WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regular Expressions

CSE 311 Autumn 2023
Lecture 20

Announcements

You'll get an email tonight or tomorrow morning with which room to go to for the exam.

Please bring an ID (Husky Card or other ID) to the exam.

- We'll be checking those during the exam.

Remember you're allowed one piece of paper of **handwritten** notes.

- Please read details on the exams page.

Announcements

Some schedule changes:

CC20 (comes out today) is due Friday. (No CC for Wednesday's lecture)

HW6 will come out on Wednesday Nov 15. Officially due Wed. Nov 22,

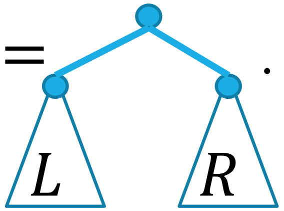
But we'll consider submissions through Friday Nov 24 as on time; one late day to submit Saturday Nov 25, two for Sunday, three for Monday.

Staff are taking Thanksgiving off. We won't be there to help on Ed; if you need a late day, we don't want that to ruin Thanksgiving.

HW7 will be released Wed Nov 22. We think you can reasonably start it on Monday Nov 27 and be fine. But if you'd prefer to work (say, you have a long plane flight) it'll be out. Due Friday Dec 1.

Structural Induction on Binary Trees (cont.)

Let $P(T)$ be "size$(T) \leq 2^{height(T)+1} - 1$". We show $P(T)$ for all binary trees T by structural induction.

$T =$  .

height(T)=1 $+ \max\{height(L), height(R)\}$

size(T)= 1 +size(L)+size(R)

size(T)=1+size(L)+size(R) $\leq 1 + 2^{height(L)+1} - 1$ $+2^{height(R)+1}$ -1 (by IH)

$\leq 2^{height(L)+1}$ $+2^{height(R)+1}$ -1 (cancel 1's)

$\leq 2^{height(T)} + 2^{height(T)} - 1 = 2^{height(T)+1} - 1$ (T taller than subtrees)

So $P(T)$ holds, and we have $P(T)$ for all binary trees T by the principle of induction.

Structural Induction on Trees

The common pattern for the inductive step will be something like:

Use the recursive definitions to talk about the sub-trees instead of the full tree.

Use the IH to talk about the target (the conclusion or the RHS of an equation) holding on the sub-trees.

Recombine the conclude the target for the full tree.

What does the inductive step look like?

Here’s a recursively-defined set:

Basis: $0 \in T$ and $5 \in T$

Recursive: If $x, y \in T$ then $x + y \in T$ and $x - y \in T$.

Let $P(x)$ be “$5|x$”

What does the inductive step look like?

Well there’s two recursive rules, so we have two things to show

Just the IS (you still need the other steps)

Let z be an arbitrary element of T not covered by the base cases. By the exclusion rule, $\mathrm{z} = x + y$ or $\mathrm{z} = x - y$ for some $x, y \in T$.

Inductive hypothesis: Suppose $P(x)$ and $P(y)$ hold

Inductive Step

Case 1: $z = x + y$

By IH $5|x$ and $5|y$ so $5a = x$ and $5b = y$ for integers a, b.

Adding, we get $x + y = 5a + 5b = 5(a + b)$. Since a, b are integers, so is $a + b$, and $P(x + y)$ holds.

Case 2: $z = x - y$

By IH $5|x$ and $5|y$ so $5a = x$ and $5b = y$ for integers a, b.

Subtracting, we get $x - y = 5a - 5b = 5(a - b)$. Since a, b are integers, so is $a - b$, and $P(x - y)$ holds.

If you don’t have a recursively-defined set

You won’t do structural induction.

You can do weak or strong induction though.

For example, Let $P(n)$ be “for all elements of S of “size” n <something> is true”

To prove “for all $x \in S$ of size n…” you need to start with “let x be an arbitrary element of size $k + 1$ in your IS.

You CAN’T start with size k and “build up” to an arbitrary element of size $k + 1$ it isn’t arbitrary.

Part 3 of the course!

Course Outline

Symbolic Logic (training wheels)

Just make arguments in mechanical ways.

Set Theory/Number Theory (bike in your backyard)

Models of computation (biking in your neighborhood)

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"

Soon: what these simple computers can do
Then: what simple computers can't do.

Last week: A problem our computers cannot solve.

The next two weeks

A big of a grab-bag—topics that don’t build on each other much.

Some stuff that’s has applications in future classes.

This week: some theory that’s useful for computer scientists

- The topics for this weeks are key parts of building compilers, we’ll use that as motivation—no details on compilers, but hopefully enough that you’ll find it interesting.
- They ALSO can be thought of as “underpowered computers.”
- We’ll use them to build up to proving what computers can and can’t do.

Next week: some mathematical tools that are useful in the future.

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]

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

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

$(a \cup bc)$

$0(0 \cup 1)1$

0^*

$(0 \cup 1)^*$

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 $\{\varepsilon, 0,00,000,0000, \ldots\}$

$(0 \cup 1)^*$

Corresponds to the set of all binary strings.

More Examples

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

0^*1^*

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

$(00 \cup 11)^*$

More Examples

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

All binary strings

0^*1^*

All binary strings with any 0's coming before all 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”

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

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

$(01 \cup 1)^*(0 \cup \varepsilon)$ 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)
```

Regular Expressions In Practice

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

| usually takes the place of ∪

? (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.

A bit more on induction

If there's time

🚨 CAUTION 🚨

When you don’t have a recursive definition, you can’t use structural induction! But you can do an inductive proof that looks like structural induction.

The way to do the proof is to define a predicate with a for-all in it

Let $P(n)$ be “for all <things> of <size> n, the claim holds”

Then in doing the inductive step you start with an arbitrary thing of size $k+1$. Start with the BIG thing, then find the small thing inside to apply the IH.

Induction: Hats!

You have n people in a line ($n \geq 2$). Each of them wears either a purple hat or a gold hat. The person at the front of the line wears a purple hat. The person at the back of the line wears a gold hat.

Show that for every arrangement of the line satisfying the rule above, there is a person with a purple hat next to someone with a gold hat.

Yes this is kinda obvious. I promise this is good induction practice.

Yes you could argue this by contradiction. I promise this is good induction practice.

Induction: Hats!

Define $P(n)$ to be "in every line of n people with gold and purple hats, with a purple hat at one end and a gold hat at the other, there is a person with a purple hat next to someone with a gold hat"

We show $P(n)$ for all integers $n \geq 2$ by induction on n.

Base Case: $n = 2$

Inductive Hypothesis:

Inductive Step:

By the principle of induction, we have $P(n)$ for all $n \geq 2$

Induction: Hats!

Define $P(n)$ to be "in every line of n people with gold and purple hats, with a purple hat at one end and a gold hat at the other, there is a person with a purple hat next to someone with a gold hat"

We show $P(n)$ for all integers $n \geq 2$ by induction on n.

Base Case: $n = 2$ The line must be just a person with a purple hat and a person with a gold hat, who are next to each other.

Inductive Hypothesis: Suppose $P(k)$ holds for an arbitrary $k \geq 2$.

Inductive Step: Consider an arbitrary line with $k + 1$ people in purple and gold hats, with a gold hat at one end and a purple hat at the other.

We start with an arbitrary BIG thing—start with an arbitrary line of size $k + 1$.

Target: there is someone in a purple hat next to someone in a gold hat.

By the principle of induction, we have $P(n)$ for all $n \geq 2$

Induction: Hats!

Define $P(n)$ to be "in every line of n people with gold and purple hats, with a purple hat at one end and a gold hat at the other, there is a person with a purple hat next to someone with a gold hat"

We show $P(n)$ for all integers $n \geq 2$ by induction on n.

Base Case: $n = 2$ The line must be just a person with a purple hat and a person with a gold hat, who are next to each other.

Inductive Hypothesis: Suppose $P(k)$ holds for an arbitrary $k \geq 2$.

Inductive Step: Consider an arbitrary line with $k + 1$ people in purple and gold hats, with a gold hat at one end and a purple hat at the other.

Case 1: There is someone with a purple hat next to the person in the gold hat at one end. Then those people are the required adjacent opposite hats.

Case 2:. There is a person with a gold hat next to the person in the gold hat at the end. Then the line from the second person to the end is length k, has a gold hat at one end and a purple hat at the other. Applying the inductive hypothesis, there is an adjacent, opposite-hat wearing pair.

In either case we have $P(k + 1)$.

By the principle of induction, we have $P(n)$ for all $n \geq 2$