expressions
    - Kleene closure \( * \) > concatenation \( \cdot \) > union \( \mid \)
    - \( a(b)c = (a \cdot (b \mid c)) \)
    - \( a(b)^* \) = \( \{a, ab, abb, ..., \} \)
    - \( a^* \) = \( \{a, a^2, a^3, ..., \} \)
  - Can change order using parentheses \( ( ) \)
    - E.g., \( a(b)c, (ab)^*, (a(b)^*) \)

Example 1

- For a regular expression \( e \), we will write \([e]\) to mean the language denoted by \( e \)
  - \( [a] = \{a\} \)
  - \( [a|b] = \{a, b\} \)
- If \( s \in [RE] \), we say that \( RE \) accepts, describes, or recognizes \( s \)
  - Example: \( aaabbbcccc \) but not \( abcb \)
  - Regexp: \( a^*b^*c^* \)
    - This is a valid regexp because:
      - \( a \) is a regexp \([a]\)
      - \( a^* \) is a regexp \([a]^* = \{\varepsilon, a, aa, ..., \} \)
      - 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?

<table>
<thead>
<tr>
<th>String</th>
<th>Recognized</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabbcc</td>
<td>Yes</td>
</tr>
<tr>
<td>abb</td>
<td>Yes, abb = abb ε, and ε ∈ [c*]</td>
</tr>
<tr>
<td>ac</td>
<td>Yes</td>
</tr>
<tr>
<td>ε</td>
<td>Yes</td>
</tr>
<tr>
<td>aacbc</td>
<td>No</td>
</tr>
<tr>
<td>abcd</td>
<td>No -- outside the language</td>
</tr>
</tbody>
</table>

Example 2

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

Example 3

- All whole numbers containing the substring 330
- Regular expression: (0|1)[0|1|...|9]*330(0|1|...|9)*
- What if we want to get rid of leading 0’s?
- ( (1|...|9)(0|1|...|9)*330(0|1|...|9)* | 330(0|1|...|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...

What Strings are in (10|0)*(10|1)*?

<table>
<thead>
<tr>
<th>String</th>
<th>Recognized</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101000 110111101</td>
<td>Yes</td>
</tr>
<tr>
<td>01010101</td>
<td>Yes</td>
</tr>
<tr>
<td>101</td>
<td>Yes</td>
</tr>
<tr>
<td>0111001</td>
<td>No</td>
</tr>
</tbody>
</table>

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 preceded 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
  - i.e., no 00 may appear after 11

Example 5

- What language does this regular expression recognize?
  - (1| ε )(0|1|...|9) | (2(0|1|2|3))
  - (0|1|...|5)(0|1|...|9)
- All valid times written in 24-hour format
  - 10:17
  - 23:59
  - 0:45
  - 8:30
Two More Examples

- \((00|00|1)^*\)
  - 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
    - \(000000 = 00\cdot00\cdot00 = 00\cdot0000 = 0000\cdot00\)
  - How else could we express this language?
    - \((00)^*\)
    - \((00|0000)^*\)
    - \((00|0000|000000)^*\)
    - 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 \(\Sigma\)
      - reads the same backward or forward
    - \(\{a^n b^n | n > 0\}\) \(a^n = \text{sequence of n a's}\)
- 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 REs can be reduced to this formal definition
  - /Ruby/ – concatenation of single-character REs
  - /{(Ruby)(Regular)}/ – union
  - /{(Ruby)^}/ – Kleene closure
  - /{(Ruby)+}/ – same as \((\text{Ruby})(\text{Ruby})^*\)
  - /{(Ruby)?}/ – same as \((\varepsilon)(\text{Ruby})\) \(\varepsilon\) is \(\varepsilon\)
  - /{[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

Summary

- Languages
  - Sets of strings
  - Operations on languages
- Regular expressions
  - Constants
  - Operators
  - Precedence