

Warm up:
How would you search through a document for all email addresses inside? (e.g. with control-f)

What if it was a document full of tweets?

@username - - -
- - -
remail@whodunnit.com



Regular Expressions (for real this time)

CSE 311 Autumn 2020
Lecture 20

Midterm Post Mortem

That did not go as I planned.

I estimated most students would be able to finish in 3 hours or so.

From the results of the survey:

25% of students said they took 5 hours or less

50% of students said they took 7 hours or less

75% of students said they took 9 hours or less.

And there was a long tail in those last percentiles. Mean was a little below 8.

Midterm Post Mortem

Why was I off by a factor of ~ 3 ?

Standard procedure is to have a subset of the TAs take a draft of the exam and time themselves. And double or triple the time to estimate how long is fair for an in-person exam.

9 TAs took a version of the exam.

4 took a harder initial version, all finished in 1.5 hours or less

5 took (essentially) the final version, all finished in 55 minutes or less.

Normal math \rightarrow 3 hours should be enough?

Why the normal math didn't work (hypotheses)

TAs had abnormal "unfair advantages" to make them faster:

Problem 1 used predicates with domain restriction built-in

Prior 311 iterations have used these frequently; it was in our review materials, and once or twice in section, but wasn't a focus on the homework.

So the TAs were even more familiar than you were.

Problem 3 was a combination of sets and a "sneak peak" at something coming next week.

I was expecting you to "process something new" they were "processing something old."

I told TAs to take "under test conditions" you weren't under test conditions.

Why the normal math didn't work (hypotheses)

Math built on expectation of students doing significant studying beforehand.

You probably didn't study for an open book exam the way students usually do for a closed book one.

Particularly since we delayed the HW5 deadline so close to the midterm.

In an exam setting, when you don't get a problem you skip it and throw whatever down with 3 minutes left.

In the take-home setting, you might take an extra 2-3 (or 11) hours to get one more good step/a little more progress/etc.

Sooooooooooooooooo...what are we doing about the final?

I won't give you an exam that's much longer than the midterm was
TAs should still be able to finish in an hour-or-so
And I'll tell you the TA time instead of estimating your time.

But we'll extend the window that you're allowed to work in (If a significant portion of the class is going to take 8+ hours, I want you to have that time to do it even if you have other work to do finals week).

Exact window TBD – have to discuss with TAs to make sure we can get everything graded in time for me to turn in grades.

Want to talk to me about grades?

<https://calendly.com/robbieweber/311grade?month=2020-11>

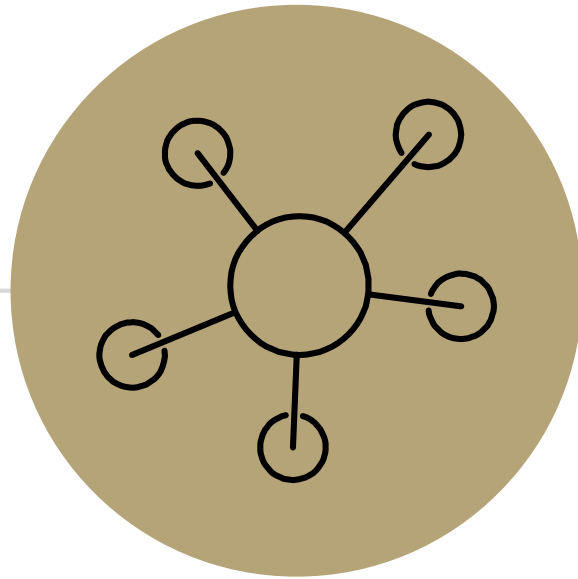
Will take you to a Calendly page.

I have 15 minute slots most of the day tomorrow.

also Monday afternoon

Reserve an appointment there and we can talk tomorrow.

We won't have exam grades yet, but I'll have a sense of how most of you did.



Part 3 of the course!

Course Outline

Symbolic Logic (training wheels; lectures 1-8)

Just make arguments in mechanical ways.

Set Theory/Arithmetic (bike in your backyard; lectures 9-19)

Models of computation (biking in your neighborhood; lectures 19-30)

Still make and communicate rigorous arguments

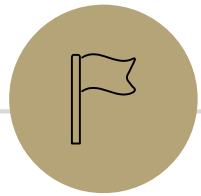
But now with objects you haven't used before.

- A first taste of how we can argue rigorously about computers.

This week: regular expressions and context free grammars – understand these “simpler computers”

After Thanksgiving: what these simple computers can do

Last week of class: what simple computers (and normal ones) can't do.



Regular Expressions

Regular Expressions

I have a giant text document. And I want to find all the email addresses inside. What does an email address look like?

[some letters and numbers] @ [more letters] . [com, net, or edu]

A diagram illustrating an email address pattern. The text "[some letters and numbers] @ [more letters] . [com, net, or edu]" is shown. A blue oval encircles the entire pattern. A red oval encircles the first part "[some letters and numbers]". A blue arrow points from the top of the blue oval down to the "@" symbol. A red arrow points from the bottom of the blue oval up to the "@" symbol. Another blue arrow points from the top of the blue oval down to the "." symbol. A red arrow points from the bottom of the blue oval up to the "." symbol. The words "com, net, or edu" are underlined in red.

We want to ctrl-f for a pattern of strings rather than a single string

Languages

A set of strings is called a language.

Σ^* is a language

“the set of all binary strings of even length” is a language.

“the set of all palindromes” is a language.


“the set of all English words” is a language.

“the set of all strings matching a given pattern” is a language.

Regular Expressions

Every pattern automatically gives you a language .
The set of all strings that match that pattern.

We'll formalize "patterns" via "regular expressions"

- Basis:
- ϵ is a regular expression. The empty string itself matches the pattern (and nothing else does). 
 - a is a regular expression, for any $a \in \Sigma$ (i.e. any character). The character itself matching this pattern.
 - \emptyset is a regular expression. No strings match this pattern.

Regular Expressions

Basis:

ε is a regular expression. The empty string itself matches the pattern (and nothing else does).

\emptyset is a regular expression. No strings match this pattern.

a is a regular expression, for any $a \in \Sigma$ (i.e. any character). The character itself matching this pattern.

Recursive

If A, B are regular expressions then $(A \cup B)$ is a regular expression matched by any string that matches A or that matches B [or both].

If A, B are regular expressions then AB is a regular expression. matched by any string x such that $x = yz$, y matches A and z matches B .

If A is a regular expression, then A^* is a regular expression. matched by any string that can be divided into 0 or more strings that match A .

Regular Expressions

$(a \cup bc)$

$\{a, bc\}$

$(a \cup b \cup \dots \cup z)$

$0(0 \cup 1)1$



$\{001, 011\}$

0^*

$0/0/0$

$\{\epsilon, 0, 00, 000, 0000, 00000, \dots\}$

$(0 \cup 1)^*$

$0/1/0$

$\Sigma = \{0, 1\}$

$\{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

} all binary strings

Regular Expressions

$(a \cup bc)$

Corresponds to $\{a, bc\}$

$0(0 \cup 1)1$

Corresponds to $\{001, 011\}$

all length three strings that start with a 0 and end in a 1.

0^*

Corresponds to $\{\epsilon, 0, 00, 000, 0000, \dots\}$

$(0 \cup 1)^*$

Corresponds to the set of all binary strings.

More Examples

$(0^*1^*)^*$

01010101

all binary strings

$(0 \cup 1)^*$

0^*1^*

00 111

{ ϵ, \dots }

$(0 \cup 1)^*(00 \cup 11)^*(0 \cup 1)^*$

all binary strings

$(00 \cup 11)^*$

{ $\epsilon, 00, 11, 0000, 0011, 1100, 1111, \dots$ }

Fill out the poll everywhere for
Activity Credit!

Go to pollev.com/cse311 and login
with your UW identity
Or text cse311 to 22333

More Examples

$(0^*1^*)^*$

All binary strings

0^*1^*



All binary strings with any 0's coming before any 1's

$(0 \cup 1)^*(00 \cup 11)^*(0 \cup 1)^*$

This is all binary strings again. Not a "good" representation, but valid.

$(00 \cup 11)^*$

All binary strings where 0s and 1s come in pairs

More Practice

You can also go the other way

Write a regular expression for "the set of all binary strings of odd length" ^{even}

$(00|01|10|11)^*(0|1)$

all binary strings of odd length

Write a regular expression for "the set of all binary strings with at most two ones"

$0^*(100)^*0^*|0^*(10)^*0^*|0^*(01)^*0^*|0^*$

Write a regular expression for "strings that don't contain 00"

More Practice

You can also go the other way

Write a regular expression for “the set of all binary strings of odd length”

$(0 \cup 1)(00 \cup 01 \cup 10 \cup 11)^*$

Write a regular expression for “the set of all binary strings with at most two ones”

$0^*(1 \cup \epsilon)0^*(1 \cup \epsilon)0^*$

Write a regular expression for “strings that don’t contain 00”

$(01 \cup 1)^*(0 \cup \epsilon)$ (key idea: all 0s followed by 1 or end of the string)

Practical Advice

Check ϵ and 1 character strings to make sure they're excluded or included (easy to miss those edge cases).

If you can break into pieces, that usually helps.

"nots" are hard (there's no "not" in standard regular expressions)

But you can negate things, usually by negating at a low-level. E.g. to have binary strings without 00, your building blocks are 1's and 0's followed by a 1

$(01 \cup 1)^*(0 \cup \epsilon)$ then make adjustments for edge cases (like ending in 0)

Remember $*$ allows for 0 copies! To say "at least one copy" use AA^* .

Regular Expressions In Practice

EXTREMELY useful. Used to define valid "tokens" (like legal variable names or all known keywords when writing compilers/languages)

Used in `grep` to actually search through documents.

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

`^` start of string

`$` end of string

`[01]` a 0 or a 1

`[0-9]` any single digit

`\.` period `\,` comma `\-` minus

`.` any single character

`ab` a followed by b **(AB)**

`(a|b)` a or b **(A ∪ B)**

`a?` zero or one of a **(A ∪ ε)**

`a*` zero or more of a **A***

`a+` one or more of a **AA***

e.g. `^[\\-+]?[0-9]*(\\.|\\,)?[0-9]+$`

General form of decimal number e.g. 9.12 or -9,8 (Europe)

$\{a \cup b \cup c \cup \dots \cup z\}$

Regular Expressions In Practice

When you only have ASCII characters (say in a programming language)

| usually takes the place of \cup

? (and perhaps creative rewriting) take the place of ε .

E.g. $(0 \cup \varepsilon)(1 \cup 10)^*$ is $0?(1|10)^*$

A Final Vocabulary Note

Not everything can be represented as a regular expression.

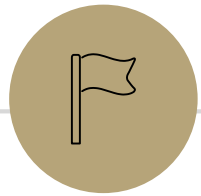
E.g. “the set of all palindromes” is not the language of any regular expression.

Some programming languages define features in their “regexes” that can’t be represented by our definition of regular expressions.

Things like “match this pattern, then have exactly that **substring** appear later.

So before you say “ah, you can’t do that with regular expressions, I learned it in 311!” you should make sure you know whether your language is calling a more powerful object “regular expressions”.

But the more “fancy features” beyond regular expressions you use, the slower the checking algorithms run, (and the harder it is to force the expressions to fit into the framework) so this is still very useful theory.



Context Free Grammars

What Can't Regular Expressions Do?

Some easy things

Things where you could say whether a string matches with just a loop

$\{0^k 1^k : k \geq 0\}$

The set of all palindromes.

And some harder things

Expressions with matched parentheses

Properly formed arithmetic expressions

Context Free Grammars can solve all of these problems!

Context Free Grammars

A context free grammar (CFG) is a finite set of production rules over:

An alphabet Σ of "terminal symbols"

A finite set V of "nonterminal symbols"

A start symbol (one of the elements of V) usually denoted S .

A production rule for a nonterminal $A \in V$ takes the form

$$A \rightarrow w_1 | w_2 | \cdots | w_k$$

Where each $w_i \in (V \cup \Sigma)^*$ is a string of nonterminals and terminals.

Context Free Grammars

We think of context free grammars as **generating** strings.

1. Start from the start symbol S .
2. Choose a nonterminal in the string, and a production rule $A \rightarrow w_1 | w_2 | \dots | w_k$ replace that copy of the nonterminal with w_i .
3. If no nonterminals remain, you're done! Otherwise, goto step 2.

A string is in the language of the CFG iff it can be generated starting from S .

Examples

$$S \rightarrow 0S0|1S1|0|1|\varepsilon$$

$$S \rightarrow 0S|S1|\varepsilon$$

$$S \rightarrow (S)|SS|\varepsilon$$