COMP 261 2024 Tri 1
Regular Expressions

Admin

- Test Today 5-6 pm: MCLT101 (A-F), MCLT103 (G-N), KKLT303 (P-Z)
- Do not discuss or post any test questions till Monday 11am

Ambiguous grammar

- Multiple parse trees
- Normally can be solved with right recursion

```
SCRIPT ::=  [ "character" NUM STATS | "room" NAME STATS ]+
STATS  ::=  STAT  |  STATS ";" STATS
STAT   ::=  ID "=" NUM
ID     ::=  matches "#[a-z]+"
NUM    ::=  matches "[1-9][0-9]+"
NAME   ::=  matches "[A-Z][a-z]*“
```

character 5 #ht=2; #age=30; #power=3 room Throne #level=2; #size=90; #cap=40

Regular Languages, Grammars, Expressions

- LL(1) grammars:
  - easy to parse with Recursive Descent parsing, with just one step lookahead
  - allows recursive constructs (with constructs nested inside constructs indefinitely)
- Easy to make more complex grammars and more complex languages
- Are there simpler grammars and simpler languages?
- Yes: "Regular languages" and "regular grammars"
  - Easier to parse
  - Equivalent to finite state automata.
  - Equivalent to regular expressions

Lexical Analysis: Using a Regular Expressions

- Need to separate the text into a sequence of tokens
- Java Scanner can use regular expression patterns to separate the tokens.
  - eg: scan.useDelimiter("\\s*(?=[<])|(?<=[>])\\s*")
    scan.useDelimiter("\\s+|(?=[{}(),;])|(?<=[{}(),;])")
- Alternative approach
  - Define a pattern matching the *tokens* (instead of matching the *separators* between tokens)
  - Still use regular expressions to define the patterns.
- There are tools to make this easier:
  - eg LEX, JFLEX, ANTLR, …
  - see http://en.wikipedia.org/wiki/Lexical_analysis
- We also use regular expression for parsing terminals

Regular Expressions to match patterns in text

- Regular expressions are useful for wide range of text matching tasks
  - lexical analysis
  - editing documents/programs/files, either interactively or scripted
  - cleaning or extracting data for a data base
  - recognising entries on a web page
  - searching in semi-structured data.
- Most large programming languages and shells have regular expressions built-in
- Various different syntax and extensions, but a fairly standard core.
- Regular expressions used in real tools are more complex than regular expressions of formal language theory.
- https://regex101.com/

Language of Regular Expressions

- Pattern built from ordinary characters and special characters (“wild cards”)
  - "pattern: 15" sequence of ordinary characters.
    matches only itself
  - "pattern: [1-9][0-9]" […] contain a set of characters – any character could match (- is a range)
    matches "pattern: 10" or "pattern: 82",
    but not "pattern: 03" or "pattern: 4"
  - "pat+ern:? [1-9][0-9]*" + means 1 or more repetitions;
    * means 0 or more repetitions;
    ? means 0 or 1 repetition (optional)
    matches "patern: 1" or "pattttttern 10034"
    but not "paern: 10"
  - "(pat)+(dog | cat) (..) groups a subpattern for +, *, or ? or for alternatives
    | means "or"
    matches "pat dog" or "pat pat pat cat"

Regular Expressions

- "a (big|red) [bB]all"

- "a (big,)*red bal+"

Extensions

X{n,m}	X, at least n but not more than m times
X{n} X{n,}	X, exactly n times or at least n times
[^….]	matches any character EXCEPT what is in the […]
\d, \s, \w, \W, …	abbreviations for common […]: digit, space, word char, non-word char
^ $	match the boundary between chars at beginning or end of a line
.	matches any character (except new lines)
\b	match the boundary between a word and non-word (either side)
\	quotes the next character

"lookahead/lookbehind"

(?=X) / (?!X)	matches a boundary, if it is followed by X / not followed by X
(?<=X) / (?<!X)	matches a boundary, if it is preceded by X, / not preceded by X

\ also used to quote characters in Java strings => may need double \\

Special characters

- Ordinary characters in regexps match the same character in
- Some characters are "special":

  [ ] | + * ? . ^ $ { } ( ) \

  (note, different libraries vary a bit)

  Special characters mean something in the Regexp language

- Quoting a special character with \ makes it "ordinary"

  \[is the left square bracket (instead of the beginning of a list of alternative characters)

  \\ is a backslash character.

- Quoting an ordinary character may make it special

  \s matches any white space

  \W matches any non-word character

Capturing groups.

- When part of a pattern is surrounded by (…), the text that matches is "captured"
- Later in the pattern you can match the same thing with \1, \2, ….
  - a\s([a-z]+)\s\1\sballoon

  matches

  a big big balloon
  a red red balloon
  a tiny tiny balloon

  but not

  a big red balloon
  a tiny big balloon

Note: this is not part of "formal" regular expressions, but is in many practical regexp libraries
It takes regexps beyond "regular languages"