Lecture 3

Tuesday, 2 February, 2021 10:49

Today: a <u>very</u> simple model of computation for deciding language membership (aka Y/N problem)

- given input, read it char by char
- keep track of a finite amt of memory
  - finite ← does not depend on input
- when no more input, outputs Y/N (accept/reject)

variants used in ... simple robots, network controllers
language processing

Pieces:

- each possible configuration of memory is called a <u>state</u>

  Q = set of possible states

- start machine in some "starting" state $s \in Q$
- given the current state, the next input char tells us how to update mem, or, which state to <u>transition to</u>

  transition function $\delta : Q \times \Sigma \to Q$

- at the end, we are in an <u>accepting</u> state or <u>rejecting</u> state.

  A = set of accepting states

  ($Q \setminus A$ = set of rejecting state)

Deterministic Finite Automaton (DFA)

DFA $M = (Q, \Sigma, \delta, s, A)$ ← set of accepting states

states alphabet transition starting

states alphabet transition fn starting state accepting states

Ex. Prog for determining if, given a binary string w
if $\#(1, w)$ is odd

Idea: count the # of 1s.
Problem: the amount of memory depends on the input!

Actually keep track of parity of this count

Parity1s: state. possibilities are $\{0,1\}$
parity ← 0
while input
read character a
if a = 1 :
parity ← 1 − parity transition
return (parity = 1)

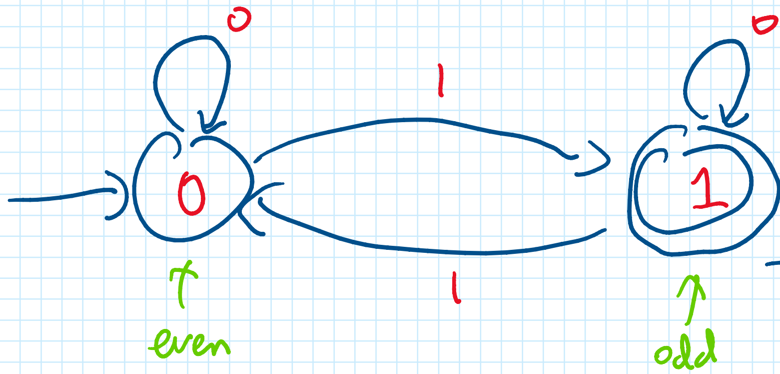
$Q = \{0,1\}$ $\Sigma = \{0,1\}$

$$\delta(q,a) = \begin{cases} q & \text{if } a=0 \\ 1-q & \text{if } a=1 \end{cases}$$

$s = 0$ $(\in Q)$
$A = \{1\}$

Graphical representation of DFAs
- vertices are states
- edges represent transitions
- for each state, there is exactly one outgoing edge per $a \in \Sigma$ (sometimes combined for visualization)
- pointer to starting state

- pointer to starting state
- doubled circles for accepting



e.g. on input 01101

state seq:

$\to 0 \to 0 \to 1 \to 0 \to 0 \to 1$

1 is accepting so return yes

on input 1010?

state seq:

$\to 0 \to 1 \to 1 \to 0 \to 0$

<u>reject</u>

Program for determining if binary string contains 00.

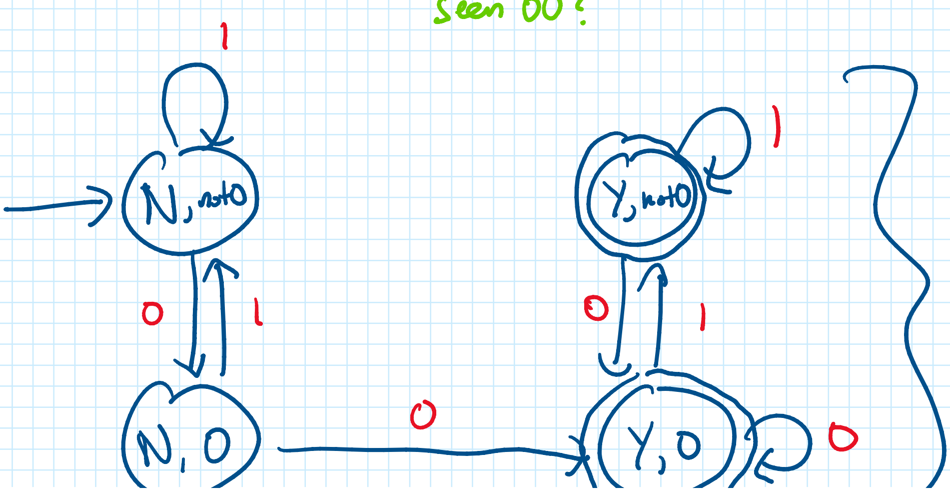
- keep track of if we've seen 00
- keep track of prev char.

  if prev = 0 and curr = 0,

  we are looking at <u>00</u>

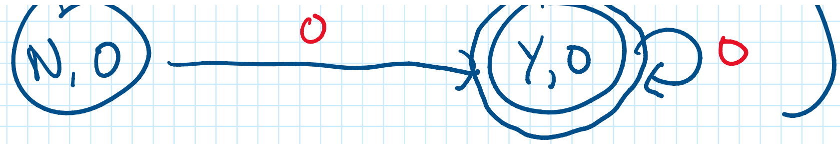
keep track of (found, prev) $\in \{Y, N\} \times \{0, \text{not}0\}$

have we seen 00?



Combine into Y ?

eh.



eh.

1001

$\to$ N, not0 $\to$ N, not0 $\to$ N, 0 $\to$ Y, 0 $\to$ Y, not0

0101

$\to$ N, not0 $\to$ N, 0 $\to$ N, not0 $\to$ N, 0 $\to$ N, not0

Mathematical properties of DFAs.

given a DFA M, the language accepted by M is $L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$

what does it mean for M to accept w?

after reading in w, we ended on an accepting state.

Q: what is the state we end on?

we'd like to give it a label like $\delta^*(w) \in Q$.

Define extended transition fn δ^*

$$\delta^* : Q \times \Sigma^* \to Q$$

$\delta^*(q, w)$ = state we end on if, starting from q, we read w.

$$\delta^*(q,w) = \begin{cases} q & \text{if } w = \varepsilon \\ \delta^*(\delta(q,a), x) & \text{if } w = ax \end{cases}$$

string is either ε or ax

$$L(M) = \{ w \in \Sigma^* \mid \delta^*(s, w) \in A \}$$

Def. A language is automatic if there is a DFA

Def. A language is <u>automatic</u> if there is a DFA that accepts it

specific to 374

spoilers: automatic $\updownarrow$ regular

Closure properties

recall regular languages are closed under:

- (finite) union
- (finite) concat
- Kleene star

if we do this to (a) regular language(s) we get a reg lang

what can we do to automatic langs to get an auto lang?

→ gives a way of building complex DFAs for harder langs from simpler DFAs for easier langs.

- Complement

regex: $\underline{\text{not } 0}$

$\varepsilon + 1 + (0+1)(0+1)(0+1)^*$

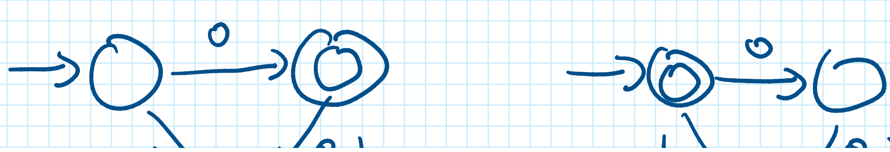
$$\begin{aligned}
\overline{L(M)} &= \Sigma^* \setminus L(M) \\
&= \Sigma^* \setminus \{ w \in \Sigma^* \mid \delta^*(s,w) \in A \} \\
&= \{ w \in \Sigma^* \mid \delta^*(s,w) \notin A \} \\
&= \{ w \in \Sigma^* \mid \delta^*(s,w) \in Q \setminus A \}
\end{aligned}$$

Given $M = (Q, \Sigma, \delta, s, A)$

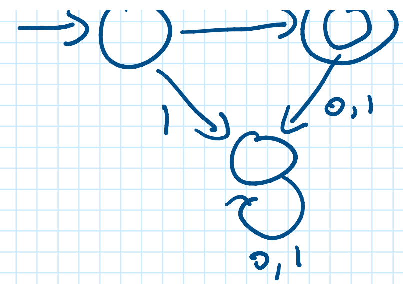
define $\overline{M} = (Q, \Sigma, \delta, s, Q \setminus A)$

the preceding derivation proves that $L(\overline{M}) = \overline{L(M)}$

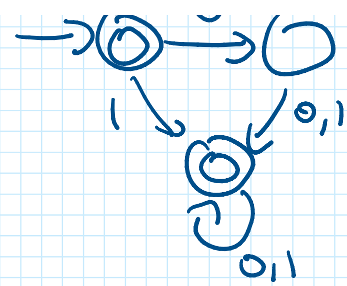
e.g.



e.g.



DFA for $\{0\}$



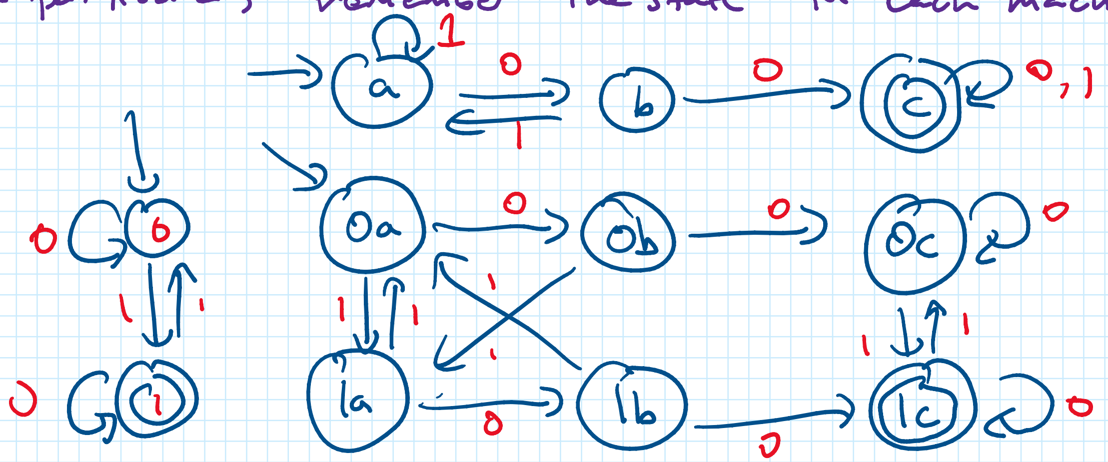
DFA for $\Sigma^* \setminus \{0\} = \overline{\{0\}}$

– Intersection

ex. $\#(1,w)$ is odd **AND** contains 00

idea. simulate keeping track of both machines.

in particular, remember the state in each machine.



10011

state seq's: $\rightarrow 0 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow 1$

$\rightarrow a \rightarrow a \rightarrow b \rightarrow c \rightarrow c \rightarrow c$

Procedure: Product Construction

Given $M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$

$M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$

Same!

$$M = (Q, \Sigma, \delta, s, A)$$

$Q = Q_1 \times Q_2$ states are tuples of states

$Q = Q_1 \times Q_2$ states are <u>tuples</u> of states

$= \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$

$\Sigma = \Sigma$ $\quad s = (s_1, s_2) \quad (\in Q_1 \times Q_2)$

$A = \{(q_1, q_2) \mid q_1 \in A_1, q_2 \in A_2\} = A_1 \times A_2 \subseteq Q_1 \times Q_2$

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

Thm: $L(M) = L(M_1) \cap L(M_2)$

Pf: (see textbook)

The DFA given by prod construction might not be the smallest DFA for $L(M_1) \cap L(M_2)$.

We don't care (for now)

- Union $\quad L(M_1) \cup L(M_2)$

  also by product construction except

  $A = \{(q_1, q_2) \mid q_1 \in A_1$ <u>or</u> $q_2 \in A_2\}$

  everything else is the same.

- Set Difference $\quad L(M_1) \setminus L(M_2)$

  for sets S, T. $\quad S \setminus T = S \cap \overline{T}$.

  $= \{x \mid x \in S \text{ and } x \notin T\}$

  so Given M_1, M_2.

do product construction on

$M_1, \overline{M_2}$

ex. #(1, w) is odd, except the ones containing 00