Week Three, Lecture One

Recap

Finite state languages are all languages accepted by a finite state machine. 🤖

- FSM = DFA or NFA

DFAs are a very concrete model of patterns. NFAs are a very abstract model. Is there something in between?

- Is there a "pure pattern" model?
- Yes, **regular expressions**.

Regular expressions are declarative, "pure patterns," not in any way imperative.

- They do not tell you *how* like a DFA

But in fact, regular expressions are exactly equivalent to both DFAs and NFAs (which are also equivalent).

$$\text{RegEx} = \text{DFA} = \text{NFA}$$

Regular Expressions 🦄:cat:🌟

With just three operations, you can represent everything you can with an FSM:

1. Union 🦄
2. Concatenation 🐱
3. Kleene star 🌟

Syntax	**Semantics**
a	{a}
ε	{ε}
Ø	{}
R ∪ S	"Set of strings denoted by R" ∪ "Set of string denoted by S"
R ○ S	"Set of strings denoted by R" ○ "Set of string denoted by S"
R^*	$(\text{Set denoted by R})^*$

Suppose we have an alphabet {0,1} a language L = {all even length strings}. How would we represent this with a RegEx?

$$((0 \cup 1) \circ (0 \cup 1))^{\backslash *}$$

Suppose we have another language L = {no two 1's adjacent}.

$$0^{\backslash *}(100^{\backslash *})^{\backslash *} \circ (1 \cup \varepsilon)$$

We need the term "$\circ$ $(1 \cup \varepsilon)$" to account for "01".

Context-Free Grammars

$$G := (V, \Sigma, R, S)$$

- V = finite set of variables
  - usually represented as capital letters
- Σ - finite set of terminals/symbols
  - same thing as alphabet
- R = finite set of rewriting rules
  - of the type $V \rightarrow (V \cup \Sigma)^*$
- S = start variable

In terms of operator precedence, Kleene-star > concatenation > union.

- This also corresponds to our intuition of concatenation as multiplication and union as addition (and kinda Kleene-star as exponentation?).