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

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
    - $|Hello| = 5$, $|\varepsilon| = 0$
  - Note: $\varnothing$ is the empty set (with 0 elements); $\varnothing \neq \{ \varepsilon \}$
- Example strings:
  - $0101 \in \Sigma = \{0,1\}$ (binary)
  - $0101 \in \Sigma$ = decimal
  - $0101 \in \Sigma$ = alphanumeric

Definition: Concatenation

- *Concatenation* is indicated by juxtaposition
  - If s_1 = super and s_2 = hero, then $s_1 s_2$ = 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 = super $\in \Sigma_1$ = {s,u,p,e,r} and s_2 = hero $\in \Sigma_2$ = {h,e,r,o}, then $s_1 s_2$ = 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 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^*

Languages (cont'd)

- Example: The set of strings of length 0 over the alphabet Σ = {a, b, c}
  - $\{s \mid s \in \Sigma^* \text{ and } |s| = 0\} = \{\varepsilon\} \neq \emptyset$
- 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. $\{\{\{\}\}\}$ or $\{^3\}^3$ or, in general, $\{^n\}^n$

- 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 Σ be an alphabet and let L, L_1, L_2 be languages over Σ
- 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 Σ, the *regular expressions* over Σ are defined inductively as

regular expression	denotes language
∅	∅
ε	{ε}
each element σ ∈ Σ	{σ}

– …

Definition of Regexps (cont'd)

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

regular expression	denotes language
AB	$L_A L_B$
(A|B)	$L_A \cup L_B$
A*	L_A^*

- There are no other regular expressions over Σ
- We use ()'s as needed for grouping

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∈[[re]], we say that re *accepts*, *describes*, or *recognizes* s.

Example 1

- All strings over Σ = {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 ([[a]] = {a})
    - a* is a regexp ([[a*]] = {ε, 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?

aabbbcc

Yes; aa ∈ [[a*]], bbb ∈ [[b*]], and cc ∈ [[c*]], so entire string is in [[a*b*c*]]

abb

Yes, abb = abbε, and ε ∈ [[c*]]

ac

Yes

ε

Yes

aacbc

No

abcd

No — outside the language

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.