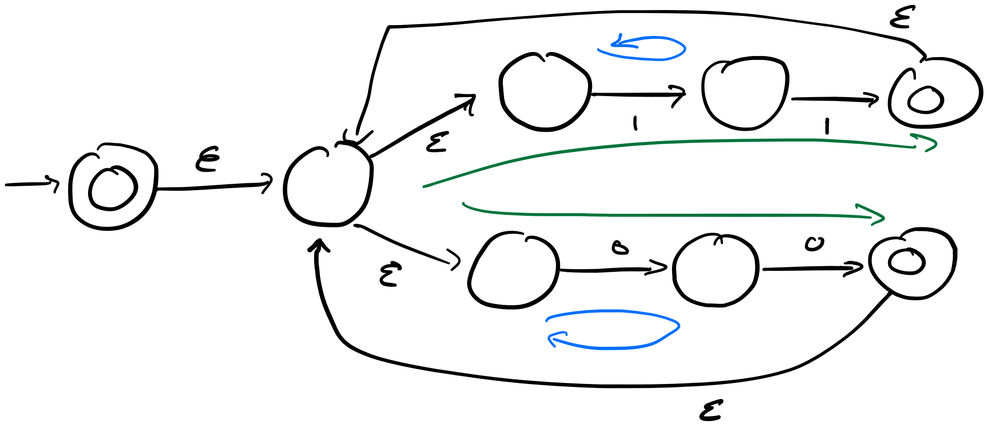
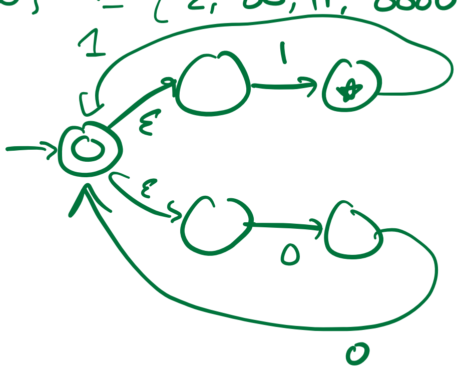
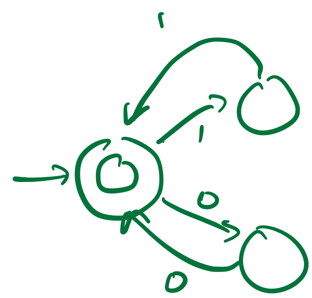


Puzzle: what regular expression is equivalent to this NFA?

Can you find an equivalent 3-state NFA?

- "All strings composed of concatenated 00 and 11 substrings"
- $(\underbrace{11 \cup 00}_{\{11, 00\}^*})^*$ $= \{\varepsilon, 00, 11, 0000, 0011, 1100, 1111, \dots\}$





Note: order of (regular) operations:

$*$, then $\circ$, then $\cup$

$\approx 2^n$

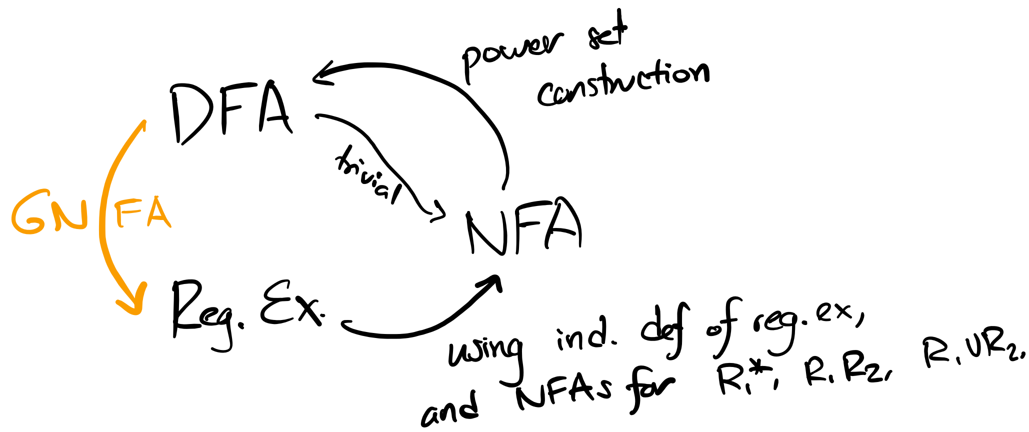
$ab^* \cup c$ $\approx x$ $+$

$(ab)^* \cup c$

$a(b^* \cup c)$

$R^+ = RR^*$

Last time:



Today:

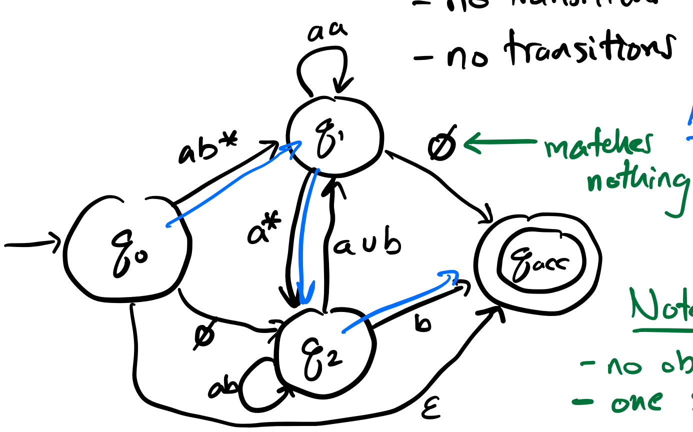
1. DFAs → GNFAs → Reg. Ex.
2. Beyond regular languages: The Pumping Lemma.

(3. More power: context-free grammars)

1. GNFA

NFAs with 3 new rules:

- can label edges (transitions) with any reg. ex.
- exactly 1 start, 1 accept.
- exactly 1 transition between each ordered pair of states, except:
  - no transitions into start.
  - no transitions out of accept.



Accepted: ε

$\underbrace{abbb}_{ab^*}\underbrace{aa}_{a^*}\underline{b}$

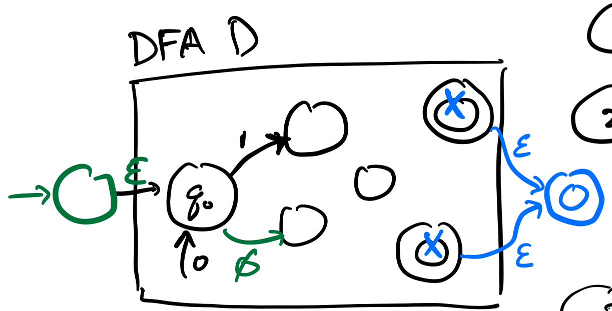
Notes:
- ø marks "dummy edge"
- no obvious "step-by-step" evaluation.
- one string may match multiple accepting branches

As before, accept if and only if some path from q_0 to q_{acc} matches the input string.

DFA → GNFA

Proposition: Every DFA has an equivalent GNFA.

Proof sketch: Begin with an arbitrary DFA, make modifications so that it matches our 3 GNFA rules: (Sipser ____)



① edges labeled with regular expression. ✓

② one start, one accept state. ✓

- create new accept state, connected to the old accept state by ε-transitions.

③ exactly one transition between each ordered pair of states,

- except none into start and
- none out of accept state.

- add $\emptyset$-transitions wherever transitions are missing.
- create a new start state ε-connected to the old one.

Claim: Our new GNFA recognizes the same language. □

Proposition: Every GNFA has an equiv. regular expression.

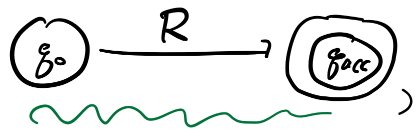
Proof Sketch:

① Pick a state arbitrarily (not q_0, q_{acc}). Call it q_{rip}.

② "Reroute traffic": modify edges so that, for every accepting path that passes through q_{rip}, (corresponding to some string w), w still accepts when q_{rip} is removed.

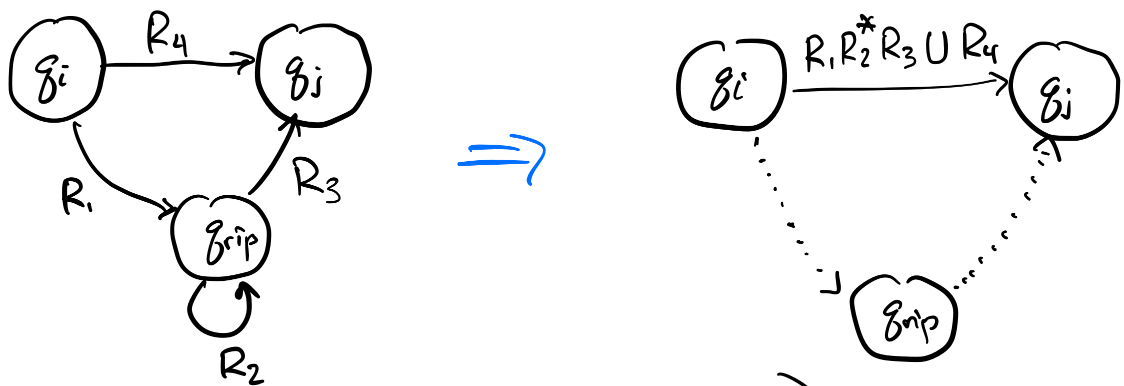
(3) Remove q_{rip}: new, equivalent GNFA with one fewer states.

(4) Repeat until we boil down to a GNFA of the following form:



, for some reg. ex. R.

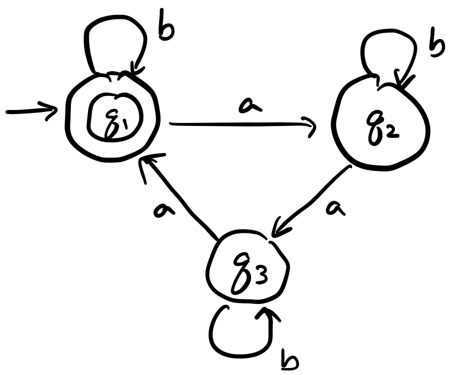
Step 2, in more detail: Suppose some accepting string w has an accepting computational branch that goes from $q_i \to q_{rip} \to q_j$, for some pair (q_i, q_j).



(Repeat for all valid pairs (q_i, q_j).)

Claim: the exact same set of substrings can transition $q_i \to q_j$ as used to transition $q_i \to q_{rip} \to q_j$. $\square$

<u>Example</u>: DFA $\to$ Reg. Ex.

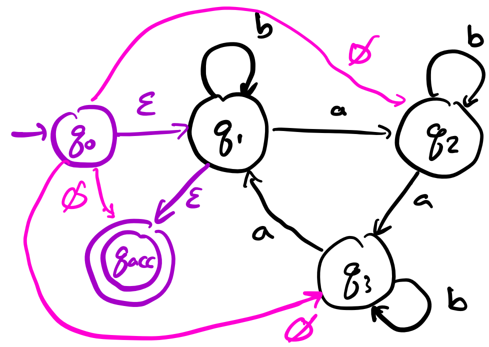


$L(D) = \{ x \in \{a,b\}^* \mid$ the number of a's is divisible by $3\}$.

1. DFA → GNFA.

- create new start, accept states.
- add $\emptyset$-transitions wherever necessary.

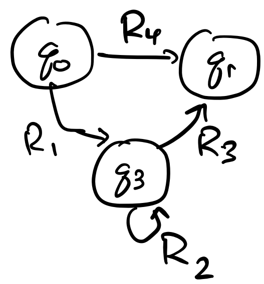
(shown: $\emptyset$-transitions from q_0. others omitted.)



2. GNFA → Reg. Ex

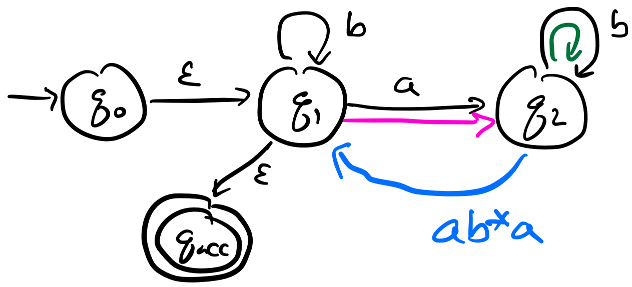
2a) Ripping out state q_3

- say, $(q_i, q_j) = (q_0, q_1)$
  $R_1 R_2^* R_3 \cup R_4 = \underline{\emptyset} b^* a \cup \varepsilon = \varepsilon$



- $(q_i, q_j) = (q_2, q_1)$
  $R_1 R_2^* R_3 \cup R_4 = ab^* a \cup \emptyset = ab^* a$

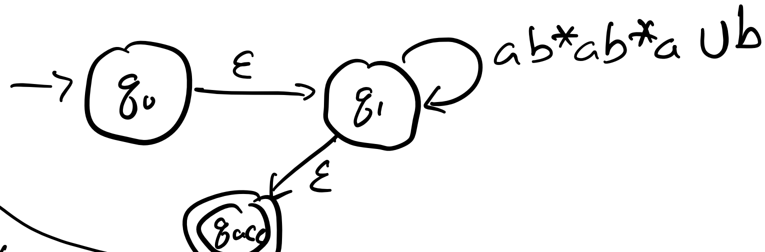
Now:



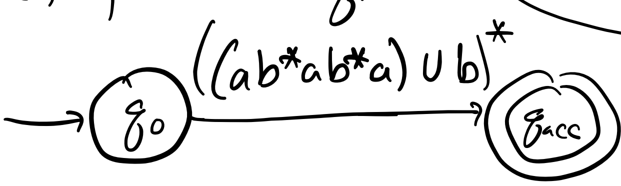
2b) rip out state q_2:

set $(q_i, q_j) = (q_1, q_1)$:

$R_1 R_2^* R_3 \cup R_4$: $a b^* (ab^* a) \cup b$



2c) rip out state q_1:



$L(D) = \{w \in \{a,b\}^* \mid w$ has a number of a's divisible by $3\}$.

Punchline: Languages recognized by DFAs, NFAs, and Reg. Ex's are exactly the reg. languages.

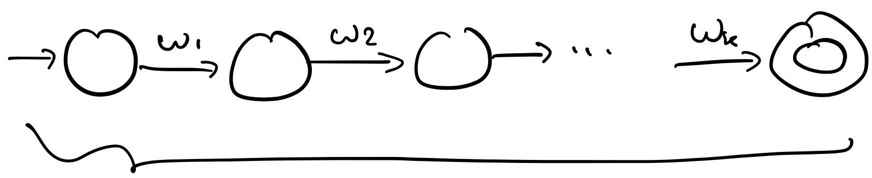
Back at 2:20

Q: Are all finite languages regular?

finite sets of strings.

$A = \{0^n 1^n \mid n \geq 0\}$

For any string $w = w_1 w_2 \cdots w_k$, w_i's in Σ, $\{w\}$ is regular:

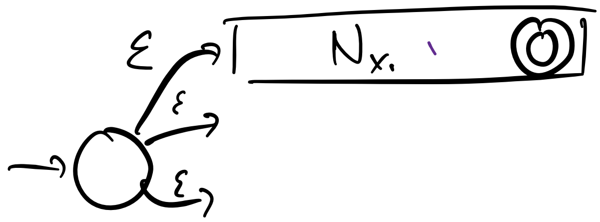


For any finite language

$B = \{x_1, x_2, \cdots x_\ell\}$, x_i's are strings in Σ^*.

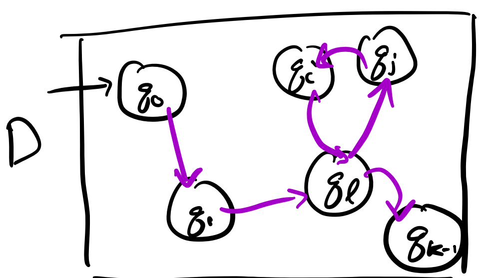
$B = \bigcup_{i \leq \ell} \{x_i\}$. B is regular by closure under union.

$\therefore$ all finite languages are regular.



Infinite regular languages:

- Let D be some DFA, and $L(D)$ be some infinite, regular language.
- Suppose D has k states.



(*) If $w \in L(D)$ is a string of length $\geq k$, then the computational path taken by D on k must touch some state twice.

In other words: any regular infinite language "creates loops" in its DFA on long strings.

We'll prove:

"no loops" $\longrightarrow$ not regular.

Pumping Lemma: Let A be an ~~infinite~~* regular language. There exists some number p (the "pumping length") such that all strings $w \in A$, $|w| \geq p$ can be divided into three substrings

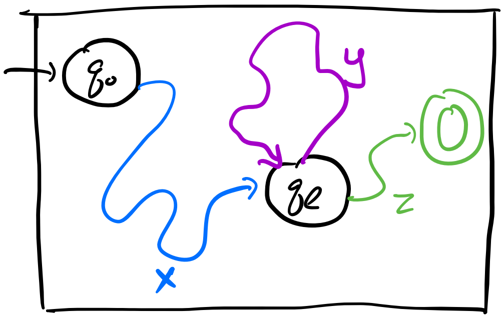
$$w = \underbrace{x}_{\text{pre-loop}}\ \underbrace{y}_{\text{"the loop"}}\ \underbrace{z}_{\text{post-loop}}$$

satisfying (1) $xy^iz \in A$ for all $i \geq 0$, // we can go around the loop i times for any i, and still accept

(2) $|y| > 0$, // non-trivial

(3) $|xy| \leq p$. // we see a loop before we touch p states

DFA D on ω, with $|\omega| \geq p$



$xyz = \omega$

* trivially true if A finite

Proof (w/reference to picture.)

Let A be an infinite regular language, and let D_A be a DFA w/ $L(D_A) = A$. Set $p = |Q_A|$, the number of states in D_A.

Let $s \in A$ be a string with $|s| \geq p$, and consider the sequence of $|s|+1$ states through which we travel from q_0 to an accept state on s.

Because $|s|+1 > p = |Q_A|$, we see some state twice in this sequence. Call this state q_{loop}, and divide s into x, y, and z such that x is the sequence of characters that takes us to q_{loop}, y is the sequence of characters that takes us $q_{loop} \to q_{loop}$, and z is the remainder of s.

Now: $xy^iz \in A$, as for any $i \geq 0$ xy^iz takes us to the same accept state along the same sequence of states, with the loop repeated i times.

$|y| > 0$ trivially,

$|xy| \leq p$ because we must see q_{loop} twice before we see $p+1 = |Q_A|+1$ states. ∎

Example.

$$L = \{0^n 1^n \mid n \geq 0\}$$

ε, 01, 0011, 000111, ...

Proof. L is nonregular.

- Assume for contradiction that L is regular.
- $\therefore$ L satisfies the PL, and there exists some number p such that for all $s \in L$, with $|s| \geq p$, s can be subdivided into

$s = xyz$, with

(1) $xy^iz \in L$ for $i \geq 0$
(2) $|y| > 0$
(3) $|xy| \leq p$.

- Choose a long "contradiction string" $s \in L$, and show that it can't be subdivided in a way satisfying (1), (2), (3).
- Choose $s = 0^p1^p$.
- If (2), (3) hold in a subdivision $s = xyz$, then y is all 0's, and consists of at least one 0.
- But in this case, if $i = 2$, then

$$xy^2z = xyyz = 0^{p+|y|}1^p \in L \text{ by (1).}$$

- But $0^{p+|y|}1^p$ is not in L for $|y| > 0$. ✗ contradiction
- Contradiction: thus L does not satisfy the PL, and my assumption of regularity is false. $\square$

Back at 3:20

Some (nonregular) languages (to prove):

$C = \{0^k 1 0^k \mid k \geq 0\}$. $\quad 0^p 1 0^p \quad 0^{\lceil p/2 \rceil} 1 0^{\lceil p/2 \rceil}$

$D = \{0^i 1^j \mid i \geq j\}$. $\quad 0^p 1^p \quad 0^{p-1} 1^{p-1} \quad 0^{p+3} 1 0^{p+3}$

$E = \{0^{k^2} \mid k \geq 0\}$. $\quad 0^{p^2}$

Goal: pick a contradiction string, in the language, length $\geq p$, and show <u>no way</u> of dividing into xyz satisfies

(1) $xy^iz \in L, \ i \geq 0$

(2) $|y| > 0$,

(3) $|xy| \leq p$.

C: Any way to divide $0^p 1 0^p$ into xyz?
If (3), (2) are true $\Rightarrow$ y is all 0's.
$xyyz$ looks like $0^{p+|y|} 1 0^p \notin C$.

D: Any way to divide $0^p 1^p$ into xyz?
If (3), (2) are true $\Rightarrow$ y is all 0's
- $xyyz$ looks like $0^{p+|y|} 1^p$
- xy^3z looks like $0^{p+2|y|} 1^p$

$\in D$.

For $0^{p-1} 1^{p-1}$, $xyyz$ is (maybe) $0^{p+|y|-1} 1^{p-1}$

take $i=0$. $xy^0z = xz = 0^{p-|y|} 1^p \notin D$.

E: Any way to divide 0^{p^2} into xyz.
If (3), (2) both hold, y is all 0's and at least one 0.

$xy^2z = xyyz = 0^{p^2+|y|}$

$\underbrace{000}_{x}\underbrace{00}_{y}\underbrace{00}_{z}$ $|xyz| = p^2$

$p^2 + |y| > p^2$, as $|y| > 0$ by (2).

$\underbrace{000}_{x}\underbrace{000}_{y}\underbrace{000}_{y}\underbrace{00}_{z}$ $|xyyz| = p^2+|y|$

$p^2 + |y| \leq p^2 + p$, as $|xy| \leq p$ by (3).

$p^2 + p < p^2 + 2p + 1 = (p+1)^2$.

$\therefore$ $p^2 + |y|$ is not a square number, and $xy^2z \notin E$.

---

3. Context-Free Grammars

Informally: Given a start variable (S) and a set of substitution rules that let us swap variables for other symbols.

The *language* of a CFG is the set of all strings we can make by applying substitution rules.

$$G \begin{cases} S \longrightarrow 0S1 \\ S \longrightarrow \varepsilon \end{cases}$$

$S \to \varepsilon$

$S \to 0S1 \to 0\varepsilon 1 = 01$

$S \to 0S1 \to 00S11 \to 000S111 \to 000111$

$L(G) = \{0^n 1^n \mid n \geq 0\}$

<u>Def. (CFG).</u> A CFG is a 4-tuple (V, Σ, R, S), where V is a finite set of variables,

Σ is a finite set of terminals ("terminal alphabet"),

R is a finite set of substitution rules, each mapping one variable to a sequence of variables and terminals,

S is the start variable.

Given a CFG G, we derive a string by beginning with the start variable and substituting until only terminals remain.

$F = \{ww^R \mid w \in \{0,1\}^*\}$

0110

100001

101101

$$G_F: \quad \begin{aligned} S &\to 0S0 \\ S &\to 1S1 \\ S &\to \varepsilon \end{aligned}$$

$S \to 1S1 \to 10S01 \to 101S101 \to 101101$

Shorthand: $S \to 0S0 \mid 1S1 \mid \varepsilon$

"Grammar"?

$S \to N_p\ V_p$

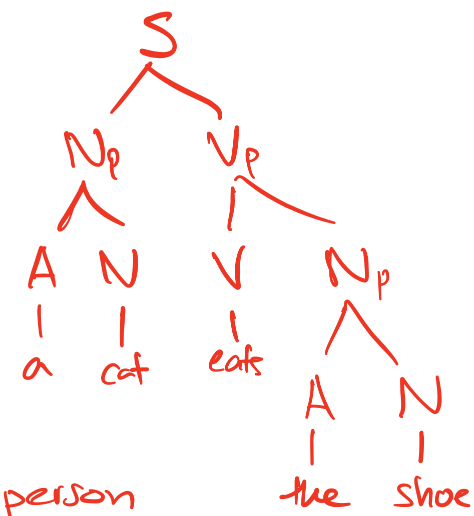
$N_p \to A N$

$V_p \to V \mid V N_p$

$V \to \text{sees} \mid \text{eats} \mid \text{smells}$

$N \to \text{dog} \mid \text{cat} \mid \text{shoe} \mid \text{person}$

$A \to \text{a} \mid \text{the}$



$N_p \to A\ A_{dj}\ N$

CFG tricks:

Union:

CFG for $L_A \cup L_B$.

$S \to A \mid B$

$A \to$ //stuff to generate L_A

$B \to$ //stuff for L_B.

Concat:

CFG for $L_A L_B$

$S \to AB$.

Star:

CFG for L_A^*

$S \to AS \mid \cancel{A} \mid \varepsilon$.

$S \to SS \mid$ other rules for S

Today:

- The PL says that every suff. long string in a regular language has a **loop**.
- If we show that a long string has no loops, language isn't regular.
- CFGs: generate strings by swapping variables.
- Languages recognized by the CFGs: CFLs.

Reminders:

- HW2 due tonight.

Next time: automata w/ stack memory!