To Ponder

What is a language?
Regular Expressions

Lecture 7
Language

- **Definition:** a set of strings

- **Examples**
  - \( \mathcal{L} = \{ \text{cat, dog, fish} \} \)
  - \( \mathcal{L} = \{ \alpha \beta | \alpha \text{ and } \beta \text{ are hex digits} \} \)
  - \( \mathcal{L} = \{ \alpha_1 \alpha_2 \alpha_3 \ldots \alpha_n | n > 0 \land (\forall_{i=1}^{n-1} \alpha_i = \alpha_{i+1}) \} \)

- **Activity:** For each \( \mathcal{L} \) above, find
  - \( |\mathcal{L}| \) (the cardinality of the set)
  - \( \max_{\sigma \in \mathcal{L}} |\sigma| \)
Q: Are C, Java, Ruby, Python, ... languages in this formal sense?
Q: Are C, Java, Ruby, Python, ... languages in this formal sense?

A: Yes!

- \( L_{Ruby} \) is the set of well-formed Ruby programs
- What the interpreter (compiler) accepts
- The syntax of the language

But what does one such string mean?

- The semantics of the language
- Not part of formal definition of "language"
- But necessary to know to claim "I know Ruby"
Regular Expression (RE)

- A formal mechanism for defining a language
  - Precise, unambiguous, well-defined
- In math, a clear distinction between:
  - Characters in strings (the "alphabet")
  - Meta characters used in writing a RE
    - $(a \cup b)^* a(a \cup b)(a \cup b)(a \cup b)$
- In computer applications, there isn't
  - Is '*' a Kleene star or an asterisk?
    - $(a|b)^*a(a|b)(a|b)(a|b)$
Literals

- A *literal* represents a character from the alphabet
- Some are easy:
  - f, i, s, h, ...
- Whitespace is hard (invisible!)
  - \t is a tab (ascii 0x09)
  - \n is a newline (ascii 0x0A)
  - \r is a carriage return (ascii 0x0D)
- So the character '\' needs to be escaped!
  - \\\n  - \ is a \ (ascii 0x5c)
Basic Operators

- ( ) for grouping, | for choice

Examples
- cat|dog|fish
- (h|H)ello
- R(uby|ails)
- (G|g)r(a|e)y

These operators are meta-characters too
- To represent the literal: \\
- (61(3|4))\\

Activity: For each RE above, write out the corresponding language explicitly (ie, as a set of strings)
Character Class

- Set of possible characters
  - (0|1|2|3|4|5|6|7|8|9) is annoying!

- Syntax: [ ]
  - Explicit list as [0123456789]
  - Range as [0-9]

- Negate with ^ at the beginning
  -[^A-Z] a character that is not a capital letter

- Activity: Write the language defined by
  - Gr[ae]y
  - 0[xFF][0-9a-fA-F]
  - [Qq][^u]
Character Class Shorthands

- **Common**
  - \d for digit, ie [0–9]
  - \s for whitespace, ie [\t\r\n]
  - \w for word character, ie [0–9a–zA–Z_]  

- **And negations too**
  - \D, \S, \W (ie [^\d],[^\s],[^\w])
  - Warning: [^\d\s] ≠ [\D\S]

- **POSIX standard (& Ruby) includes**
  - [[:alpha:]] alphabetic character
  - [[:lower:]] lowercase alphabetic character
  - [[:digit:]] decimal digit (unicode! Eg \)`
  - [[:xdigit:]] hexadecimal digit
  - [[:space:]] whitespace including newlines
Wildcards

A . matches any character (almost)
- Includes space, tab, punctuation, etc!
- But does not include newline

So add . to list of meta-characters
- Use \. for a literal period

Examples
- Gr.y
- buckeye\.d

Problem: What is RE for OSU email address for everyone named Smith?
- Answer is not: smith\.d@osu\.edu
Repetition

- Applies to preceding character or ( ) group
  - ? means 0 or 1 time
  - * means 0 or more times (unbounded)
  - + means 1 or more times (unbounded)
  - \{k\} means exactly k times
  - \{a,b\} means k times, for a \leq k \leq b

- More meta-characters to escape!
  - \? \* \+ \{ \}

Computer Science and Engineering  The Ohio State University
Examples

- colour
- smith\. [1-9]\d*@osu\.edu
- 0[xX] (0|[1-9a-fA-F][0-9a-fA-F]*)
- .*\.*\.*\.jpe?g
Your Turn

- (Language consisting of) strings that:
  - Contain only letters, numbers, and _
  - Start with a letter
  - Do not contain 2 consecutive _'s
  - Do not end with _

- Exemplars and counter-exemplars:
  - EOF, 4Temp, Test_Case3, _class,
    a4_Sap_X, S___T_2

- Write the corresponding RE
Your Turn

- (Language consisting of) strings that:
  - Contain only letters, numbers, and _
  - Start with a letter
  - Do not contain 2 consecutive _'s
  - Do not end with _

- Exemplars and counter-exemplars:
  - EOF, 4Temp, Test_Case3, _class, a4_Sap_X, S__T_2

- Write the corresponding RE
Your Turn

- (Language consisting of) strings that:
  - Contain only letters, numbers, and _
  - Start with a letter
  - Do not contain 2 consecutive _'s
  - Do not end with _

- Exemplars and counter-exemplars:
  - EOF, 4Temp, Test_Case3, _class, a4_Sap_X, S__T_2

- Write the corresponding RE
  $[a-zA-Z](_[a-zA-Z0-9]|[^a-zA-Z0-9])*$
Finite State Automota (FSA)

- An FSA is an "accepting rule"
  - Finite set of states
  - Transition function (relation) between states based on next character in string
    - DFA vs NFA
  - Start state ($s_0$)
  - Set of accepting states

- An FSA "accepts" a string if you can start in $s_0$ and end up in an accepting state, consuming 1 character per step
Example

What language is defined by this FSA?
Example

- What language is defined by this FSA?
- A. Binary strings (0's and 1's) with an even number of 0's
Your Turn

- (Language consisting of) strings that:
  - Contain only letters, numbers, and _
  - Start with a letter
  - Do not contain 2 consecutive _'s
  - Do not end with _

- Exemplars and counter-exemplars:
  - EOF, 4Temp, Test_Case3, _class,
    a4_Sap_X, S__T_2

- Write the corresponding FSA
Solution
Solution
Fundamental Results

- Expressive power of RE is the same as FSA
- Expressive power of RE is limited
  - Write a RE for "strings of balanced parens"
    - () (()) (), () (), ((((())))) , ...
    - (((, () ()), , ...
  - Can not be done! (impossibility result)
- Take CSE 3321...
REs in Practice

- REs often used to find a "match"
  - A substring s within a longer string such that s is in the language defined by the RE (CSE|cse)?3901

- Possible uses:
  - Report matching substrings and locations
  - Replace match with something else

- Practical aspects of using REs this way
  - Anchors
  - Greedy vs lazy matching
Anchors

- Used to specify where matching string should be with respect to a line of text
- Newlines are natural breaking points
  - `^` anchors to the beginning of a line
  - `$` anchors to the end of a line
  - Ruby: \A \z for beginning/end of string

Examples

`^Hello World$`
`\A[Tt]he`
`^[^\d].\.jpe?g`
`end\.\z`
Greedy vs Lazy

- Repetition (+ and *) allows multiple matches to begin at same place
  - Example: <.*>
    
    `<h1>Title</h1>
    <h1>Title</h1>`

- The match selected depends on whether the repetition matching is
  - greedy, ie matches as much as possible
  - lazy, ie matches as little as possible

- Default is typically greedy
- For lazy matching, use *? or +?
Regular Expressions in Ruby

- Instance of a class (Regexp)
  
  \texttt{pattern} = \texttt{Regexp.new('^{Rub.')}

- But literal notation is common: \texttt{/pattern/}
  
  \texttt{/[aeiou]*/}

  \texttt{%r{hello+} \#no need to escape /}

- Match operator \texttt{=~} (negated as \texttt{!~})
  
  - Operands: String and Regexp (in either order)
  - Returns index of \texttt{first} match (or nil if not present)
    
    'hello world' \texttt{=~ /o/} \texttt{ #=> 4}
    
    /or/ \texttt{=~ 'hello'} \texttt{ #=> nil}

- Case equality, Regexp \texttt{===} String, \texttt{\rightarrow} Boolean

- Options post-pended: \texttt{/pattern/options}
  
  - i ignore case
  - x ignore whitespace & comments ("free spacing")
Strings and Regular Expressions

- Find all matches as an array
  ```
a.scan /[[[:alpha:]]/]
  ```
- Delimiter for splitting string into array
  ```
a.split /[aeiou]/
  ```
- Substitution: sub and gsub (+/- !)
  - Replace first match vs all ("globally")
  ```
a = "the quick brown fox"
a.sub /[aeiou]/, '@'
  #=> "th@ quick brown fox"
a.gsub /[aeiou]/, '@'
  #=> "th@ q@@ck br@wn f@x"
  ```
Your Turn (Regular Expressions)

- Check if phone number in valid format
  
  ```python
  phone = "614-292-2900"  # not ok
  phone = "(614) 292-2900"  # ok
  ```

  ```python
  format = ?
  if phone ? format  # well-formatted
  ```
Your Turn (Regular Expressions)

- Check if phone number in valid format
  
  ```
  phone = "614-292-2900"  # not ok
  phone = "(614) 292-2900" # ok
  ```

  ```
  format = /\A\(\d{3}\) \d{3}-\d{4}\z/ 
  if phone =~ format  # well-formatted
  ...  
  ```
Summary

- Language: A set of strings
- RE: Defines a language
  - Recipe for making elements of language
- Literals
  - Distinguish characters and metacharacters
- Character classes
  - Represent 1 character in RE
- Repetition
- FSA
  - Expressive power same as RE