Announcements:

* HW 1 graded, solution published

* Very soon:
  - checkup 3 reopens
  - checkup 4
  - HW 2
  - HW 1 regrade requests ~> try to come to OHs
  - Feedback

* Quiz 2: Oct 22nd in class
  - we will cover NFAs + Regexp
  - you can use anything we learned up to today
  - We'll have handout 2

* Last time :

(1) We defined regexp + their languages

(2) Regular expression characterize regular languages.

L is regular $\iff$ $\exists$ regexp R with $L(R) = L$

$R = \phi$

$R = \varepsilon$

$R = a$, $a \in \Sigma$

} base cases

$R = (R_1 \cup R_2)$

$R = R_1 \circ R_2$

$R = R_1^*$

} inductive cases

Some quick examples:

$((0 \cup 1)(0 \cup 1))^*$ $L = ?$ $L = \{x \in \{0,1\}^* \mid |x| \text{ is even}\}$

$(0 \cup 1)(0 \cup 1)$ gives you 00, 01, 10, 00

$((0 \cup 1)^* 00) \cup (11 (0 \cup 1)^*)$ $L = ?$

$(0 \cup 1)^* 00$: ends in 00

$11 (0 \cup 1)^*$: starts in 11

$L = \{w \mid w \text{ ends in } 00 \text{ or } w \text{ starts in } 11\}$

Wrapping up regular languages:

L is regular $\overset{\text{def}}{\iff} \exists$ DFA recognizing L

$\overset{MN}{\iff} \sim_L$ has a finite # of equivalence classes

$\iff \exists$ a NFA recognizing L

$\iff \exists$ regexp generating L

Aside: efficiency

=> DFA $\longrightarrow$ NFA : that's efficient (same # of states)

=> NFA $\longrightarrow$ DFA: not efficient NFA with k states transform into DFA with 2^k states

=> Reg Exp → NFA : efficient : # of states in NFA is linear in length of the regexp

=> NFA → Regexp: length of reg.exp exponential in # of states of NFA

* DFAs can be efficiently minimized

* Regexp can be minimized

↳ no known efficient algos: "PSPACE COMPLETE"

(If rumors are true, no efficient algorithm)

Applications of regular languages:

- pattern matching / text processing (grep, regexp in python)
- Networks: modelling communication protocols
  (what message sequences are allowed?)

- Compilers: recognizing keywords, operators...

- Streaming algorithms: you get data sequentially
  & must process it on the fly

- NLP: recognizing words in a text

Applications often use all 3 models: DFAs / NFA / regexp.

Why did we study them in the context of this course?

- Example of working with a simple model 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 DFAs: so what's next?

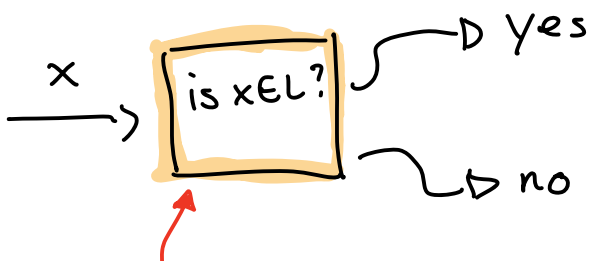
- we'll skip context-free languages
  - → it's cool
  - → very useful for compilers + NLP
  - → see book if interested

In lecture 1 we asked:

which questions are easy / hard / impossible to solve?

* Now you know what is possible for DFAs
* next: what's possible for any computer?

  ⇒ which languages are "computable"?



before: a DFA / NFA

next: a computer!

* last unit: what is computable efficiently?

→ intro to "complexity theory"



how many steps of computation does it take the computer?

* What does a computer have that a DFA doesn't have?

* What are things your computer can do but a DFA can't?

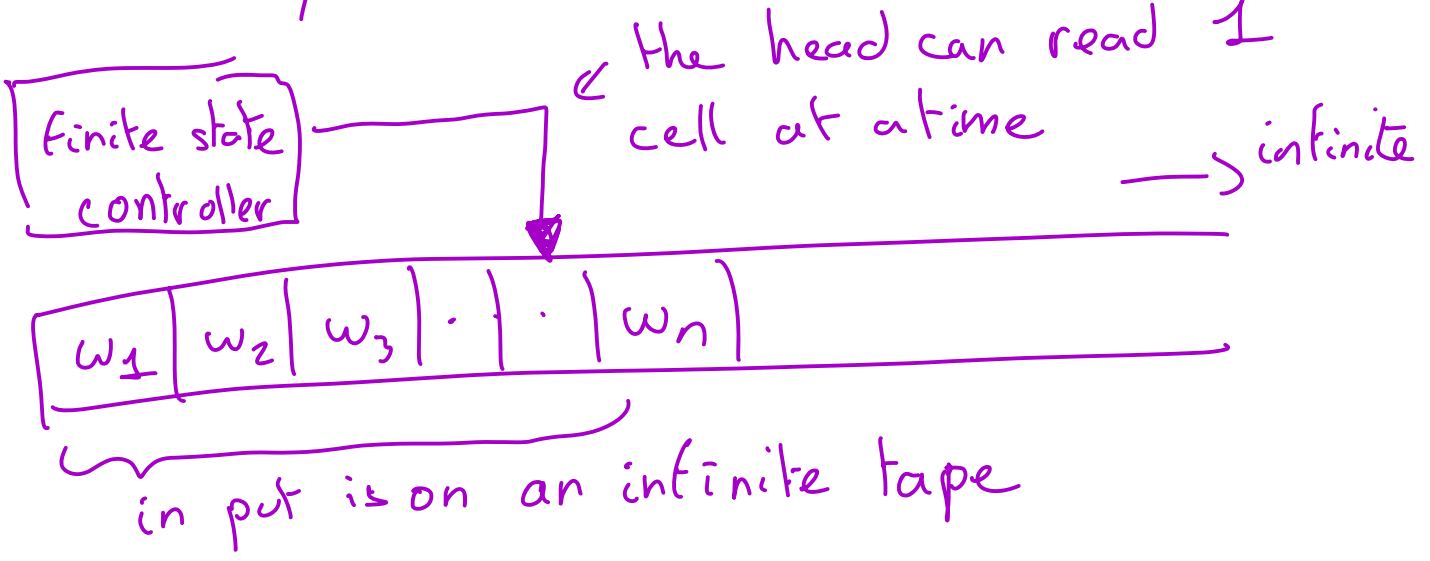
  - parallelism
  - memory → a computer can store the entire input

  - non determinism => the computer has it
  - can write to the memory
  - ALU

Next lecture Turing machines (TM)

- unbounded memory
- unlimited access to the input
- write to memory

TM, informaly



The head can move left / right and it can update the cell it's looking at.