Announcements:

- We tried to capture learning goals from HW1, webpage/tips
- Hope that those who followed our tips did well!
- If you did → but didn't → talk to us! We'll help. (also: optional reassessment)

- Good luck on Quiz 1!
- HW1 graded, solutions published.
- To be announced very soon (email through edstem):
  - checkup 3 reopened
  - checkup 4
  - HW 2
  - HW 1 / Grady feedback request
- Quiz 2: Oct 22 in class
  - will cover NFAs + Reg Exp
    - you can use anything we learned upto today (=regular languages)
    - we'll have Handout 2
  - Final during registrar's announced time → mid Oct

Last time: Defined regexp + their languages, proved they characterize regular languages.

Recap:

Def: R is a regular expression over alphabet Σ if:

base cases:
- $R = \emptyset$
- $R = \varepsilon$
- $R = a$ for some $a \in \Sigma$

inductive cases:
- $R = (R_1 \cup R_2)$ for R_1, R_2 that are regular expressions
- $R = (R_1 \circ R_2)$ for R_1, R_2 that are regular expressions
- $R = (R)^*$ for R that's a regular expression

Def: For a regular expression R we define $L(R)$ the language associated with R (aka generated by R), as follows:

- $L(\emptyset) = \{\}$ $(= \emptyset)$
- $L(\varepsilon) = \{\varepsilon\}$
- $L(a) = \{a\}$ $\forall a \in \Sigma$
- $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$
- $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$
- $L((R)^*) = L(R)^*$

Theorem: A language L is generated by a reg exp $\iff$ L is regular

Wrapping up regular languages

L is regular $\iff$ $\exists$ DFA recognizing L

$\iff$ $\sim_L$ has finitely many equivalence classes

$\iff$ $\exists$ NFA recognizing L

$\iff$ $\exists$ regexp generating L

Aside: Comments about efficiency:

- DFA → NFA ------------> same (linear) # states
- NFA → DFA ---------> exponential blowup in # states
- Reg Exp → NFA ---------> # states in NFA is linear in length of R.E.
- NFA → (GNFA) → Reg Exp -----> length of Reg.exp exponential in # states of NFA

- DFA can be efficiently minimized (in P)
- Regexp can also be minimized, but we don't know whether it can be done efficiently (probably not, it's "PSPACE Complete")

→ we'll discuss efficiency later in class

Applications of Regular languages:

- pattern matching / text processing (grep, awk, regexp in Python)
- Networking: modeling communication protocols (allowed message sequences)
- Compilers: recognizing tokens (identifiers, keywords, operators, ...)
- streaming algorithms
- Hardware
- NLP/speech: finite state models for word formation / pronunciation

These applications often use all 3 models: RegExp, NFA, DFA
(sometimes use extensions: DFA with output beyond accept/reject)

For us (in this class) also pedagogical tool:

- Example of working with models of computation
- introduction of several important ideas:
  - reasoning which languages are/are not recognizable in some model of computation
  - non-determinism
  - ⋮

But your laptop can do much more than DFA...

What next?

- skipping context-free languages
  → interesting stuff!
  → very useful for compilers + NLP
  → see book if interested

- Recall in Lecture 1 we asked:

which questions are easy / hard / impossible to solve?

- we answered what's possible for DFA
- next unit: what's possible for any computer?

(=which languages are computable) → "computability"

- last unit: what is computable efficiently?
  → intro to "complexity theory"

Computability

Our next model of computation: Turing Machine (TM).

What was missing in DFA that your laptop has?

- **memory**: DFA has a fixed amount of memory, can't, e.g., implement a Counter
  - since to count to an unbounded number, need unbounded memory.
  - ⇝ eg. $\{a^n b^n \mid n \geq 0\}$ seems easy to recognize if we could implement a counter
- **read left to right** (can't go back & forth)
  - eg, $\{w \mid w$ is a palindrome$\}$ seems easy to recognize if we could go back & forth
- **read-only** (no writing allowed) ⇝ eg seems like we need writing (in addition to memory) to implement a Counter

→ TM model will address all these

- unbounded memory
- read+write access
- move right+left

→ will define next time!

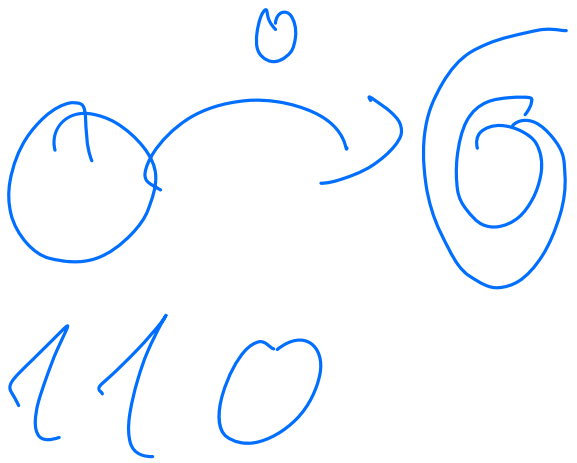
Other things brought up in class discussions:

- longer output (rather than accept/reject)?

- efficiency/time:
  - what about random access?
    (eg, go directly to last symbol?
    to middle symbol?)
  - or more complex memory structure to support efficiency? (eg, caching...)

TM model will not directly address these.
However, all these turn out to not matter
if you only care about what's
possible/impossible to solve. ← important question we are focusing on!

- Last unit of class will focus on efficiency ("easy vs hard") where these can be addressed!



0110

$(01^*) \cup 1) \circ \varepsilon$

$\{0, 01, 011, 0111, 1, \ldots\}$