

Regular Expressions – Introduction

Kurt Schmidt

Dept. of Computer Science, Drexel University

April 19, 2017

Intro

Regular Expressions

Regular Expressions – Introduction

Kurt Schmidt

Intro

Syntax

Escaping

Syntactic Sugar

Anchors

Greedy

Epilogue

- Patterns that define a set of strings
 - Called a *regular language*
- *Not* wildcards
 - Similar notion, but, different mechanism
- Used by many utilities:
vi, less, emacs, grep, sed, awk, ed, tr, perl, etc.
 - Note, syntax varies slightly between utilities
- *Very* handy
 - Can be used to test for primality, or play tic-tac-toe
 - But, even simple REs will have you wondering how you got along without them

Finding Strings with `grep`

- `grep` is a handy tool for searching text files
- We shall use `egrep` (`grep -E`)
`egrep regex file(s)`
- If no files are provided, `grep` reads `stdin` (behaves as a filter)

```
$ egrep Waldo *.locations  
...  
$ who | egrep Waldo
```

- You might also use `awk`, search in `vim` or `emacs`, etc.

Syntax

Common Syntax

Regular
Expressions –
Introduction

Kurt Schmidt

Intro

Syntax

Escaping

Syntactic
Sugar

Anchors

Greedy

Epilogue

Primitive Operations (define REs)

- c** Any literal character matches itself
- r^*** Kleene Star – matches 0 or more
- $r_1 r_2$** Concatenation – r_1 followed by r_2
- $r_1 | r_2$** Choice – r_1 *or* r_2
- (r)** Parentheses are used for grouping, to force evaluation
- ** Escape character
 - Turns off special meaning of *metacharacters*
 - Occasionally turns *on* special meaning of ordinary characters

Simple Patterns – Literals

- Literal character – fundamental building block
 - Literal character – fundamental building block
- Using concatenation, a literal string matches itself
- Consider this input file, `input1`:

```
You see my cat  
pass by in your car  
He waves to you
```

```
$ egrep you input1  
pass by in your car  
He waves to you
```

Basic Operations

These 3 operations define regular expressions. Listed in order of increasing precedence.

Given regular expressions R and S , and let $L(X)$ be the set of strings described by the regex X (the language of X):

- Union – $R|S$
 $L(R|S) = L(R) \cup L(S)$
- Concatenation – RS
 $L(RS) = \{rs \mid r \in R, s \in S\}$
- Closure – R^*
 $L(R^*) = \{\epsilon, r, rr, rrr, \dots \mid r \in R\}$

| – Union

- To get any line that contains by or waves:

```
$ egrep 'by|waves' input1
pass by in your car
He waves to you
```

- Note, | is a shell metacharacter
 - Use quotes to keep the shell's grubby hands off of it
- Use parentheses to force evaluation

```
$ egrep '(Y|y)ou' input1
You see my cat
pass by in your car
He waves to you
```

Concatenation

Hopefully, apparent already.

- Accomplished simply by juxtaposition
 - No operators
- E.g., `(a|b)c\.(log|txt)` matches a string that:
 - Starts with a `a` or `b`
 - `,` followed by a `c`
 - `,` followed by a literal `.`
 - `,` ending with either `txt` or `log`
- Note, the period was escaped (explanation follows)

* – Closure

R^* — R , matched 0 or more times

- $*$ modifies the *previous* RE
- It does not match anything on its own
- ab^*c matches ac abc $abbc$ $abbbc$ $abbbbc$...
- $(ab)^*c$ matches c abc $ababc$ $abababc$...

Escaping Metacharacters – \

- Regex metacharacters (`.` `|` `()` `*` `+` `\` etc.) must be escaped to turn off their special meaning, so they can be matched¹
- Note, to match the backslash, it, in turn, must be escaped: `\\`
- Please note, unlike in the shell, quotation marks, single or double, **don't do a thing**, except to match that character
 - I.e., `'` `"` are not special in a regular expression
 - They are not metacharacters
 - They are not used for escaping

¹Depending on the utility, you might escape a metacharacter to turn it on

Vim's Magic, Grep's Extended REs

Regular Expressions – Introduction

Kurt Schmidt

Intro

Syntax
Escaping

Syntactic
Sugar

Anchors

Greedy

Epilogue

- For the sake of convenience, some would-be metacharacters are simply literal characters by default
 - They need to be escaped to turn on their special behavior
- In vim, type `:help magic`, read
- For (e)grep, see the manpages for discussion on “extended regular expressions”¹
- Syntax across utilities will vary a bit
 - You might find `txt2regex` helpful

¹ Do not confuse EREs with Perl REs

Syntactic Sugar

Common Syntax

Syntactic Sugar

. Matches any single character¹

R? Zero or one occurrences of R

R+ One or more occurrences of R

[...] Character class – matches any single character in brackets

[^...] Character class, inverted

Anchors

^ Beginning of line

\$ End of line

\< \> Word anchors

¹In many utilities, *nobody* matches the newline

. – Any Character

- Use the dot `.` to match any single character
 - Many utilities are line-oriented, or built on line-oriented utilities
 - Often, *nobody* matches the newline

```
$ egrep '.ou' input1
You see my cat
pass by in your car
He waves to you
```


[] – Character Classes

Matches any *single* character in the brackets:

- `[brc]at` matches `bat cat rat`
 - *Not* `Bat`
- Careful! `[Y,y]ou` matches `You you ,ou`
- `[ab]*yz` matches `yz ayz byz abyz aayz bbyz bayz`
`abbayz bbbbbbbbbbbbbbbbabbbbbbbbbbbbbbyz ...`
- Very few characters have special meaning inside the brackets:
 - Range, if it's *not* the first character
 - ^ Negation of class, if it *is* the first character

Ranges in Character Classes

- is used to create ranges inside a character class.
 - `0x[0-7]` matches `0x0 0x1 ... 0x3 ... 0x7`
 - *Not* `0x8`
 - `[c1-n]ode` matches `code lode mode node`
 - `[c,l-n]ode` also matches `,ode`
 - `[a-zA-Z]` matches any single letter
 - `[a-Z]` doesn't match anything (if using ASCII)
 - `[A-z]` also matches `[\] ^ _ ``
 - To match the `-` character, place it first:
 - `[-ln]ode` matches `-ode lode node`

^ – Invert Character Class

^ negates the notion of the match, if it appears first.

`[^C]` matches *any* character not in *C*

- `[^rbc]at` matches `hat` `zat` `Cat` `Bat` `sat` ...

- *Not* `rat` `bat` `cat` `at`

- To match the `^` character, put it elsewhere:

- `[r^bc]at` matches `rat` `bat` `cat` `^at`

POSIX Bracketed Expressions

Regular
Expressions –
Introduction

Kurt Schmidt

Intro

Syntax

Escaping

Syntactic
Sugar

Anchors

Greedy

Epilogue

These are widely implemented.
(Note, they, in turn, need to be in brackets.)

`[:alnum:]` `[:alpha:]` `[:ascii:]`

`[:blank:]` `[:cntrl:]` `[:digit:]`

`[:graph:]` `[:lower:]` `[:print:]`

`[:punct:]` `[:space:]` `[:upper:]`

`[:word:]` `[:xdigit:]`

Pre-defined Character Classes

Some classes are so popular, they have nicknames:

`\d` any numeric digit

`\w` word character (alphanumeric or `_`)

`\s` whitespace

These classes are also inverted:

`\D` any character, *not* a digit

`\W` *not* a word character

`\S` *not* whitespace

Anchors

Line Anchors

- They provide context for a regex
- They do *not* match, nor consume, any characters

```
$ egrep '[Yy]ou' input1  
You see my cat  
pass by in your car  
He waves to you
```

- Use the caret (^) to anchor the beginning of a line:

```
$ egrep '^[Yy]ou' input1  
You see my cat
```

- Use the dollar sign (\$) to anchor the end of a line:

```
$ egrep '[Yy]ou$' input1  
He waves to you
```

Word Anchors

Regular
Expressions –
Introduction

Kurt Schmidt

Intro

Syntax

Escaping

Syntactic

Sugar

Anchors

Greedy

Epilogue

- Use `\<` and `\>` to match the beginning/end of a word

```
$ egrep '\<[Yy]ou\>' input1
```

```
You see my cat
```

```
He waves to you
```

```
$ egrep 'our\>' input1
```

```
pass by in your car
```

- Some utilities use `\b` also, or, instead of, for beginning and end word anchors

Greedy

Regular Expressions Are Greedy

Regular Expressions – Introduction

Kurt Schmidt

Intro

Syntax

Escaping

Syntactic Sugar

Anchors

Greedy

Epilogue

- REs are, by default, greedy
- Will match the longest string they can
- Use inverted classes, be a little more thoughtful about your regex

Epilogue

Epilogue

Regular

Expressions –
Introduction

Kurt Schmidt

Intro

Syntax

Escaping

Syntactic
Sugar

Anchors

Greedy

Epilogue

- Various utilities use slightly different flavors
- Some, e.g., treat a particular character as special, while others want them escaped to invoke their special behavior
 - Vim has “magic” and “nomagic”
 - grep distinguishes between regular expressions, and extended regular expressions (egrep)
 - Perl has added extensions (in fact, not regular)
- `man -s7 regex` might be helpful
- Experiment, and RTFM