Teaser: What is the language described by the regular expression

$$3(134 \cup 203 \cup (2(5 \cup 6)1))^?$$

(Decimal alphabet.)

$$L = \{3134, 3203, 3251, 3261\}$$

COMS 3261 ∈ L - CS Theory

Lecture 4: NFAs → regular expressions
Nonregular languages &
the pumping lemma.

Announcements: HW #2: due tonight @ 11:59PM EST
HW #3: due 7/19, 11:59PM EST.

Readings: Sipser 1.3, 1.4

Today:

1. Review
2. (We showed any language described by a regular expression has an equivalent NFA)
   ↳ Complete proof that a language is

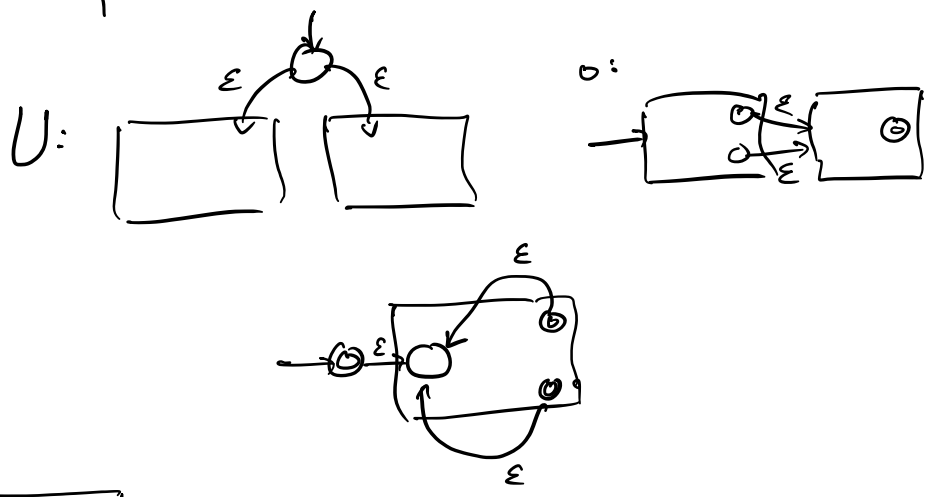
regular if and only if it is described by some regular expression.

3. Nonregular languages
4. The "pumping lemma"

1. Review

1. Regular $\iff$ recognized by some DFA $\iff$ recognized by some NFA.

2. Regular languages closed under the regular operations $\cup$, $\circ$, $*$:



Fact: The regular languages are closed under the set complement operation, written $\overline{A}$ for the language A.

Def. If A is a language on Σ, then $\overline{A} := \Sigma^* \setminus A$

Proof sketch. If A is a regular language, let D be a DFA that recognizes A. Swapping accept and reject states gives us a new DFA recognizing $\overline{A}$.

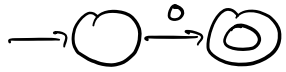
3. Regular expressions describe/evaluate to languages. Built up using the regular operations $(\cup, *, \circ)$.

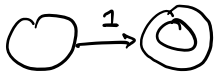
Example. $(1 \cup 0)$, $\{1\} \cup \{0\} = \{1, 0\}$

$(01)^*$, $(\{0\} \circ \{1\})^* = \{\varepsilon, 01, 0101, \ldots\}$

4. Any regular expression can be converted to an NFA.

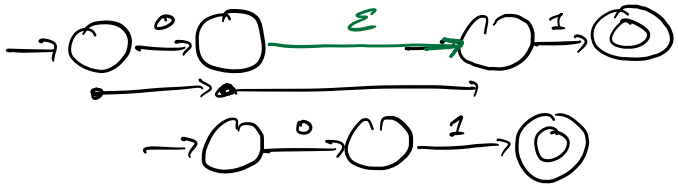
Example. $(01)^* 1 \cup \varepsilon$

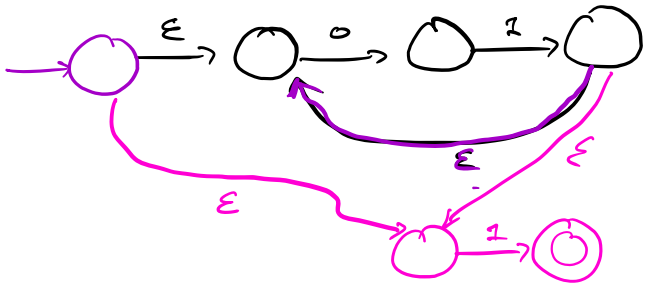
0: 

1: 

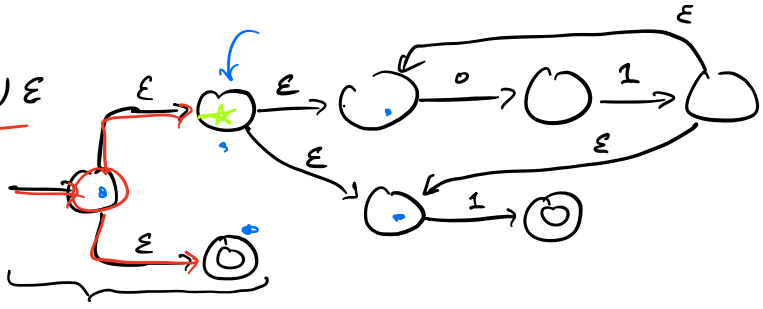
ε: 

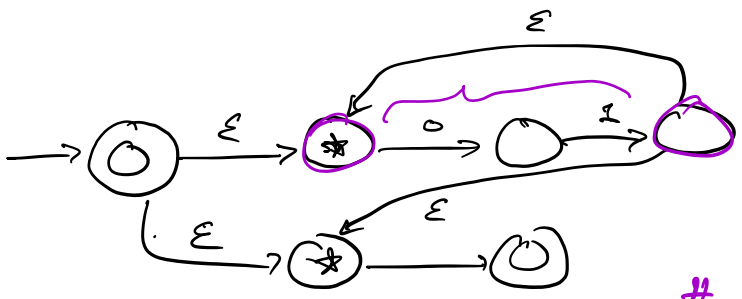
$\emptyset$: 

01: 

$(01)^*$: 

$(01)^* \circ 1$

$(01)^* 1 \cup \varepsilon$ 



01

2. DFAs ⟶ Regular Expressions.

Theorem: A language is regular if and only if some regular expression evaluates to the language.

Lemma 1: Any regular expression has an equivalent NFA. ✓ proved. (reg. exp. → NFA → regular)

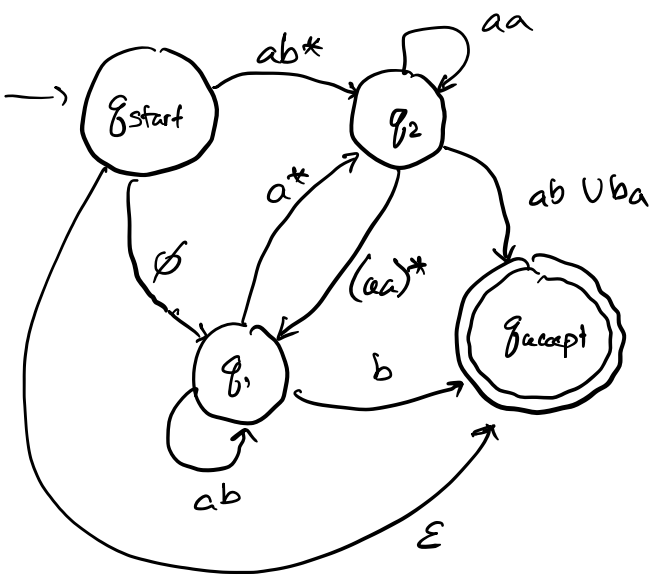
Lemma 2: Any DFA has an equivalent regular expression. (Reg → DFA → reg. exp.)

Lemma 2 proof idea:

1. Start with any DFA
2. Convert our DFA to a new kind of automaton called a GNFA (Generalized Nondeterministic Finite Automaton), which has a convenient structure.
3. Convert ("boil down") a GNFA into an equivalent regular expression.

GNFA: Like an NFA, with regular expressions as edge labels.

$\Sigma = \{a, b\}$



$(Q - \{q_{accept}\}) \times (Q - \{q_{start}\})$

Special rules: Exactly one start state q_{start} and exactly one accept state q_{accept}. Exactly one transition between each ordered pair of states, except the start state (no incoming transitions) and the accept state (no outgoing transitions.)

Definition (Formal definition of a GNFA.) A GNFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where

Q is a finite set of states,

Σ is a finite input alphabet,

q_{start} and q_{accept} are the start and accept states,

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

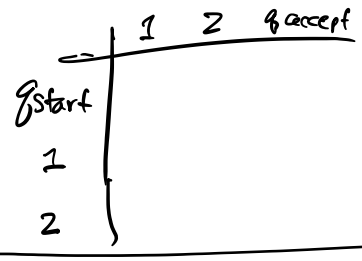
$\mathcal{R}$ is the set of all regular expressions on Σ.

A GNFA accepts a string $w = w_1 w_2 \cdots w_k$, where each $w_i \in \Sigma^*$, for $i \in [k]$, if there exists a sequence of states $q_0, q_1, \ldots, q_k$ such that

$q_0 = q_{start}$, $q_k = q_{accept}$, and

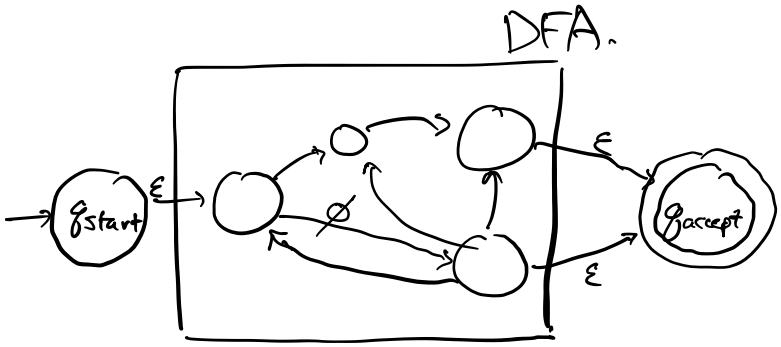
for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

Q: why $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to \mathcal{R}$?



So: defined a GNFA.

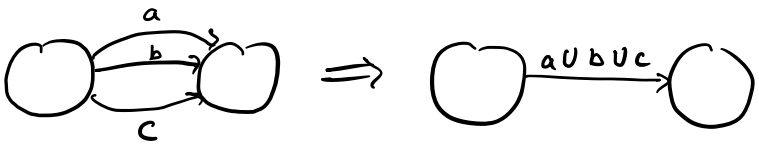
Now: How to convert a DFA into an equivalent GNFA.



Conversion procedure:

1. Create a new start state w/ no incoming edges, connected by an ε-edge to the old start state.
2. Create a new accept state w/ ε-arrows from the old accept states. (old accept states → regular states.)
3. Add $\emptyset$-arrows between any ordered pair of states w/ no transition (except transitions to the start state or from the accept state.)
4. If there are multiple transitions between two

states, merge them with a union ∪.



Claim: this GNFA now meets our requirement that there is exactly one transition between each ordered pair of states.

Now: we can convert a DFA to a GNFA.

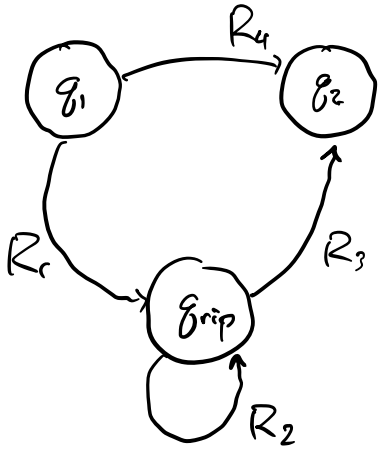
Proof structure: Regular language ✓→ recognized by DFA ✓→ equivalent GNFA ↓ ? equivalent regular expression.

Break: back at 11:23

Idea: Given a GNFA, remove one state at a time by combining transitions using regular expressions. Once we have a 2-state GNFA, we're done.

R_1 = language recognized by this GNFA.

How do we remove states? Consider any pair of states q_1 and q_2, and some third state q_{rip} to be "ripped out."



Goal: remove q_{rip} and replace R_4 with a new regular expression that captures all strings that could go from q_1 to q_2 through q_{rip}

If we do this for all state pairs not including q_{rip} (also not starting w/ q_{accept} or ending w/ q_{start}.) we have an equivalent GNFA with one fewer state.

Lemma 2. Any DFA has an equivalent regular expression.

Proof. Let D be some DFA. First, we convert D to a GNFA $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$ using the procedure sketched above:

1. Add new start and accept states with no incoming edges for the start state and no outgoing edges for the accept state.
2. Add dummy $\emptyset$-edges as necessary.
3. Merge multiple edges by using a union ($\cup$).

[(q_1, q_2) and (q_2, q_1) are different ordered pairs.]

Second, we repeatedly replace G with an equivalent GNFA G' that has one fewer state, using the following procedure:

CONVERT(G):

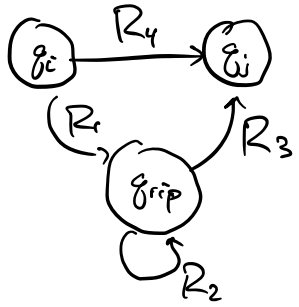
- Let $k = |Q|$ be the number of states in G.
- If $k > 2$, select a state $q_{rip} \neq q_{start}, q_{accept}$.

Let $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$, such that:

$$Q' = Q \setminus \{q_{rip}\}$$

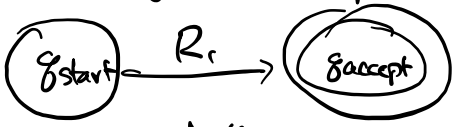
For each $(q_i, q_j) \in (Q - \{q_{accept}\}) \times (Q - \{q_{start}\})$, define $\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4$, where

$R_1 = \delta(q_i, q_{rip})$,
$R_2 = \delta(q_{rip}, q_{rip})$,
$R_3 = \delta(q_{rip}, q_j)$,
$R_4 = \delta(q_i, q_j)$



Then return G'.

- If $k = 2$, return $\delta(q_{start}, q_{accept})$

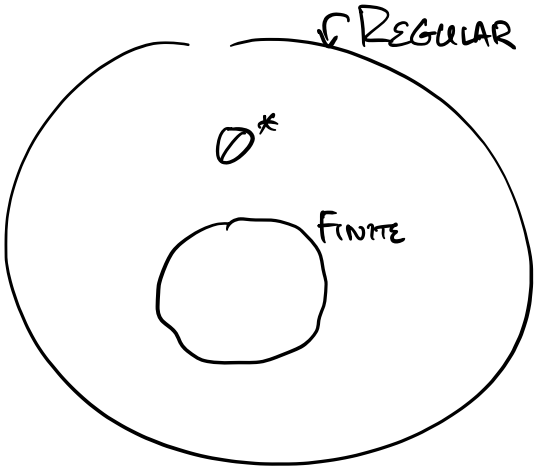


Now: after running CONVERT repeatedly, we eventually get a regular expression R_1 equivalent to G, which is equivalent to D. ∎

Punchline: A language is regular if and only if some regular expression evaluates to it.

3. Nonregular Languages

Regular $\iff$ DFA recognizes $\iff$ NFA recognizes
$\iff$ regular expression describes.



NONREGULAR
(???)

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

Proposition: All languages with finite size are regular

Proof sketch: Any finite language is a finite union of languages with a single string. We can recognize any single string and then use closure under union.

$$\{0, 1, 11, \varepsilon\} = \{0\} \cup \{1\} \cup \{11\} \cup \{\varepsilon\}$$

Examples. $B = \{0^n 1^n \mid n \in \mathbb{N}_{\geq 0}\}$
(will turn out nonregular)

$D = \{w \mid w$ has an equal number of '0x' and '1x0' substrings$\}$
(regular)

x x x 0 0 x 0 x x x 0 0 x 0 x

Break: Back at ~ 12:05, 12:06.

---

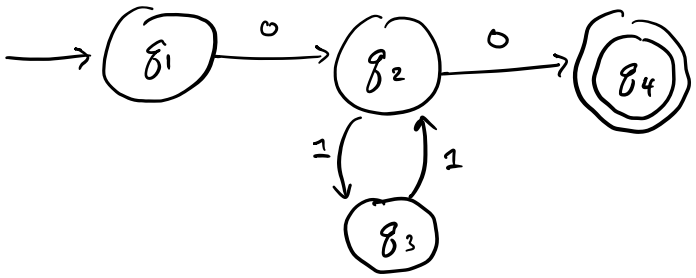
4. The "pumping Lemma"

math speak for 'not intuitive'

Idea: come up with a 'technical' property that all regular languages have. Thus if a language doesn't have this property, it must not be regular.

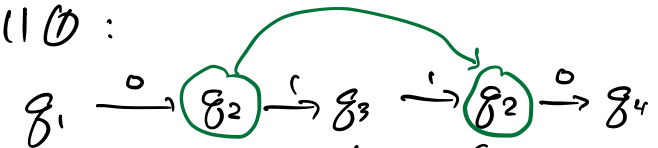
Example of an infinite regular language.

$$0(11)^*0 = \{00, 0110, 011110, 01111110, \ldots\}$$



This NFA recognizes $0(11)^*0$. It has $|Q|=4$ states. Any accepting string with at least 4 symbols passes through at least 5 states, on an accepting branch.

Example 0110:



And thus I visit the same state twice (I make a loop.) Moreover, any loop can be repeated an arbitrary number of times to make an accepting computation for some new string.

Example: $q_1, q_2, q_3, q_2, q_3, q_2, q_4$: 01110

q_1, q_2, q_4 : 00

$q_1 q_2 q_3 q_2 q_3 q_2 q_3 \cdots q_2 q_4$:

0 1 1 1 1 ... 0

Core insight: Automata for infinite regular languages must have "a loop." Repeating the loop must create more state sequences that correspond to accepting computations.

Lemma (The pumping lemma). If A is a regular language, then there is some number p ("the pumping length") such that, for any string $s \in A$ with length $|s| \geq p$, s can be divided into three pieces, $s = xyz$, satisfying

1. for each $i \geq 0$, $xy^iz \in A$, } go around the loop i times.
2. $|y| > 0$ } = nontrivial nontrivial loop
3. $|xy| \leq p$. } after p symbols, we repeat a state.

Proof idea: consider the DFA of a regular language and show that all sufficiently long strings have a loop.

Proof. Let A be a regular language and $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing A. Let $p = |Q|$ be the number of states in D.

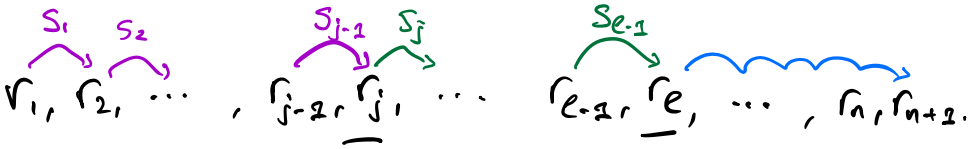
Now, let s be any string of length $n \geq p$ that is accepted by D. By the definition of DFA acceptance, there exists a sequence of states

$$r_1, r_2, \cdots, r_{n+1}$$

that satisfies $r_1 = q_0$, $r_{n+1} \in F$, and each intermediate

state satisfies $r_{i+1} = \delta(r_i, s_i)$, where $s = s_1 s_2 \cdots s_n$.

Because we have $n+1 > n \geq p$ states in our sequence, two states must be the same. Call these states r_j and r_ℓ.



Now: let $x = s_1 s_2 \cdots s_{j-1}$, $y = s_j s_{j+1} \cdots s_{\ell-1}$, and $z = s_\ell \cdots s_n$. Now x takes our DFA computation from $q_0 = r_1$ to r_j, y takes us from r_j back to r_j, and z takes us from r_j to an accept state r_{n+1}. Thus:

xy^iz accepts for all $i \geq 0$ // going around the loop i times.

$|y| > 0$, as $j < \ell$.

$|xy| \leq p$, as $\ell \leq p+1$. // we see a duplicate state in the first $p+1$ states.

These are precisely the conditions of the pumping lemma. ∎

Reminders: HW 2 due today
HW 3 due Monday
Readings: end of Sipser 1.3, 1.4.

Next time: PL examples, on to bigger and better language classes

---

Q: 0 hours for Latex?

\usepackage{tikz}

7.6) Set X (alphabet)
Set Y

XY indicates exactly one symbol from X, followed by exactly one symbol from Y.

$D = \{0, 1, \cdots, 9\}$

$D^+ . D^+$ Σ

↳ 079.623

$X = \{0, 1, 2\}$ $Y = \{3, 4\}$

$XY = \{03, 04, 13, 14, 23, 24\}$

capital X, capital Y, sets. defined to stand for

$X = \{0\} \cup \{1\} \cup \{2\}$

$\Sigma = \bigcup_{x \in \Sigma} \{x\}$

$pre(A) = \{ xy \mid x \in A, y \in \Sigma^* \}$

↑ lowercase x and y, defined to stand for strings.

× 'times symbol'

X

pre(A). pre: languages → languages.

$a+b$: $+(a,b)$

$\int(a,b)$

$A \cup B$: $U(A,B)$

A^* : $*(A)$

pre(A):

pre operation: takes any language and outputs a new language.

want to show: If A is regular, pre(A) is regular.