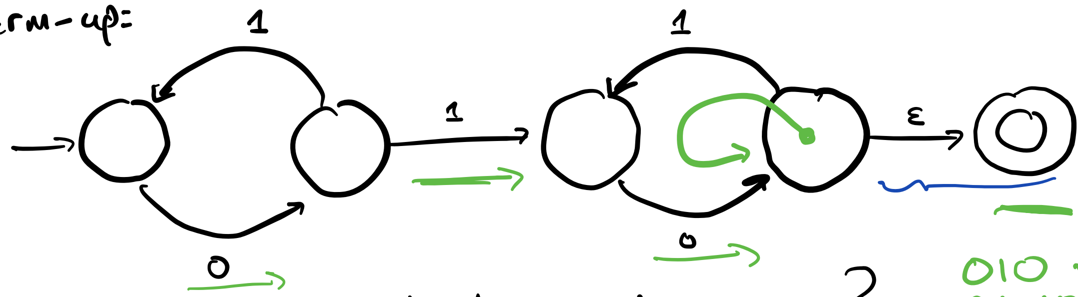
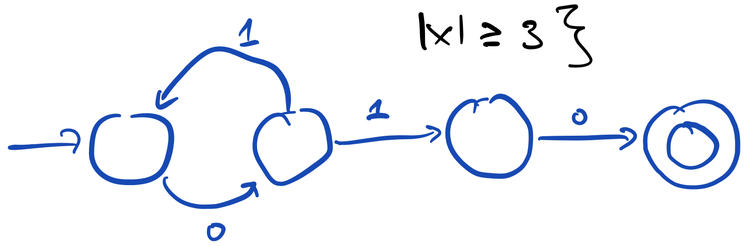
Warm-up:

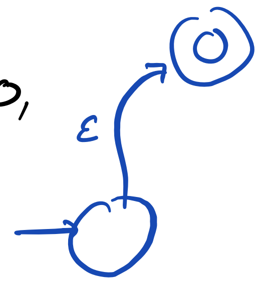


010 ✓
01010 ✓
0101010 ✓

- what language does this NFA recognize?
- can you build an equivalent NFA with ≤ 4 states?
  - and ≤ 4 transitions?

$\{x \in \{0,1\}^* \mid x$ starts, and ends with 0, alternates 0's and 1's, $|x| \geq 3\}$





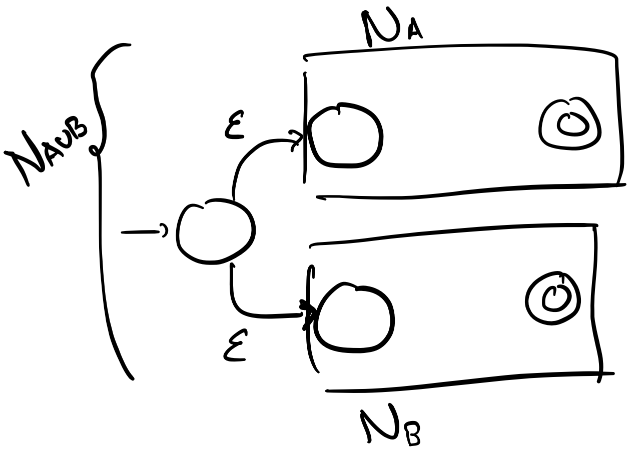
Last time:

- Regular languages closed under complement ($\overline{}$), closed under union ($\cup$), intersection ($\cap$),
- New automaton - NFA
  NFA-recognizable languages closed under ($\cup$)

Today:

- NFA-recognizable languages closed under $\cup$, $\circ$, $*$
- NFAs can be converted to equivalent DFA's!
- Regular Expressions: RegEx $\rightarrow$ NFA,

DFAs → RegEx.

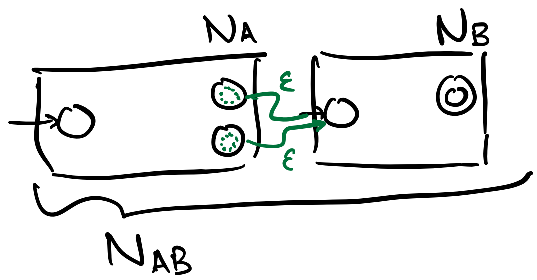


Claim: Regular Languages are closed under union ($\cup$).

Proof: By NFA modification.

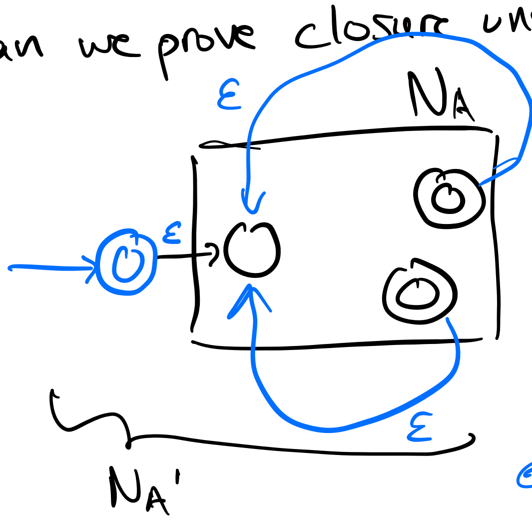
- A, B regular
- N_A, N_B be NFAs that recognize A, B.

(Last time: closure under concatenation:



)

Q: Can we prove closure under star ($*$) similarly?



- N_A recognizes the regular language A.

($A^* :=$ $\{a_1 a_2 \cdots a_k \mid a_i \in A, k \geq 0\}$)

- add ε-transitions from each accept state to start.
- add an extra state to accept ε.

Proof. $L(N_A') = A^*$.

1) If $w \in A^*$, then N_A' accepts on w.

By definition, $w = w_1 w_2 \cdots w_k$ for $w_1, w_2 \cdots w_k \in A$, $k \geq 0$. Because $L(N_A) = A$, each string w_i corresponds to a sequence of transitions from N_A's start state to some accept state.

$\therefore$ $w_1 w_2 \dots w_k$ corresponds to a path from the start state of N_A' to an accept state that loops back to the start on our new ε-transitions $k-1$ times.

2) If N_A' accepts w, $w \in A^*$. To accept, there must exist a sequence of transitions from start to an accept state in N_A', on w.

This sequence must consist of one or more subsequences that travel from start to accept, divided by ε-transitions back to the start.

Each subsequence corresponds to a string in A, so w can be written as the concatenation of 1 or more strings from A.

(side case: $w = \varepsilon$.)

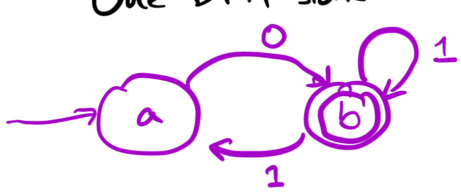
<u>Theorem:</u> NFAs, DFAs recognize the same set of languages: the regular languages.

<u>Proof (DFA $\Rightarrow$ NFA).</u> DFA state diagrams are NFA state diagrams

<u>Proof (NFA $\Rightarrow$ DFA).</u>

<u>Idea:</u> Track every live branch of computation at once.

One DFA state will represent multiple NFA states.

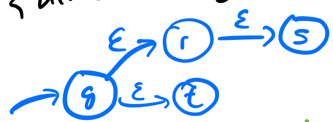


$\{a\}$ $\quad$ $\{a,b\}$

$\{b\}$ $\quad$ $\emptyset$

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We'll construct a DFA $D = (Q', \Sigma, \delta', q_0', F')$ that accepts if and only if N accepts.

$$Q' = \mathcal{P}(Q)$$ // $\mathcal{P}(Q) = \{\text{all subsets of } Q\}$



Σ unchanged

$q_0' = E(\{q_0\})$ $E(A)$ = "the states reachable from A by ε-transitions."

$F' = \{R \mid R \subseteq Q,\ R \cap F \neq \emptyset\}$

$\delta' =$ (
(1) We occupy some DFA state = set of NFA states.
(2) We read in $a \in \Sigma$.
(3) We move to occupy all states reachable from our current position by a-transitions.
(4) We occupy all states reachable by ε-transitions.
)

For $R \subseteq Q$, and $a \in \Sigma$, define

$$\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$$

$$\delta' : \mathcal{P}(Q) \times \Sigma \longrightarrow \mathcal{P}(Q).$$

Claim: N accepts string $w = w_1 w_2 \cdots w_n$, $w_i \in \Sigma$, if and only if D accepts w. Proof by induction.

- At "step" 0: N, D both occupy $E(q_0)$.
- Assume N, D both occupy $R_i \subseteq Q$ at step i.
- At step $i+1$: N occupies $\bigcup_{r_i \in R_i} E(\delta(r_i, w_{i+1}))$

  D occupies $\delta'(R_i, w_{i+1}) =$
  $\{q \in Q \mid q \in \underline{E(\delta(r_i, w_{i+1}))} \text{ for } r_i \in R_i\}$

$\therefore$ At step n, N and D occupy the same state(s) $R_k \subseteq Q$.

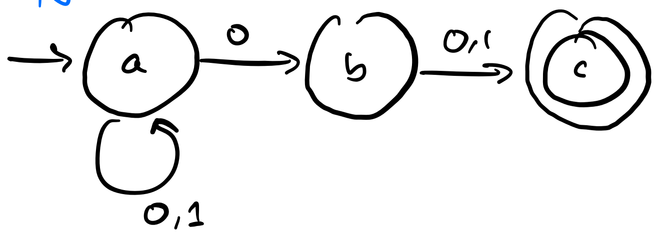
Both accept if and only if $R_k \cap F \neq \emptyset$;

when R_k contains an accept state. $\square$

Example conversion: NFA $\to$ DFA.

$b \to f$ 0,1

N:



$N = (Q, \Sigma, \delta, q_0, F)$:

$Q = \{a, b, c\}$ ✓

$\Sigma = \{0, 1\}$ ✓

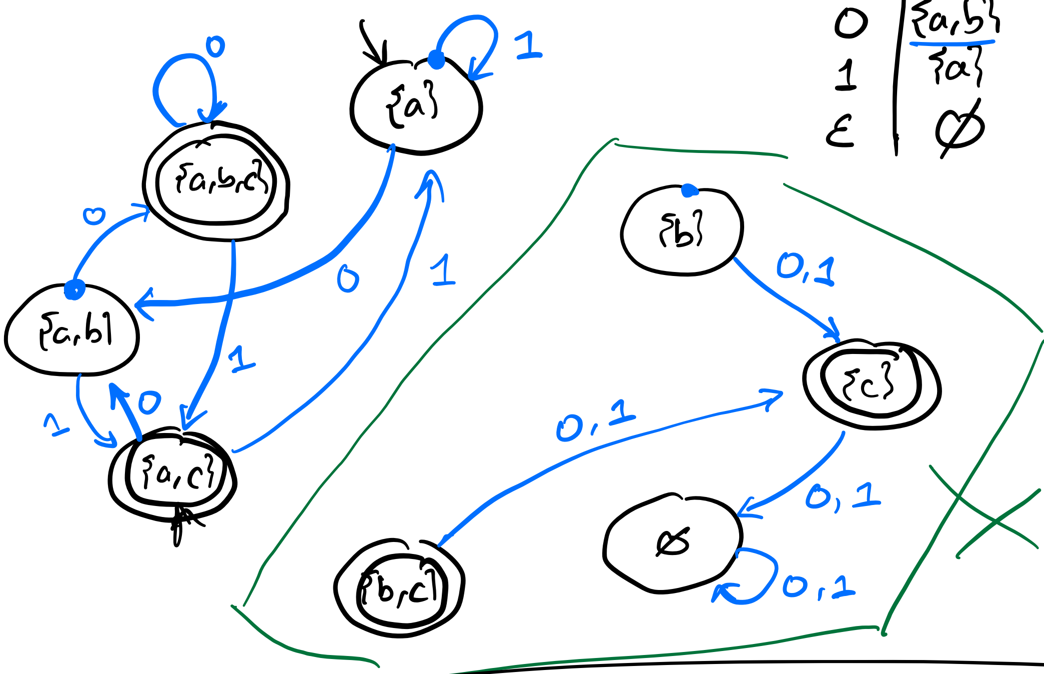
$q_0 = a$ $\qquad$ $E(\{a\})$

$F = \{c\}$

$L(N)$: $\{x \in \{0,1\}^* \mid x$ ends in 00 or 01$\}$.

δ:

	a	b	c
0	$\{a,b\}$	$\{c\}$	$\emptyset$
1	$\{a\}$	$\{c\}$	$\emptyset$
ε	$\emptyset$	$\emptyset$	$\emptyset$.



Back at 2:35

Punchline:

NFA-recognizable

$=$ DFA-recognizable

$=$ Regular languages.

Regular languages are closed under $\neg, \cup, \cap, \circ, *$

$A = \{x \in \{0,1\}^* \mid$ x consists of an odd number of 0's, followed by even num of 1's, OR an even num of 0's, followed by an odd num of 1's$\}$

000 1111

00 111

$\hookrightarrow$ $B = \{x \in \{0,1\}^* \mid x$ is an odd num of 0's$\}$

$C = \{x \in \{0,1\}^* \mid x$ is an even num of 0's$\}$

$D = \{$ ______ " ______ odd num of 1's $\}$

$E = \{$ ______ " ______ even num of 1's $\}$

Say B, C, D, E regular.

$(B \circ E) \cup (C \circ D) = A$ regular, by closure under $\circ$, $\cup$.

---

2. Regular Expressions.

unix: find "HW[0-9]*.(tex|pdf)"

HW4.pdf

HW137.tex

HW.pdf

$\{tex, pdf\}$

$\{H\} \circ \{W\} \circ \{0,1,2,\dots,9\}^* \circ (\{tex\} \cup \{pdf\})$

$\{HW\} \circ \{0,1 \dots 9\}^* \circ \{tex, pdf\}$

Idea: regular operations combine simple languages to produce more complex ones.

Def. (Regular Expression). (Inductive defn). R is a regular expression if

(1) R is an "atom": $a \in \Sigma$ $(\{a\})$, ε $(\{\varepsilon\})$, or $\emptyset$

(2) R results from applying $\cup$, $\circ$, or $*$ to other regular expressions: $R = R_1 \cup R_2$, $R_1 \circ R_2 = R_1 R_2$, R_1^*, where R_1, R_2 are regular expressions.

— Σ denote $\underline{\bigcup_{a \in \Sigma} \{a\}}$ ≈ "any character in Σ" ($\Sigma = \{0, 1\}$, in reg. ex, Σ denotes $(0 \cup 1)$)

— $R^+ = RR^*$ ≈ "at least one string from R, concatenated"

— R^k, for some $k \geq 0$: "k concatenated strings from R." $(R^4 = RRRR)$

Examples:

- $\Sigma\Sigma\Sigma$ ≈ any 3 elements of Σ, concatenated.

  If $\Sigma = \{0, 1\}$, 000, 001, 010, 100, 011, 101, 110, 111

- $(0 \cup 1)^*$ ≈ "any binary string"
- $(0 \cup 1)^* 1$ ≈ any binary string ending in 1.
- $0^* 1 0^*$ = every binary string w/exactly one 1.

$\{\varepsilon, 0, 00, 000, \ldots\} \circ \{1\} \circ \{\varepsilon, 0, 00, 000 \ldots\}$

- $(\Sigma\Sigma)^* = \{\varepsilon, 00, 01, 10, 11, 0000, \ldots\}$

$D = \{0, 1, 2, 3, \cdots 9\}$ ("-", "." $\in \Sigma$)

- $\underbrace{(\varepsilon \cup -) D^+ . D^+}$

12.34

-4.0

99.9999

$R^+ = RR^*$

$\{r_1 r_2 \cdots r_k \mid r_i \in R, k \geq 1\}$

Puzzle:

Reg. ex.'s for the following: ($\Sigma = \{0, 1\}$)

1. Strings of even length ending in 1. $(\Sigma\Sigma)^*$ $\Sigma^* 1$ | $(\Sigma\Sigma)^* \Sigma 1$
2. Strings with exactly two 1's. $0^* 1 0^* 1 0^*$
3. Strings over $\{a, b, c\}$ in which no a's follow b's, no a's or b's follow c's.

ba
ca
cb

$a^* b^* c^*$

4*. strings that don't contain "00."

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

$\underbrace{1^* \cup}_{(0 \cup \varepsilon)} \ 1^* 0 (1^+ 0)^* 1^*$ correct (?)

$1^* 0 \underline{1^+} 0 1^*$

$1^* 0 1^+ 0 1^+ 0 1^*$

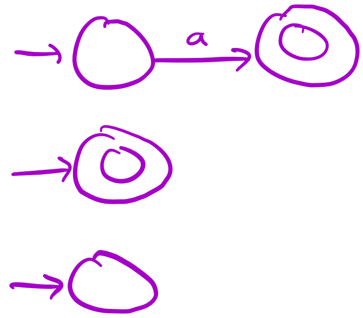
$1^* 0 1^+ 0 1^+ 0 1^+ 0 1^*$

$(0 \cup \varepsilon)(1^+ 0)^* 1^*$ = better (?)

Prop: Regular expressions have equivalent NFAs.

Proof: Let R be a regular expression. By definition, R matches one of 6 cases:

$$R = \begin{cases} a \in \Sigma \\ \varepsilon \\ \emptyset \end{cases}$$

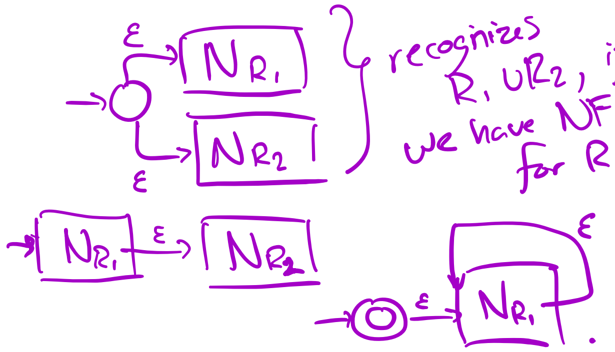


recognizes $\{a\}$

$\{\varepsilon\}$

$\emptyset$

$$R = \begin{cases} R_1 \cup R_2 \\ R_1 R_2 \\ R_1^* \end{cases}$$



recognizes $R_1 \cup R_2$, if we have NFAs for R_1, R_2.

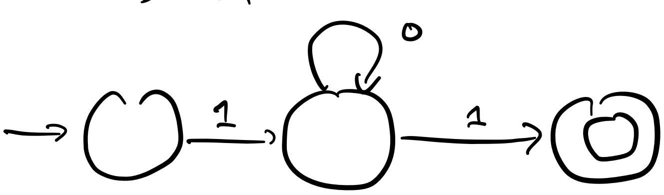
Ind. Hyp: We can build NFAs for all Reg Ex.'s that use at most k symbols

$\Longrightarrow$ We can build NFAs for all Reg Ex's that use $k+1$ symbols, by constructions above. $\square$

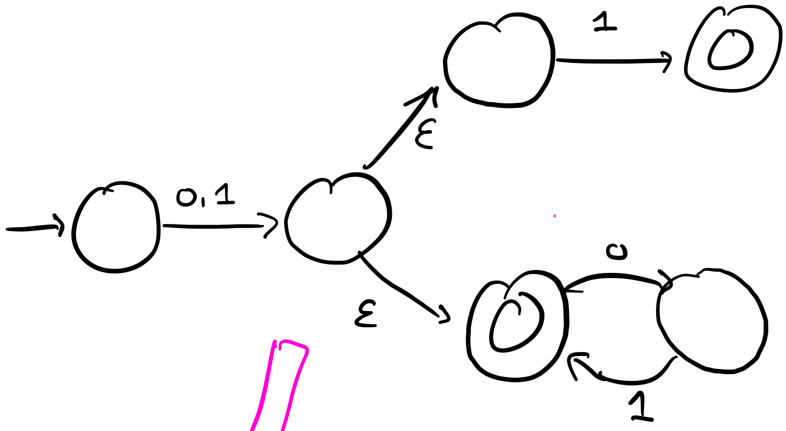
Back at 3:42.

Showed: Reg. Ex $\longrightarrow$ NFA.

To do: DFA $\longrightarrow$ GNFA $\longrightarrow$ Reg. Expr.



$\approx 10^*1$

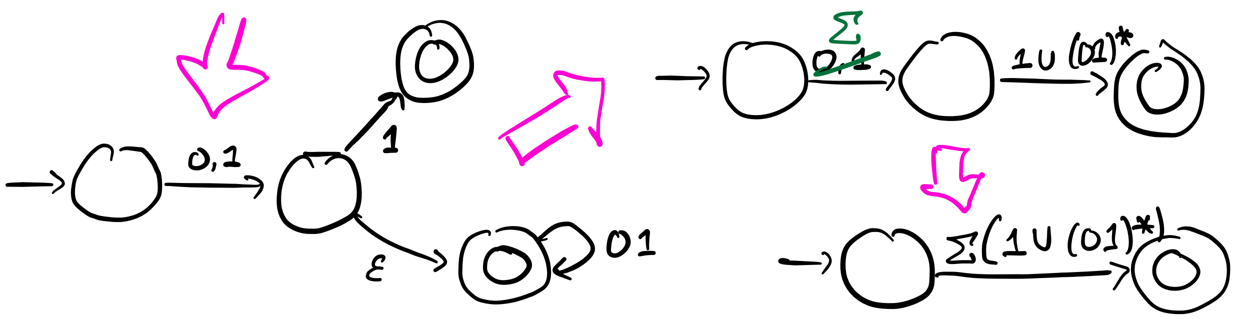


Top branch: $\Sigma 1$

Bottom branch: $\Sigma(01)^*$

Total: $\Sigma(1 \cup (01)^*)$

Main proof idea: progressively "reduce" automata by using more and more complex edge labels.



Generalized NFAs (GNFAs):

New rules:

- can label edges with any regular expression

(as before, we'll accept if and only if there is a path from start to accept state matching the input string.)

- allow exactly one start and one accept state.
- exactly one transition between every ordered pair of states (usually, labeled $\emptyset$)
  - except none into the start
  - or out of the accept state.

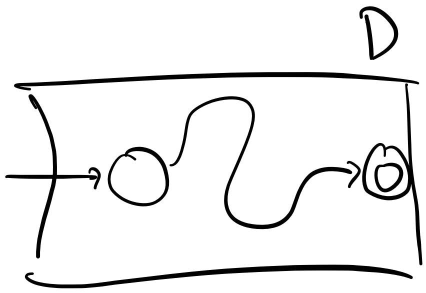
Reminders: HW2 (short) due Mon, @ 11:59 PM

My Zoom hours tonight 5:30

and Sunday 5:30

Next time: On beyond regular languages.

HW 1, Q5:



$w \in L(D)$

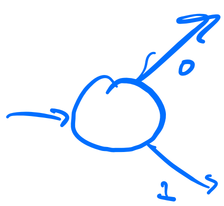
$doub(L(D))$

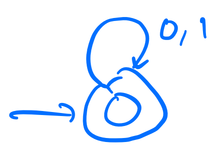
$w = w_1 w_2 w_3$

$\Downarrow$

$w_1 w_1 w_2 w_2 w_3 w_3$

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

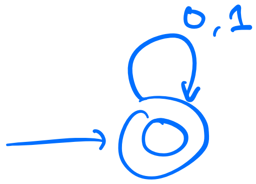


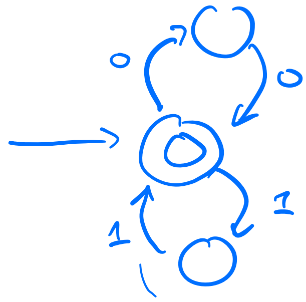


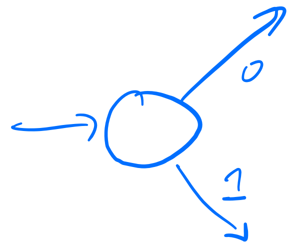
$\{0, 1\}^*$

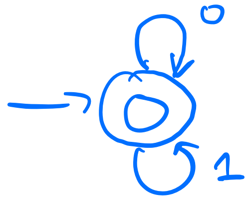
$doub(L(D)) = \{\varepsilon, 00, 11, 0000, 0011, \}$

$doub(\{0, 10, 111\}) = \{00, 1100, 111111\}$


0,1


0
0
1
1


0
1


0
1