Today:

1. Regular expressions evaluate to (exactly the) regular languages
   - 1.1 regex → NFA
   - 1.2 DFA → regex
     - 1.1.1 GNFA
     - 1.1.2 DFA → GNFA
     - 1.1.3 GNFA → Regex

   } – structure of big proofs – tinkering math

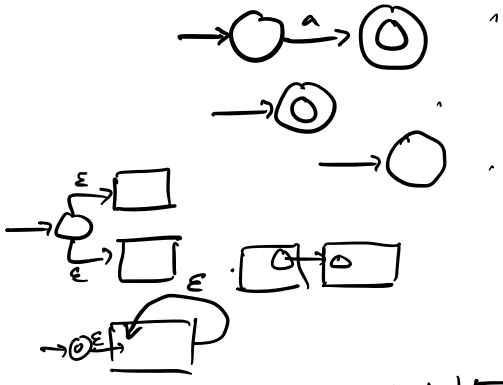
   Punchline: DFAs, NFAs, Regexs equivalent.

2. Nonregular Languages
   - 2.1. Pumping Lemma.

Regular expressions, inductively:

(1) $R = a$, $a \in \Sigma$ $(\{a\})$
(2) $R = \varepsilon$ $(\{\varepsilon\})$
(3) $R = \emptyset$
(4) $R = R_1 \cup R_2$
(5) $R = R_1 R_2 = R_1 \circ R_2$
(6) $R = R_1^*$

Equiv NFA construction:

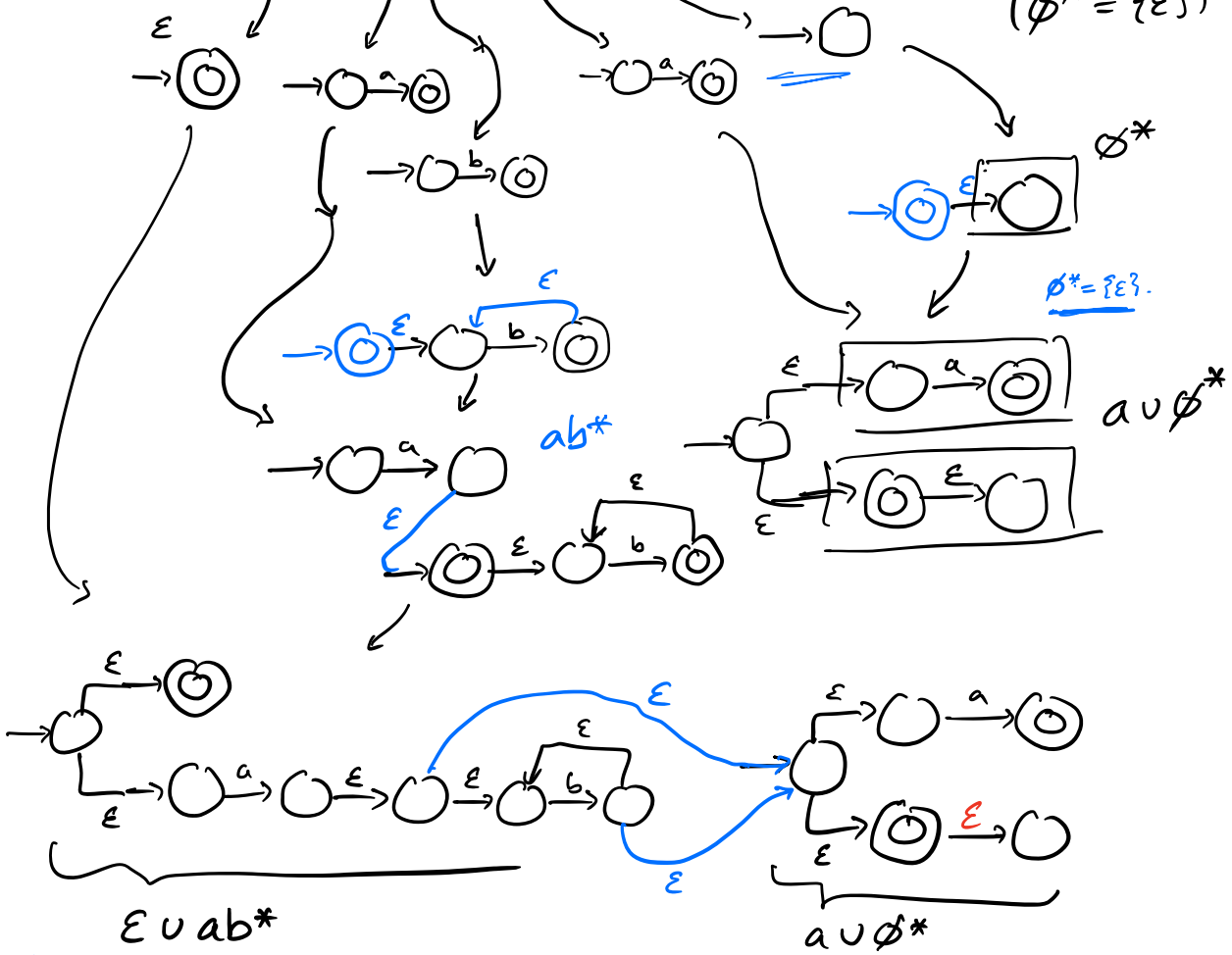


Showed: turn a regular expression into an equivalent NFA by building NFA "gadgets" for each of our 6 inductive rules.

Example:

Example. $(\varepsilon \cup ab^*)(a \cup \emptyset^*)$. Convert to NFA.

order of operations: star first!



$(\varepsilon \cup ab^*)(a \cup \emptyset^*)$

$(\varepsilon \cup ab^*)(a \cup \underbrace{\emptyset^*}_{=\varepsilon})$

$= (\varepsilon \cup ab^*)(a \cup \varepsilon)$

$= (\cancel{\varepsilon}a \cup \cancel{\varepsilon\varepsilon} \cup ab^*a \cup ab^*\varepsilon)$

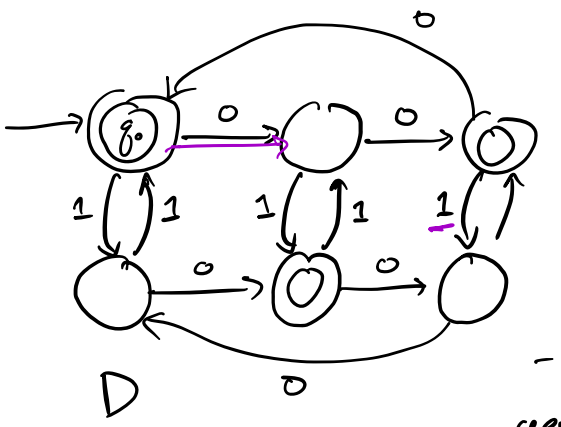
$\{a\} \in ab^*$ $(\varepsilon \cup ab^*a \cup ab^*)$

...

Takeaway: Languages generated by regular expressions are regular.

Want to show: All regular languages can be generated by some regular expression.

Regular Language → DFA ✓ → ? → Reg. ex.



String is accepted ↔ computational path on that string takes us from q_0 to an accept state.

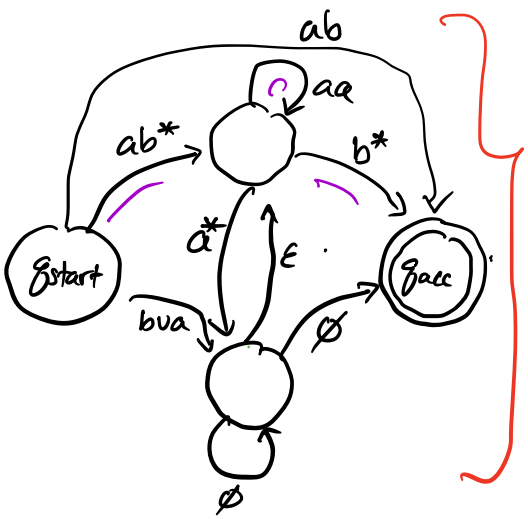
- A regular expression that generates the labels of every accepting path evaluates L(D).

To do this, we'll cook up a math machine that uses regular expressions as edge labels.

Generalized NFA (GNFA)

Special rules:

- 1 start state, 1 accept state
- 1 transition between every (ordered) pair of states.
- no transitions to the start or from the finish.



bbbb ✓

(ab) ✓

(abb)(aa)(b) ✓

wait!
- multiple paths?
- unclear computation?

Formally: A GNFA is a 5-tuple

$$(Q, \Sigma, q_{start}, q_{accept}, \delta),$$

Q: state set; Σ: alphabet $\{a, b\}$; δ: transition function

with transition function

$$\delta : (Q \setminus \{q_{accept}\}) \times (Q \setminus \{q_{start}\}) \to \mathcal{R},$$

$\mathcal{R}$:= the set of regular expressions over Σ.

$\delta(q_{start}, q_{accept}) = ab.$ (!)

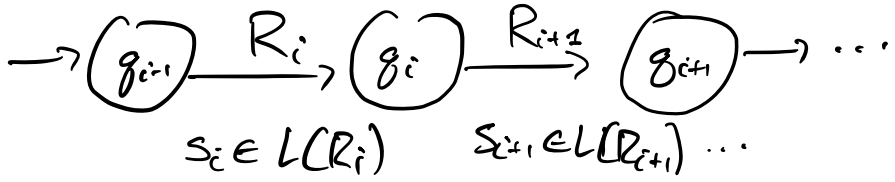
(DFA $\delta : Q \times \Sigma \to Q$.)

- "computation"? Not well-defined.

A GNFA accepts a string $w = s_1 s_2 \cdots s_k$, with all s_i's strings over Σ, if there is a sequence of states

$$q_{start}, q_1, q_2 \cdots, q_{accept}$$ with

each string $s_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

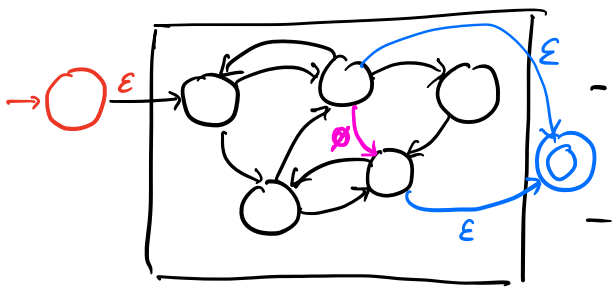


Recall our goal: DFA → GNFA → Reg ex.

(1) DFA → GNFA.

(Sketch).

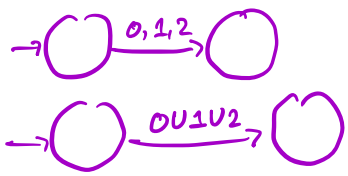
D



How do we turn this into a GNFA (state diagram) that is equivalent?

- Good news: edge labels like 0, 1 are already reg. ex.s!
- Issues:
  (1) multiple edges between pairs of states
  (2) edges into start
  (3) edges from finish
  (4) multiple accept states
  (5) missing edges

(1) Fix: take the union.



(2) Edges into start: new start, w/epsilon edge

(3) Edges into Finish: new accept, ε-edges, Also fixes (4)

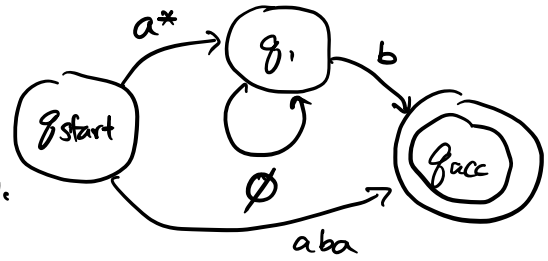
(5) Replace missing edges with ∅-edges. (∅ matches no string.)

Result: a GNFA that accepts the same set of strings as the input DFA.

— Next: GNFA → DFA; now: Q's / break until 2:25. —

Ex: GNFA empty set transitions.

"dummy edges"
no accepting path uses an $\emptyset$-transition.



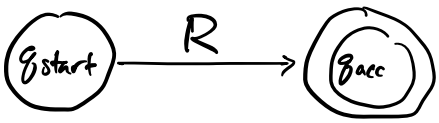
$\varepsilon b = b$

ab aba
aab
aaab

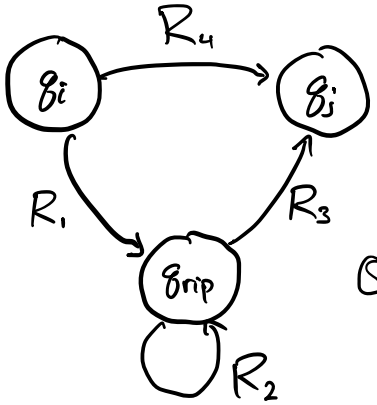
$a^* \emptyset b = \emptyset$

1.4 GNFA → Reg Ex.

Idea: remove one state at a time, "redirecting traffic" to other transitions. Eventually, we'll get



accepts strings that match R!

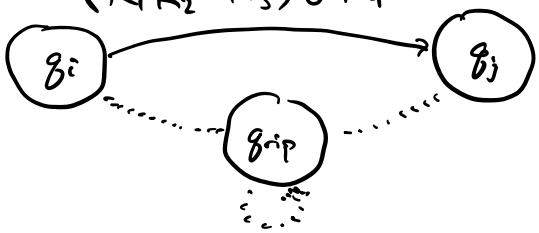


goal: reroute all traffic from q_i to q_j that passes through q_{rip}.
(Do for all possible pairs q_i, q_j).

Q: How can we go $q_i \to q_j$, optionally through q_{rip}?

$R_1 R_3$
$R_1 R_2 R_3$
$R_1 R_2 R_2 R_3$
⋮
} $R_1 R_2^* R_3$

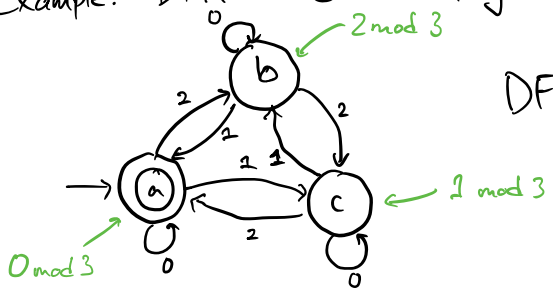
A: Replace R_4 with $(R_1 R_2^* R_3) \cup R_4$



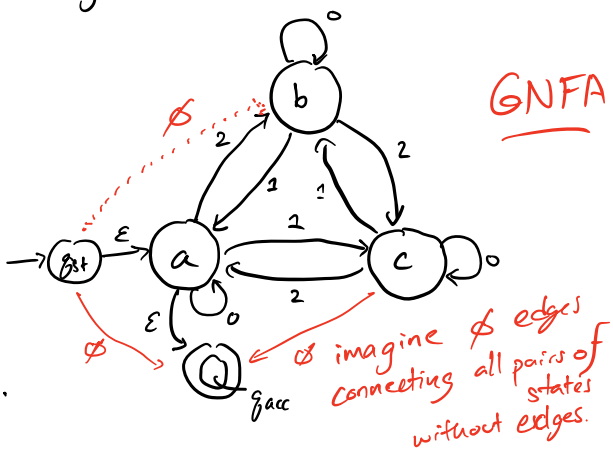
Overall strategy: do this for all (q_i, q_j), so q_{rip} is redundant and can be removed.

(Repeat removing states until we reach → q_{start} —R→ q_{acc}.)

Example: DFA → GNFA → Reg. Ex.



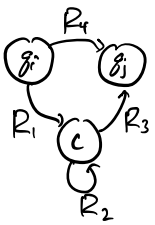
DFA: sums up digits of the input string and accepts if the result is divisible by 3.

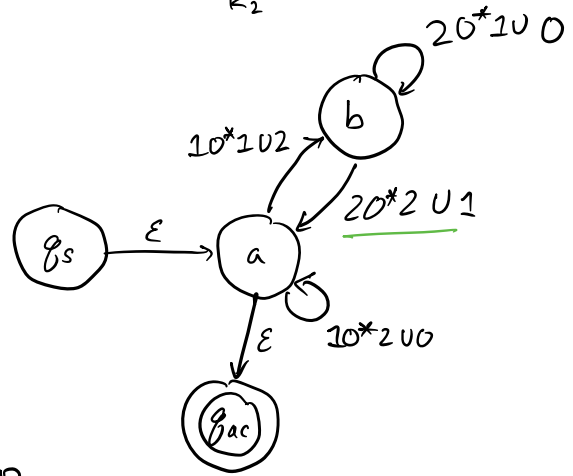


1. DFA → GNFA.
   - start w/no incoming edges
   - accept w/no outgoing edges
   - edges everywhere else.

2. Rip out states & redirect traffic.
   - Rip out c.

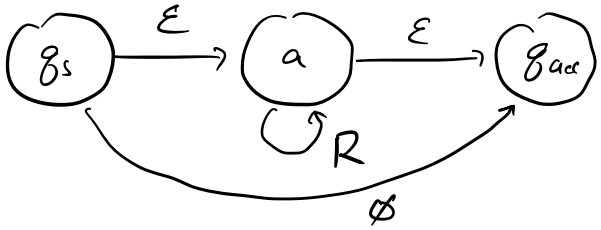
State pair (q_i, q_j)	Reg. ex $(R_1 R_2^* R_3) \cup R_4$
(q_{start}, q_{accept})	$\emptyset$
(q_s, a)	$(\emptyset 0^* 2) \cup \varepsilon = \varepsilon$
(q_s, b)	$\emptyset$
(a, a)	$(10^*2 \cup 0)$
(a, b)	$10^*1 \cup 2$
(a, q_a)	$\emptyset \cup \varepsilon = \varepsilon$
(b, b)	$20^*1 \cup 0$
(b, a)	$20^*2 \cup 1$
(b, q_{acc})	$\emptyset$





   - Rip out b.

State pair (q_i, q_j)	$(R_1 R_2^* R_3) \cup R_4$
(q_s, q_a)	$\emptyset (20^*1 \cup 0)^* \emptyset \cup \emptyset = \emptyset$
(q_s, a)	$(\emptyset (\text{stuff})^* (\text{stuff})) \cup \varepsilon = \varepsilon$
(a, a)	$((10^*1)\cup 2)((20^*1)\cup 0)^*((20^*2\cup 1) \cup ((10^*2)\cup 0)$
(a, q_{acc})	$\emptyset \cup \varepsilon = \varepsilon$ $= R$



$(q_s, q_{acc}) \Rightarrow \varepsilon R^* \varepsilon \cup \emptyset$

Finally: we have that $L(D)$ is equal to the language represented by the reg. exp.

$$R^* = \Big(\big((1O^*1)\cup 2\big)\big((2O^*1)\cup O\big)^*\big((2O^*2)\cup 1\big) \cup \big((1O^*2)\cup O\big)\Big)^*.$$

Reference video: DFA → GNFA → Reg Ex.

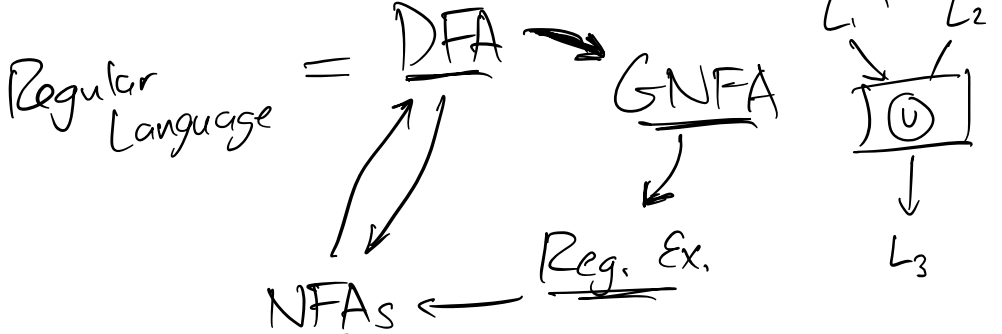
Also: Sipser 69-74.

Eye on the ball:

Proved: DFAs have equivalent regular expressions.

DFA ⟶ GNFA ⟶ Reg. ex.

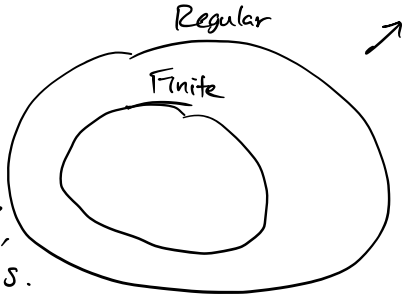
Takeaway:



All of these tools are equivalent in their expressive power!

— Break — to 3:12 —

Beyond Regular Languages



Puzzle:

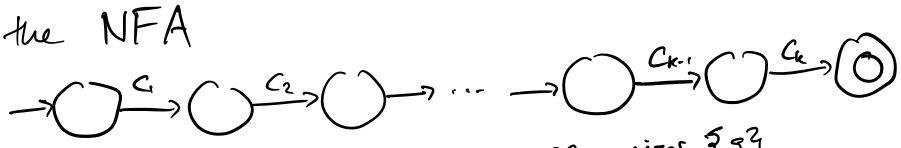
- Given a string $s = c_1 c_2 \cdots c_k$, $c_i \in \Sigma$, build a state diagram that accepts only s.
  ("sketch" — DFA or NFA — $\{s\}$.)
- Sketch an NFA or use closure under regular operations to show that any finite language $\{s_1, s_2 \cdots s_n\}$ is regular.

(Def. Finite Language is a finite set of strings.)

Proposition. Finite languages are regular.

Proof. Suppose I have a string $s = c_1 c_2 \cdots c_k$.

Then the NFA



recognizes $\{s\}$.

By closure under union, any finite language $L = \{s_1, s_2 \cdots s_n\}$ is regular, because

$$L = \{s_1\} \cup \{s_2\} \cup \cdots \cup \{s_n\} = \bigcup_{0 < i \leq n} \{s_i\}. \quad *$$

* Why doesn't this work for infinite languages? infinity stuff.

So: All finite languages, and some infinite ones ($\{0,1\}^*$, etc.) are regular! Is anything not?

$\{w \in \Sigma^* \mid w = w^R\}$ (palindromes)

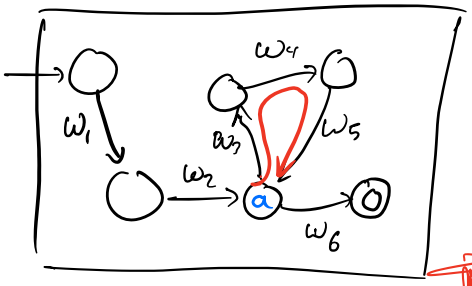
$\{0^n 1^n \mid n \geq 0\}$ 0 string, then 1 string of same length.

Idea: (1) Show "all regular languages satisfy X."
(2) Show "my language L doesn't satisfy X."
∴ L isn't regular.

A Property of all (Infinite) Regular Languages.

DFA

D



($|Q|=6$)
($w_1 w_2 \cdots w_6$)

- Say $L(D)$ is regular and infinite.
- So: we can pick $w \in L(D)$ that satisfies $|w| \geq |Q|$, the number of states in D.
- ∴ this string's path has a loop!

"Long enough strings in DFAs make loops."

$$w_1 w_2 \underbrace{w_3 w_4 w_5}_{a\ \ a} w_6 \qquad (w_i \in \Sigma)$$

↑ take me around a loop

$$w_1 w_2 w_6 \in L(D)$$
$$w_1 w_2 (w_3 w_4 w_5)(w_3 w_4 w_5) w_6 \in L(D).$$

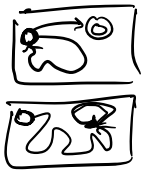
Pumping Lemma (Informal.) Let D_A be the DFA for some infinite regular language A. All sufficiently long strings $w \in A$ make a loop when evaluated by D_A.

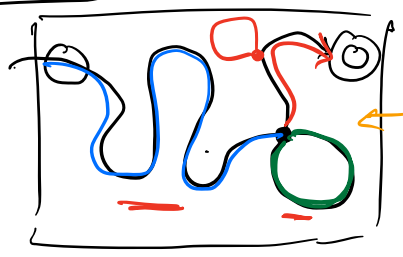
Pumping Lemma (Formal). Let A be a regular language. There is some number p (called the "pumping length") such that all strings $s \in A$ with $|s| \geq p$ can be divided into three substrings $s = xyz$ (path with $|s|+1$ states.) satisfying

1. $xy^iz \in A$, for all $i \geq 0$.
2. $|y| > 0$, // nontrivial.
3. and $|xy| \leq p$. // must loop as/before we hit $|Q|+1$ states.

before loop; during loop; after loop.

// I can go around the loop as much as I want.





computational path on s

follow me!

* why is this true for finite languages? Set p = length of the longest string + 1. (vacuous)

To prove this:

Fix an infinite regular language L. Fix a DFA D_L that recognizes it. Pick an arbitrary string of length at least $p = |Q|$, the number of states in the DFA, and infer a loop. (Sipser p. 78).

Next time:

See how to use the pumping lemma.

Reminders: HW2, my O hrs (5:30-6:30)

Videos: Ex 4: DFAs → Reg. Ex.

LR: The Pumping Lemma.

Puzzle: How would you show $L = \{0^k 1^k \mid k \geq 0\}$ is nonregular using the PL?

BELOW: **really** long s = many loops, much confusion.

extra Q's. But: first loop ends at or before $|Q|$ symbols.

"part before the end of the first loop" $\leq |Q|$ symbols

$|xy| \leq p$.

(maybe my string is $s = xyyyz$.)

Then my PL division will set

$x' = x$

$y' = y$

$z' = yyz$

$(0 \cup 1)\emptyset$

$\{0,1\}\{\varepsilon\}$ $\qquad$ $\{0,1\} \cdot \{\}$ $\Rightarrow$ "concatenate any string in $\{0,1\}$ with any string in $\{\}$."

output: $\{0 \cdot \varepsilon, 1 \cdot \varepsilon\}$ $\qquad$ $\rightarrow$ no strings in $\{\}$

$= \{0, 1\}$ $\qquad$ output: $\{\}$

$\emptyset^* = \{\varepsilon\}$???

$\emptyset^* \Rightarrow$ "concatenate k strings picked from the empty set for any $k \geq 0$."

$\{\} \Rightarrow$ output $\{\varepsilon\}$

↑ $k=0$ case.

NFA: $\delta(q_0, 0) = \emptyset$

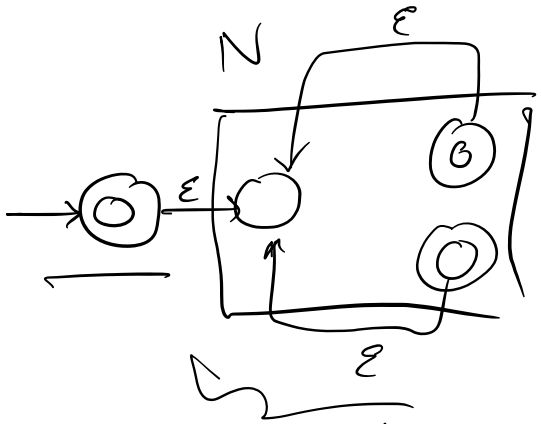
"no states"

"which states do I branch to if I'm in q_0 and I see 0"?

NFA that accepts $\emptyset = \{\}$.

NFA accept rule: accept if at least one branch accepts.

* NFA conversion rule.



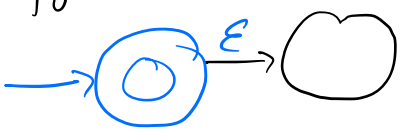
repeat case.

$L(N)^*$

1. ε-transitions from accept to start
2. add a new start state to make sure I accept ε.

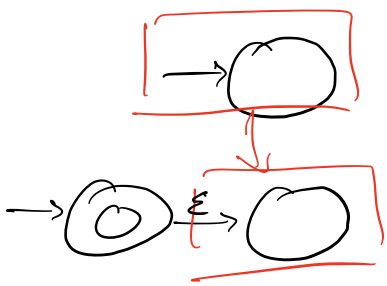
---

apply to



Example: Regex → NFA algorithm.

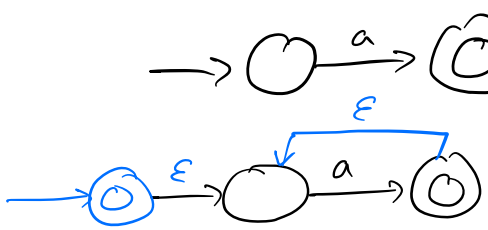
$\emptyset^* = R_1^*$, if $R_1 = \emptyset$



// basic NFA for ∅

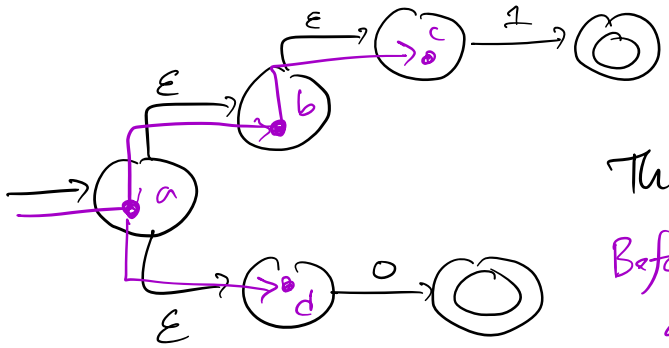
// * rule applied to this

a^*



// basic NFA for a

// * rule applied to this

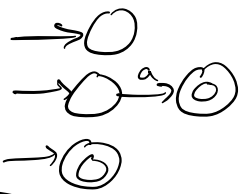


This ~~accepts~~ recognizes $\{0,1\}$

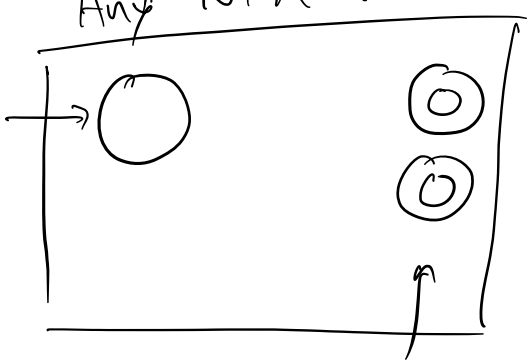
Before reading first char:
always in $\{a,b,c,d\}$

$\varepsilon\varepsilon = \varepsilon$

var x = "" + "";



Any NFA N:



≥0 accept states.

* rule:

(1) Draw $\xrightarrow{\varepsilon}$ from every ◎ to the →○

(2) Add →◎$\xrightarrow{\varepsilon}$ to the start state →○.

This will make an NFA N'
such that $L(N') = L(N)^*$.