HW0 Recap
HW0 Recap

• Read from stdin, write to stdout:
  • Python: `sys.stdin`, `print`
  • C++ `cin`, `cout`
  • Java: `System.in`, `System.out`
    • `Scanner scanner = new Scanner(System.in).readLine();` (drops newlines)

• Power set of the empty set?
  • The power set of a set $S$ is the set of all possible subsets of $S$
  • Including empty set, and $S$ itself
  • `{{}}`

• XML parsing:
  • Java: `javax.xml.parsers`
  • Python: `xml.etree.ElementTree: parse and findall`
  • C++: `pugixml`
  • Regular expressions???
RegEx match open tags except XHTML self-closing

I need to match all of these opening tags:

```html
<p>
<a href="foo">
</a>
</p>
```

But not these:

```html
<br />
<hr class="foo" />
```

You can't parse [X]HTML with regex. Because HTML can't be parsed. Regex is not a tool that can be used to correctly parse HTML. As I have explained before, regular expressions are not equipped to break down HTML into its meaningful parts but it is not getting to me. Even enhanced irregular regular expressions are not up to the task of parsing HTML. You will never be able to parse [X]HTML with regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The `<center>` cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow it is too late it is too late we cannot be saved the trepassing of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied dear lord help us how can anyone survive this scourge using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes using regex as a tool to process HTML establishes a breach between this world and the dread realm of corrupt entities (like SGML entities, but more corrupt) a mere glimpse of the world of regex parsers for HTML will instantly transport a programmer's consciousness into a world of ceaseless screaming, he comes, the pestilent slithy regex-infection will devour your HTML parser, application and existence for all time like Visual Basic only worse he comes he does not fight he comes he comes he comes the unholy radiance destroying all enlightenment. HTML tags leaking from your eyes like liquid pain, the song of regular expression parsing will extinguish the voices of mortal man from the sphere I can see it can you see it it is beautiful the final snuffing of the lies of Man ALL IS LOST ALL IS LOST the bony he comes he comes he comes the bony he comes he comes the bony he comes he comes the bony he comes he comes the bony he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes his god he comes
HW1

• Will include core set of pkgs
  • python3
  • default-jdk
  • build-essential
  • nodejs

• Any other libraries/tools:
  • manually install using in Makefile `setup`

• Review slides about math vs representation in code
In-class example (from last time)

• Design machine $M$ that recognizes: $\{w \mid w \text{ has exactly three 1’s}\}$

• Where $\Sigma = \{0, 1\}$,

• Remember:

**Definition 1.5**

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the **states**,
2. $\Sigma$ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states**.
Proving that a language is regular

- Prove that this lang is regular: \( \{ w \mid w \text{ has exactly three 1's} \} \)

A language is called a regular language if some finite automaton recognizes it.
Operations on Regular Languages
Password Requirements

» Passwords must have a minimum length of ten (10) characters - but more is better!
» Passwords **must include at least 3** different types of characters:
  » upper-case letters (A-Z)
  » lower-case letters (a-z)
  » symbols or special characters (%, &, *, $, etc.)
  » numbers (0-9)
» Passwords cannot contain all or part of your email address
» Passwords cannot be re-used
Password checker

M5: AND

M3: OR

M1: Check special chars

M2: Check uppercase

M4: Check length

Want to be able to easily combine finite automata machines

To keep combining operations must be **closed**!
“Closed” Operations

• Natural numbers = \( \{0, 1, 2, \ldots\} \)
  • Closed under addition: if \( x \) and \( y \) are Natural, then \( z = x + y \) is a Nat
  • Closed under multiplication?
    • yes
  • Closed under subtraction?
    • no

• Integers = \( \{\ldots, -2, -1, 0, 1, 2, \ldots\} \)
  • Closed under addition and multiplication
  • Closed under subtraction?
    • yes
  • Closed under division?
    • no

• Rational numbers = \( \{x \mid x = y/z, y \text{ and } z \text{ are ints}\} \)
  • Closed under division?
    • No?
    • Yes if \( z \neq 0 \)

Any set is **closed** under some operation if applying that operation to members of the set returns an object still in the set.
Why Closed Operations on RegularLanguages?

• Closed operations preserves “regularness”

• I.e., it preserves the same computation model

• So result of combining machines can be combined again
Password checker: “Or” = “Union”

M3: OR

M1: Check special chars

M2: Check uppercase
A Closed Operation: Union

**Theorem 1.25**

The class of regular languages is closed under the union operation.

In other words, if \( A_1 \) and \( A_2 \) are regular languages, so is \( A_1 \cup A_2 \).

- How do we prove that a language is regular?
  - Create a FSM recognizing it!
- Create machine combining machines recognizing \( A_1 \) and \( A_2 \).
Kinds of Mathematical Proof

• Proof by construction
  • Construct the mathematical object in question

• Proof by contradiction

• Proof by induction
Union Closed?

**Theorem 1.25**

The class of regular languages is closed under the union operation.

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

**Proof** (implement for hw1)

- Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize $A_1$,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize $A_2$,
- Construct a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using $M_1$ and $M_2$
- $M$ runs its input on both $M_1$ and $M_2$ in parallel, accept if either accepts
- states of $M$: $Q = \{(r_1, r_2) | r_1 \in Q_1$ and $r_2 \in Q_2\}$. This set is the *Cartesian product* of sets $Q_1$ and $Q_2$
- $M$’s transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $M$ start state: $(q_1, q_2)$
- $M$ accept states: $F = \{(r_1, r_2) | r_1 \in F_1$ or $r_2 \in F_2\}$
Another Operation: Concatenation

**Theorem 1.26**

The class of regular languages is closed under the concatenation operation.

In other words, if \( A_1 \) and \( A_2 \) are regular languages then so is \( A_1 \circ A_2 \).

- Can’t directly combine \( A_1 \) and \( A_2 \)
  - don’t know when to switch from \( A_1 \) to \( A_2 \) (can only read input once)
- It would create a new kind of machine!
- So is concatenation not closed???
Let $N_1$ recognize $A_1$, and $N_2$ recognize $A_2$.

**Want:** Construction of $N$ to recognize $A_1 \circ A_2$.

$\epsilon = \text{empty string} = \text{no input}$

So $N$ can:
- stay in current state **and**
- move to next state

$N$ is a new kind of machine, an NFA!
Check-in Quiz 2

On gradescope
End of Class Survey

See course website
HW1