Regular Expressions

Comp 2400: Fall 2008

Prof. Chris GauthierDickey

What are regular expressions?

- A compact way to specify patterns in text
- A compact way to specify a finite state machine without pictures
- Typically a set of characters and symbols which are expanded to match finite and infinite strings of text

What do they look like?

- Regular characters are part of regular expressions
  - A-Z, a-z, 0-9, plus other symbols
- Some characters are 'meta-characters'
  - . ^ $ * + ? { } [ ] \ | ( )

Simple matches

- A sequence of characters are matched in a regex:
  - abc matches 'abc'
  - if you use a regex for searching or matching in a string, it could match any sequence of abc as a substring:
    - abc would find abc in aaaabcccc at the 4th character position

[]: your first meta-characters

- [] denote a character 'class', or set of characters that can be matched
    - [abc] matches with a, b, or c
    - [a-z] matches any character a to z
    - [0-9] matches any character 0 to 9
    - [] are case senstive and can be combined
        - [A-Za-z] matches a to z regardless of case

- What if you need to match meta-characters inside the character class?
  - By default, they will match on their own
    - [a-z$] will match a-z and $
- Special characters like newline are matched by a backslash
  - \n matches newline, \t matches tab, \r\n matches the end of line on Windows or Mac

Introducing sed

- Now that we're starting with regular expressions, we'd like an easy way to test them out
  - Introducing sed: stream-editor
    - uses regular expressions, among other things, to edit text on the fly using the typical unix I/O model
- sed -E s/[a-zA-Z]/1/g
  - Will replace anything in the character class with 1, try it!

regexs and sed

- Originally, sed only supported basic regular expressions, and +, ? were not supported
    - They could be represented using {1,} and {0,1} respectively
- POSIX.2 defined regular expressions
    - use the -E flag with sed to get full regular expressions

Back to regexs

- The (and) group characters together
- Typically we use grouping with modifiers
  - Modified with +, *, ^, ?, and $
    - + means the regex repeated 1 or more times
    - * means the regex is repeated 0 or more times
    - ^ means the regex begins at the start of the line
    - $ matches the end of line character
    - ? means 0 or 1 of a single character or group

Regexs and the longest sequence

- Matches always occur on the longest sequence:
  - a+ will always match aaaaaa instead of just the first a in aaaaaa (ie, it won't match 6 times)
  - Try sed -E s/a{1,2}/YES/
    - try caaat, and it will return what?
      - cYESaat or cYESat

Examples

- [a-z]+ matches any group of characters with only the letters a-z
    - `sed -E s/[a-z]+/1/g`
- (car)* matches 0 or more cars
- unix(es)? matches unix or unixes
- ^re will match recount, but not Andre
- re$ on the other hand will match Andre

Using { and }

- {n, m} are used for repeating
  - n and m are integers
  - n is the minimum number, m is the maximum number
  - leaving out m means it can repeat any number of times
- {5} means repeat exactly 5 times
- {0,1} means repeat 0 or 1 times
- {1,} means repeat 1 or more times
- {1,5} means repeat 1 to 5 times

Warnings with bounds

- a{3} matches exactly 3 a's: aaa
- a{1,3} matches between 1 and 3 a's:
  - a, aa, aaa
  - But, if you match against aaaa, it will match twice, aaa, and a

More complex regexs

- The bar, '|' lets the regex choose between two patterns
  - a|b means match a or b
  - cat|car means match cat or car
    - How else could you match the above example?
- The . matches any character, but by default doesn't match the end-of-line character
- c.t matches c followed by anything followed by t

The anti-class

- We can match against all characters not in a class by starting with ^
    - [^a-z] matches anything that's NOT a-z
    - sed -E s/[^abc]+/NOABC/g
        - Given abcdef will return: abcNOABC

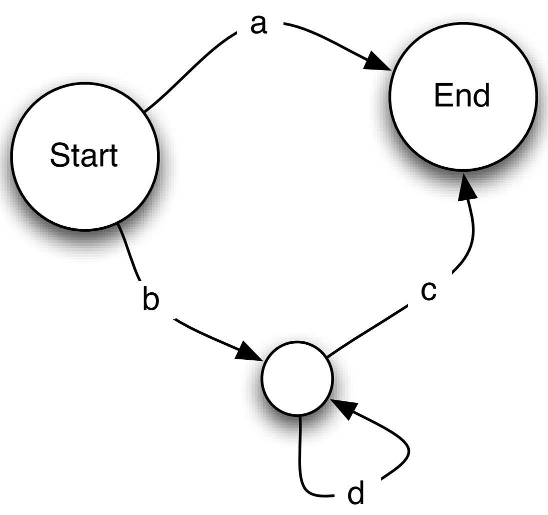
Standard Character Classes

- Any of the following surrounded by [: :]
  - alnum alpha blank cntrl
  - digit graph lower print
  - punct space upper xdigit
    - [:alnum:] in our locale is [0-9A-Za-z]
    - [:alpha:] is [A-Za-z]
    - [:blank:] is [\t]

- [:cntrl:] is any control character
- [:digit:] is [0-9]
- [:graph:] is any printable character, but not space or space-like things
- [:lower:] is [a-z]
- [:print:] is any printable character, including space
- [:punct:] is anything not a space or an [:alnum:]
- [:space:] is [\t\n\v\f\r]
- [:upper:] is [A-Z]
- [:xdigit:] is [0-9A-Fa-f]

Regexs as FSAs

Regex: a | bd*c

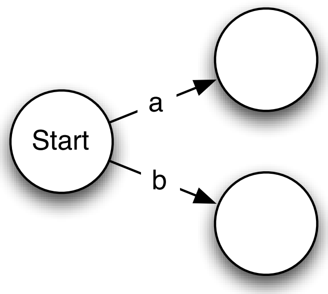


- A regular expression is one way to express Finite State Automata (or machine)
- An FSA can be represented using a regex or a graph

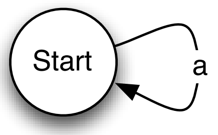
Building blocks of FSAs

- All FSAs can be constructed by two basic building blocks
  - alternation '|'
  - Kleene star '*'
- Q: How can we represent the others?

Regex: a | b



Regex: a*



Questions

- Imagine that you didn't have +, how could you represent it using the other regex constructs?
- Imagine that you didn't have ?, how could you represent it using other regex constructs?