CMSC 330: Organization of Programming Languages

Theory of Regular Expressions
Switching gears

• 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 work?
  – Mixture of a very practical tool (string matching) 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?
  – Is there a minimal set of constructs?
  – 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

• First, some definitions...
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)
    • $\emptyset = \{}\}$
    • $\emptyset \neq \{\varepsilon\}$
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 $s_1 \cdot s_2$
  - 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 $s_1 = \text{super} \in \Sigma_1 = \{s,u,p,e,r\}$ and $s_2 = \text{hero} \in \Sigma_2 = \{h,e,r,o\}$, then $s_1s_2 = \text{superhero} \in \Sigma_3 = \{e,h,o,p,r,s,u\}$
Definition: Language

• A language is a set of strings over an alphabet

• Example: The set of all strings over $\Sigma$
  – Often written $\Sigma^*$

• 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}/
Languages (cont’d)

• Example: The set of strings of length 0 over the alphabet $\Sigma = \{a, b, c\}$
  $\{-s | s \in \Sigma^* \text{ and } |s| = 0\} = \{\varepsilon\} \neq \emptyset$

• Example: The set of all valid Ruby programs
  Is there a regular expression for this language?

  No. Matching (an arbitrary number of) brackets so that they are balanced is impossible. { { { … } } }

• 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
  - $L^0 = \{\varepsilon\}$
  - $L^n = LL^{n-1}$ for $n > 0$

• In other words,
  - $L^1 = LL^0 = L\{\varepsilon\} = L$
  - $L^2 = LL^1 = LL$
  - $L^3 = LL^2 =LLL$
  - $...$
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’d)

• Kleene closure is defined as

\[ L^* = \bigcup_{i \in [0..\infty]} L^i \]

• 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>${\varepsilon}$</td>
</tr>
<tr>
<td>each element $\sigma \in \Sigma$</td>
<td>${\sigma}$</td>
</tr>
</tbody>
</table>

- ...
Definition of Regexps (cont’d)

• Let \( A \) and \( B \) be regular expressions denoting languages \( L_A \) and \( L_B \), respectively.

\[
\begin{array}{|c|c|}
\hline
\text{regular expression} & \text{denotes language} \\
\hline
AB & L_A L_B \\
(A|B) & L_A \cup L_B \\
A^* & L_A^* \\
\hline
\end{array}
\]

• There are no other regular expressions over \( \Sigma \).
• We use (\()\)’s as needed for grouping.
Regexp Precedence

• Operation precedence (high to low):
  – Kleene closure: "*"
  – Concatenation
  – Union: "|
  – Grouping: " (" and ")"
The Language Denoted by an r.e.

• 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 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: $aaabbbccc$ but not $abcb$

• Regexp: $a^*b^*c^*$
  – This is a valid regexp because:
    • $a$ is a regexp ($[[a]] = \{a\}$)
    • $a^*$ is a regexp ($[[a^*]] = \{\epsilon, a, aa, ...\}$)
    • 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?

- $aabbcc$
  - Yes; $aa \in [a^*]$, $bbb \in [b^*]$, and $cc \in [c^*]$, so entire string is in $[a^*b^*c^*]$.

- $abb$
  - Yes, $abb = abb\varepsilon$, and $\varepsilon \in [c^*]$

- $ac$
  - Yes

- $\varepsilon$
  - Yes

- $aacbc$
  - No

- $abcd$
  - No -- outside the language
Example 2

• All strings over $\Sigma = \{a, b, c\}$
• 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: \((0|1|...|9)*330(0|1|...|9)*\)
- 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…
Example 4

• What language does this regular expression recognize?

\[
( (1|\varepsilon)(0|1|...|9) \mid (2(0|1|2|3)) ) : (0|1|...|5)(0|1|...|9)
\]

• All valid times written in 24-hour format

- 10:17
- 23:59
- 0:45
- 8:30
Two More Examples

- \((000|00|1)^*\)
  - 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
    * \(000000 = 00\cdot00\cdot00 = 00\cdot0000 = 0000\cdot00\)
  - How else could we express this language?
    * \((00)^*\)
    * \((00|000000)^*\)
    * \((00|0000|000000)^*\)
    * 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
    • $\{a^n b^n \mid n > 0\}$ ($a^n = \text{sequence of } n \text{ a’s}$)

• 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 `(ε|(Ruby))` (// is ε)
  - `/[a-z]/` – same as `(a|b|c|...|z)`
  - `/[^0-9]/` – same as `(a|b|c|...)` for `a,b,c,... ∈ Σ - {0..9}`
  - `^, $` – correspond to extra characters in alphabet
Practice

Give the regular expressions for the following languages:

• 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