CMSC 330: Organization of Programming Languages

OCaml Regular Expressions
String Processing in OCaml

- String module provides many useful functions for manipulating strings
  - Concatenate two strings
  - Extract substrings
  - Search for a substring and Replace with something else
String Operations in OCaml

- What if we want to find more complicated patterns? E.g.,
  - Either Steve, Stephen, Steven, Stefan, or Esteve
  - All words that have even number vowels

We need Regular Expressions
Regular Expressions

- A regular expression is a pattern that describes a set of strings. It is useful for
  - Searching and matching
  - Formally describing strings
    - The symbols (lexemes or tokens) that make up a language
- Common to lots of languages and tools
  - Syntax for them in sed, grep, awk, Perl, Python, Ruby, …
    - Popularized (and made fast) as a language feature in Perl
- Based on some elegant theory
  - Future lecture
OCaml Regular Expressions

Multiple Regexp libraries exist:

• **RE**: a pure OCaml regular expressions library that supports several formats (glob, posix, str…)
  • In this lecture, we will use the posix format of the RE library

• **Str**: OCaml comes with the *Str* module.
  • This module is **not** recommended because it is not particularly fast
  • It does not support Unicode
#require "re" (* only needed in Utop *)

# let str2re t = Re.Posix.compile (Re.Posix.re t);

# let r = str2re "[a-z][0-9]+";;

val r : re = <abstr>

# Re.matches r "a12#b22abcd";;
- : string list = ["a12"; "b22"]

A letter followed by one or more digits
Basic Concepts

A regular expression is a pattern that the regular expression engine attempts to match in input text.

A pattern consists of one or more character literals, operators, or constructs.

- “OCaml”: Strings are matched exactly
- “a|b”: A vertical bar separates alternatives. (Boolean Or)
- “ab*”: A quantifier (?, *, +, {n}) after an element (such as a character, or group) specifies how many times the element is allowed to repeat.
- The wildcard . matches any character.
Repetition in Regular Expressions

The following are suffixes on a regular expression $e$

- $e^*$: *zero or more occurrences of $e$*
- $e^+$: *one or more occurrences of $e* so $e^+$ is the same as $ee^*$

- $a^*$: "", "a", "aa", "aaa", ...
- $a^+$: "a", "aa", "aaa", ...
- $bc^*$: "b", "bc", "bcc", ...
- $a+b^*$: "a", "ab", "aa", "aab", "aabb", "aabbb", "aaa", ...
Repetition in Regular Expressions

The following are suffixes on a regular expression $e$

- $e^*$: zero or more occurrences of $e$
- $e^+$: one or more occurrences of $e$
  - so $e^+$ is the same as $ee^*$
- $e?$: exactly zero or one $e$
- $e\{x\}$: exactly $x$ occurrences of $e$
- $e\{x,\}$: at least $x$ occurrences of $e$
- $e\{x,y\}$: at least $x$ and at most $y$ occurrences of $e$
Watch Out for Precedence

- \((\text{OCaml})^*\) means \{"", "OCaml", "OCamlOCaml", ...\}

- \(\text{OCaml}^*\) means \{"OCam", "OCaml", "OCamllll", ...

- Best to use parentheses to disambiguate
  - Note that parentheses have another use, to extract matches, as we’ll see later
Character Classes

- [abcd]
  - {"a", "b", "c", "d"} (Can you write this another way?)

- [a-zA-Z0-9]
  - Any upper- or lower-case letter or digit

- [^0-9]
  - Any character except 0-9 (the ^ means not, and must come first)

- [\t\n ]
  - Tab, newline or space

- [a-zA-Z_\$][a-zA-Z_\$0-9] *
  - Java identifiers ($ escaped...see next slide)
Special Characters

^  beginning of line
$  end of line
\$  just a $

Using $^\text{pattern}$ ensures entire string/line must match pattern.

Languages like Ruby and Python provide more special characters.
Potential Syntax Confusions

- ^
  - Inside regex character class: *not*
  - Outside regex character class: beginning of line

- ( )
  - Inside character classes: literal characters ( )
    - Note /(0..2)/ does not mean 012
  - Outside character classes in regex: used for grouping

- –
  - Inside regex character classes: range (e.g., a to z given by [a-z])
  - Outside regular expressions: subtraction
Summary

Let $re$ represents an arbitrary pattern; then:

- $re$ – matches regexp $re$
- $(re_1|re_2)$ – match either $re_1$ or $re_2$
- $(re)^*$ – match 0 or more occurrences of $re$
- $(re)^+$ – match 1 or more occurrences of $re$
- $(re)^?$ – match 0 or 1 occurrences of $re$
- $(re)^{2}$ – match exactly two occurrences of $re$
- $[a-z]$ – same as (a|b|c|...|z)
- $[^0-9]$ – match any character that is not 0, 1, etc.
- $^, \$ – match start or end of string
Try out regexps at rubular.com
Regular Expression Practice

- Any string containing two consecutive \textit{ab}

- Any string containing \textit{a} or two consecutive \textit{b}
Regular Expression Practice

- Any string containing two consecutive `ab`
  
  `(ab){2}`

- Any string containing `a` or two consecutive `b`
  
  `a|bb`
Regular Expression Practice

Contains *sss* or *ccc*
Regular Expression Practice

Contains sss or ccc

s{3}|c{3}
Regular Expression Practice

Contains exactly 2 b's, not necessarily consecutive.

\[
\begin{array}{c}
\text{beginning} \\
\wedge \\
\text{2 b's}
\end{array} \quad \begin{array}{c}
b \\
\quad b \\
\end{array} \quad \begin{array}{c}
\text{end} \\
\$ \\
\end{array}
\]
Regular Expression Practice

Contains exactly 2 b's, not necessarily consecutive.

```
^[^b]* b[^b]* b[^b]* $
```

beginning

Any character not b

2 b's

end

CMSC 330 Spring 2024
Regular Expression Practice

- Starts with c, followed by one lowercase vowel, and ends with any number of lowercase letters

\(^c[aouei][a-z]*\) $
Regular Expression Practice

- Starts with `c`, followed by one lowercase vowel, and ends with any number of lowercase letters

\(^c\ [aouei] \ [a-z]^* \$

- one vowel
- any number of letters
Regular Expression Practice

- Starts with a and has exactly 0 or 1 letter after that
Regular Expression Practice

- Starts with a and has exactly 0 or 1 letter after that

```regex
^a[A-Ża-z]?$
```
Regular Expression Practice

- Only lowercase letters, in any amount, in alphabetic order
Regular Expression Practice

- Only lowercase letters, in any amount, in alphabetic order

```
^a*b*c*d*e*f*g*h*i*j*k*l*m*n*o*p*r*t*u*v*w*x*y*z*$
```
Regular Expression Practice

- Contains one or more `ab` or `ba`
Contains one or more $ab$ or $ba$

$(ab|ba)^+$
Regular Expression Practice

- Precisely steve, steven, or stephen
Precisely steve, steven, or stephen

\^ste(ve|phen|ven)\$
Regular Expression Practice

- Even length string
Regular Expression Practice

- Even length string

\[^{(\_\_)}]*\$

any two characters
Regular Expression Practice

- Even number of lowercase vowels
Even number of lowercase vowels

\(^([^\text{aouei}]^[\text{aouei}][^\text{aouei}]^[\text{aouei}][^\text{aouei}]*)\)$

Non-vowel\hspace{1cm}vowel
Regular Expression Practice

- Starts with anything but b, followed by one or more a’s and then no other characters
Regular Expression Practice

- Starts with anything but b, followed by one or more a’s and then no other characters

\^[^b]+a+$
Quiz 1

How many different strings could this regex match?

^Hello, Anyone awake?$

A. 1  
B. 2  
C. 4  
D. More than 4
How many different strings could this regex match?

\[ ^{\text{e or nothing}}\text{Hello, Anyone awake?}$$ \]

A. 1
B. 2
C. 4
D. More than 4
Quiz 2

Which regex is not equivalent to the others?

A. ^[cmSc]$  
B. ^c?m?s?c?$  
C. ^(c|m|s|c)$  
D. ^([cm]|[sc])$
Quiz 2

Which regex is not equivalent to the others?

A. `^[cmsc]$`
B. `^c?m?s?c?$`
C. `^(c|m|s|c)$`
D. `^([cm]|[sc])$`

Which regex is not equivalent to the others?
Quiz 3

Which string does not match the regex?

\[[a-z]\{4\}[0-9]\{3\}\]

A. “cmsc\d\d\d”
B. “cmsc330”
C. “hellocmsc330”
D. “cmsc330world”
Quiz 3

Which string does not match the regex?

Recall that without ^ and $, a regex will match any substring

\[ [a-z] \{4\} [0-9] \{3\} \]

A. “cmsg\d\d\d”
B. “cmsg330”
C. “hellocmsg330”
D. “cmsg330world”
RE Library

- Modules
  - Emacs, Glob, Perl, Pcre, Posix, Str
- Basic Functions
  - matches: extracts the matched substring
  - compile: Compile a regular expression into an executable version that can be used to match strings
  - exec: matches str against the compiled expression re, and returns the matched groups if any
  - split: splits s into chunks separated by the regular expression
Example (again)

```ocaml
#require "re" (* only needed in Utop *)
# let str2re t = Re.Posix.compile (Re.Posix.re t);;

#let r = str2re "[a-z][0-9]+";;
   val r : re = <abstr>

# Re.matches r "a12#b22abcd";;
- : string list = ["a12"; "b22"]
```

A letter followed by one or more digits
Extracting Substrings based on Regexps

- Capturing Groups
  - Re remembers which strings matched the parenthesized parts of a Regexp
  - These parts can be referred as Groups
Example: Capturing Groups

```
let r = str2re "^Min:([0-9]+) Max:([0-9]+)$";;
let t = Re.exec r "Min:50 Max:99";;
let min = Re.Group.get t 1;; (* 50 *)
let max = Re.Group.get t 2;; (* 99 *)
```

- **Input**
  - Min: 1 Max: 27
  - Min: 10 Max: 30
  - Min: 11 Max: 30
  - Min: a Max: 24

- **Output**
  - min=1 max=27
  - min=10 max=30

Extra space messes up match
Not a digit; messes up match
What is the output of the following code?

```ocaml
let r = str2re "([A-Z]+)"
let t = Re.exec r "HELP! I’m stuck"
Re.Group.get t 1
```

A. H
B. HELP
C. I
D. I’m stuck
What is the output of the following code?

```javascript
let r = str2re "([A-Z]+)"
Let t = Re.exec r "HELP! I’m stuck"
Re.Group.get t 1
```

A. H
B. HELP
C. I
D. I’m stuck
Quiz 5

What is the output of the following code?

```ml
let r = str2re "[0-9] ([A-Za-z]+).*([0-9])";;
let t = Re.exec r "Why was 6 afraid of 7?";;
Re.Group.get t 2
```

A. afraid
B. 7
C. 6
D. (empty string)
Quiz 5

What is the output of the following code?

```ocaml
let r = str2re "[0-9] ([A-Za-z]+).*([0-9])";;
let t = Re.exec r "Why was 6 afraid of 7?";;
Re.Group.get t 2
```

A. afraid
B. 7
C. 6
D. (empty string)
Re.matches

- extracts all matched substrings as a list

```ocaml
let r = str2re "[A-Za-z]+ \[0-9]+";;
Re.matches r "CMSC 330 Spring 2021";;
# ["CMSC 330", "Spring 2021"]
```

```ocaml
let r = str2re "[A-Za-z0-9]{2}";
Re.matches r "CMSC 330 Spring 2021";;
["CM", "SC", "33", "Sp", "ri", "ng", "20", "21"]
```
What is the output of the following code?

```
let r = str2re "[A-Za-z]{2}";;
Re.matches r "Hello World";;
```

A. ["Hello"; “World”]
B. ["Hello World”]
C. ["He"; "ll"; "Wо"; "rl"]
D. ["He"; "ll"; "о " "Wо"; "rl"; "d" ]
What is the output of the following code?

```
let r = str2re "[A-Za-z]{2}";;
Re.matches r "Hello World";;
```

A. ["Hello"; "World"]
B. ["Hello World"]
C. ["He"; "ll"; "Wo"; "rl"]
D. ["He"; "ll"; "o " "Wo"; "rl"; "d"]
What is the output of the following code?

```
let r = str2re "[A-Za-z]+";;
Re.matches r "To be, or not to be!";;
```

A. ["To","be","or","not","to","be!"]
B. ["To","be","or","not","to","be"]
C. [["To","be"],["or","not"],["to","be!"]]
D. ["to","be!"]
Quiz 7

What is the output of the following code?

```haskell
let r = str2re "[A-Za-z]+";;
Re.matches r "To be, or not to be!";;
```

A. [“To”,“be,”,“or”,“not”,“to”,“be!”]
B. [“To”,“be”,“or”,“not”,“to”,“be”]
C. [ [“To”,“be,”],[“or”,“not”],[“to”,“be!”] ]
D. [“to”,“be!”]