

Final stuff

Tuesday, May 12, 2009
2:00 PM

Final exam: Thursday 4-6PM

Room: 3117

The final exam cannot be taken late!

Languages on the final: C, Java, Scheme, Prolog, Perl, Python, Ocaml.

No writing of code, but a lot of reading of code.

Know scope rules in C, Java, Scheme, Prolog, Perl, and Python.

Section 10.1 of the book -- read for the final.

All lecture notes and code samples on the 330 webpage.

Regular expressions

Tuesday, May 12, 2009
2:12 PM

Read Section 13.4.1 -- about scope in Python.

Read Section 13.4.2 in the book -- about regular expressions.

regular language: a set of strings that can be defined in terms of

- concatenation
- alternation (|)
- repetition (Kleene closure) (*)

Context-free languages are a set of strings that can be defined in terms of concatenation, alternation, repetition AND RECURSION.

context-free language can be generated from a context-free GRAMMAR

regular language can be generated from
a **regular GRAMMAR**

 example

digit -> 0|1|2|3|4|5|6|7|8|9

nonneg_integer -> digit digit*

Regular expression

Tuesday, May 12, 2009
2:24 PM

"a regular grammar that only consists of one production"

example: aa^*

aa^* can be used to generate an infinite number of strings: a, aa, aaa, aaaa, aaaaa, ...

example: $a|b$
generates a, b

example: $(aa^*)(b|c)$
generates ab, ac, aab, aac, aaab, aaac, ...

A Regular expression is one of the following:

Tuesday, May 12, 2009
2:28 PM

1. A character
2. The empty string ϵ
3. Two regular expressions concatenated
4. Two regular expressions separated by vertical bar or | (alternation)
5. a regular expression followed by a Kleene star or *

offer regular expressions for string matching and manipulation.

Perl, Python, and Ruby offer a rich set of extensions to the notation of regular expressions.

I will introduce these "extended regexps" using Perl.

In Perl, you can describe all regular expressions, and Perl offers additional notation.

1. some extra notation is just abbreviating regular expressions but doesn't increase the power of the notation.

`a+` --> equivalent to `aa*`

$a?$ --> zero or one a , equivalent to $\epsilon \mid a$

2. some extra notation does make "Perl regular expressions" more powerful than standard CS theory "regular expressions."

example: $\backslash 1$, $\backslash 2$, etc. used for "recursion-like" behavior

3. some notation facilitates storage and manipulation of substrings (so you can use results in the rest of your program).

In Perl, Python, Ruby, you MATCH strings to patterns.

Tuesday, May 12, 2009
2:41 PM

In traditional CS theory, we generate strings from regular expressions (regular grammars).

In Perl:

"advanced regexp" is a pattern

A string **matches** a pattern if a substring can be generated from the pattern.

(ab)|c

"ab" match? yes

"c" match? yes

"abd" match? yes

"abc" match? yes

"hic" match? yes

"ddd" match? no

```
$str = "abc";  
if ($str =~ /(ab)|c/) {  
    print "match\n";  
}  
=~ is the MATCH operator.  
regexp in / /.
```

Extensions in Perl "advanced" regexps

Tuesday, May 12, 2009
2:51 PM

? means zero or one

/a(bc)?/ matches with a, abc.

What if I don't want abcabc to match?

Perl has **zero-length assertions**.

^ matches at the beginning of the string

\$ matches at the end of the string

/^a(bc)?\$/

Some additional notation:

. matches any character other than a newline

[a-z] is equivalent to a|b|c|d|e ... y|z

[abcde] is equivalent to a|b|c|d|e

Perl substitution

Tuesday, May 12, 2009
2:57 PM

`s///`

```
$name = "vibha";  
$name = s/bha/jay/;  
print "$name\n";  
# would print out "vijay"
```

What if a pattern can match more than one place within a string?

Perl by default uses the "leftmost longest" rule.

```
string abcabcabcdebc  
pattern /(bc)+/ will match bcabc.
```

Other alternatives:

`*?` means I want the leftmost shortest

match for *

"aardvark"

a.* matches the whole string

aa*

aa*?

Pulling variables out of regexps

Tuesday, May 12, 2009
3:07 PM

string that matches **FIRST** parenthesized subpattern can be referred to in Perl as `\1`

string that matches **second** parenthesized subpattern can be referred to in Perl as `\2`

`\1, \2` -- later on in the pattern itself
`$1, $2` -- variables you can use in later code.