CMSC 330: Organization of Programming Languages

Theory of Regular Expressions

Last Lecture

- Ruby language
  - Regular expressions
  - Arrays
  - Code blocks
  - Hash
  - File
  - Exceptions

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 Σ

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

Definition: String

- A string is a finite sequence of symbols from Σ
  - ε is the empty string (“” in Ruby)
  - |s| is the length of string s
    - |Hello| = 5, |ε| = 0
  - Note: Ø is the empty set (with 0 elements); Ø ≠ {ε}

Example strings:
- 0101 ∈ Σ = {0,1} (binary)
- 0101 ∈ Σ = decimal
- 0101 ∈ Σ = alphanumeric

Definition: Concatenation

- Concatenation is indicated by juxtaposition
  - If s₁ = super and s₂ = hero, then s₁s₂ = superhero
  - Sometimes also written s₁·s₂
  - For any string s, we have sε = s = sε
  - You can concatenate strings from different alphabets, then the new alphabet is the union of the originals:
    - If s₁ = super ∈ Σ₁ = {s,u,p,e,r} and s₂ = hero ∈ Σ₂ = {h,e,r,o}, then s₁s₂ = superhero ∈ Σ₃ = {h,e,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 Σ = {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 Σ
  - Often written Σ*
Definition: Language (cont.)

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

- 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

- 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: \(L_1 = \{\text{hi}, \text{bye}\}, L_2 = \{1^*, 2^*\}\)
    - \(L_1L_2 = \{\text{hi}1, \text{hi}2, \text{bye}1, \text{bye}2\}\)

- Union is defined as
  - \(L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}\)
  - Example: \(L_1 = \{\text{hi}, \text{bye}\}, L_2 = \{1^*, 2^*\}\)
    - \(L_1 \cup L_2 = \{\text{hi}, \text{bye}, 1^*, 2^*\}\)

Operations on Languages (cont.)

- Define \(L^n\) inductively as
  - \(L^0 = \{\epsilon\}\)
  - \(L^n = LL^{n-1}\) for \(n > 0\)

- In other words,
  - \(L^1 = LL^0 = L(\epsilon) = L\)
  - \(L^2 = LL^1 = LL\)
  - \(L^3 = LL^2 = LLL\)
  - \(\ldots\)

Examples of \(L^n\)

- Let \(L = \{a, b, c\}\)

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

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

Definition: Regular Expressions

- 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>

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>

Operations

- There are no other regular expressions over \( \Sigma \)

Precedence

- Order in which operators are applied
  
  - In arithmetic
    
    \( \times > + \)
    
    \[ 2 \times 3 + 4 = (2 \times 3) + 4 = 10 \]
  
  - In regular expressions
    
    \( * > \cup > | \)
    
    \[ ab|c = (a b) | c = \{ab, c\} \]
    
    \[ ab^* = a (b^*) = \{a, ab, abb, \ldots\} \]
    
    \[ ab^* = a (b^*) = \{a, \varepsilon, b, bb, bbb, \ldots\} \]
    
  - Can change order using parentheses ( )
    
    E.g., \( a(b|c), (ab)^*, (a|b)^* \)
The Language Denoted by an RE

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 (\([a^*] = \{\varepsilon, a, aa, ...\}\))
  - 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; ( aa \in [a^<em>] ), ( bbb \in [b^</em>] ), and ( cc \in [c^*] ) so entire string is in ([a^*b^<em>c^</em>] )</td>
</tr>
<tr>
<td>abb</td>
<td>Yes, ( abb \in [abv] ), and ( \varepsilon \in [c^*] )</td>
</tr>
<tr>
<td>ac</td>
<td>Yes</td>
</tr>
<tr>
<td>( \varepsilon )</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 \( \Sigma = \{a, b, c\} \)
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: \((01|...9)^*330(01|...9)^*\)
- What if we want to get rid of leading 0's?
- \((1|...9)(0|...9)^*330(0|...9)^* | 330(0|...9)^*\)
- 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
  - i.e., no 00 may appear after 11

Example 5

- What language does this regular expression recognize?
  - \((1|\epsilon)(0|1|...9) | (2(0|1|2|3))\) : \((0|1|...5)(0|1|...9)\)

- 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 00 = 00 00 = 00 00
  - 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
      - \( \{a^n b^n | n > 0 \} \) (\(a^* = \text{sequence of } n \text{ a's}\))
  - 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 REs can be reduced to this formal definition
  - `/Ruby/` – concatenation of single-character REs
  - `/Ruby(Regular)/` – union
  - `/Ruby\*/` – Kleene closure
  - `/Ruby\+/` – same as `(Ruby)(Ruby)*`
  - `/Ruby\?/` – same as `( Ruby )` (\(l\) is \(\epsilon\))
  - `/[a-zA-Z]/` – same as `(a|b|c|...)`
  - `/[^0-9]/` – same as `(a|b|c|...)` 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