# CMSC 330: Organization of Programming Languages

Regular Expressions, Part 3

## 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 turn a r.e. into a program
- Can r.e.'s represent all possible languages?
  - the answer turns out to be no!
  - the languages representable by regular expressions are called, appropriately, the *regular languages*



## Some Definitions

- An *alphabet* is a finite set of symbols
  - usually denoted Σ
- 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
- *Concatenation* is indicated by juxtaposition
  - if $s_1$ = super and $s_2$ = hero, then $s_1s_2$ = superhero
  - sometimes also written $s_1 \cdot s_2$
  - for any string s, we have sε = εs = s



## Languages

- 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
  - Are all strings over the alphabet in the language?
  - Is there a Ruby regular expression for this language?
    - Is the Ruby regular expression over the same alphabet?
- Example: The set of all strings over Σ

## Languages (cont'd)

- Example: The set of all valid Ruby programs
  - Is there a Ruby regular expression for this language?
- Example: The set of strings of length 0 over the alphabet $\Sigma = \{a, b, c\}$
  - $\{s \mid s$ is composed of zero or more symbols of the alphabet, and $|s| = 0\} = \{\varepsilon\} \neq \emptyset$



## 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 $L = \{a, b, cd\}$
- Then
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{a, b, cd\}$
  - $L^2 = \{aa, ab, acd, ba, bb, bcd, cda, cdb, cdcd\}$

## 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 $L$
- Example: for $L = \{a, b\}$

  $L^* = \{\varepsilon, a, b, aa, ab, bb, ba, aaa, \ldots\}$



## Definition of Regular Expressions

- Given an alphabet $\Sigma$, the *regular expressions* over $\Sigma$ are defined as

| regular expression | denotes language |
|---|---|
| $\varnothing$ | $\varnothing$ |
| $\varepsilon$ | $\{\varepsilon\}$ |
| each element $\sigma \in \Sigma$ | $\{\sigma\}$ |

  - ...



## Definition of Regular Expressions, con't.

- Let $A$ and $B$ be regular expressions over $\Sigma$, 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 for $\Sigma$
- We use ()'s as needed for grouping



## Precedence

- Order in which operators are applied
  - In arithmetic
    - Multiplication × > addition +
    - 2 × 3 + 4 = (2 × 3) + 4 = 10
  - In regular expressions
    - Kleene closure * > concatenation > union |
    - ab|c = ( a b ) | c = {"ab", "c"}
    - ab* = a ( b* ) = {"a", "ab", "abb"…}
    - a|b* = a | ( b* ) = {"a", "", "b", "bb", "bbb"…}
  - Can change order using parentheses ( )
    - E.g., a(b|c), (ab)*, (a|b)*

## 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
- Regular expression: a*b*c*
  - this is a valid regular expression for Σ because...
  - a is a regular expression ([[a]] = {a})
  - a* is a regular expression ([[a*]] = {ε, a, aa, ...})
  - similarly for b* and c*
  - so a*b*c* is a regular expression



## Which strings does a*b*c* recognize?

aabbbcc

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

abb

yes, abb = abbε, and ε ∈ [[c*]]

ac

yes

ε

yes

aacbc

no

abcd

no



## Example 2

- All strings over Σ = {a, b, c}
- Regular expression: (a|b|c)*
- Are there other regular expressions that describe 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 rule out numbers with leading 0's?
- ( (1|...|9)(0|1|...|9)*330(0|1|...|9)* | 330(0|1|...|9)* )
- Any other solutions?
- What about all whole numbers not containing the substring 330?
  - We can write an r.e. for this, but it may not be obvious yet how



## Example 4

- What language does (10|0)*(10|1)* denote?
  - (10|0)*
    - 0 may appear anywhere
    - 1 must always be followed by 0
    - so adjacent 1's aren't possible
  - (10|1)*
    - 1 may appear anywhere
    - 0 must always be preceded by 1
    - so adjacent 0's aren't possible
  - put together, all strings of 0's and 1's where every pair of adjacent 0's precedes any pair of adjacent 1's



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

00101000 110111101

first part in [[(10|0)*]]

second part in [[(10|1)*]]

notice that 0010 also in [[(10|0)*]]

But the remainder of the string is not in [[(10|1)*]]

0010101

yes

101

yes

011001

no



## 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

- (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 = 00·00·00 = 00·0000 = 0000·00



## 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 Σ
    - { $a^n b^n$ | n > 0 } ($a^n$ = 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 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}