COMS W3261 – Computer Science Theory

Lecture 4: Convert NFAs → Regular Expressions
Nonregular languages & the pumping lemma.

Teaser: $3(134 \cup 203 \cup (2(5 \cup 6)1))$

What language is this?

Announcements: HW #2: 7/12 @ 11:59 PM EST

HW #3: 7/19 @ 11:59 PM EST.

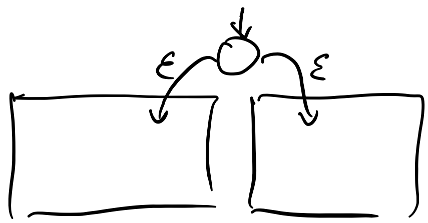
Readings: Sipser, end of section 1.3; section 1.4.

Today:

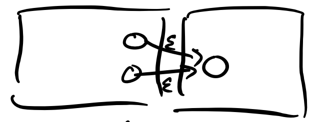
1. Review
2. (We showed Regular Expression → NFA.)
   Will show that NFA → Regular Expression.
   (Complete proof that Reg. Expression ↔ NFA.)
3. Nonregular Languages.
4. The "pumping lemma."

1. Review:

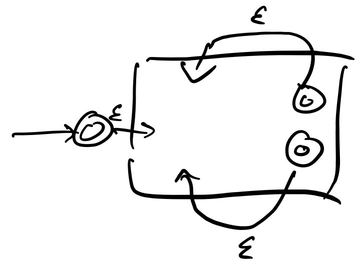
- Regular languages ⟷ recognized by some DFA
  ⟷ recognized by some NFA
- Regular languages are closed under $\cup$, $\circ$, $*$:



(∪)



(∘)



(*)

(Note: Regular languages are closed under <u>set complement</u>. Suppose we have Σ. Any language over this alphabet is a subset of Σ^*. Set complement:

$$\text{For } A, \quad \overline{A} = \Sigma^* \setminus A.$$

<u>Proof:</u> swapping the accept, reject states of a DFA that recognizes A ensures we recognize $\overline{A}$.)

3. Regular expressions describe/evaluate to languages. We use the regular operations to build them up from base cases.

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

$(01)^* \Rightarrow \{\varepsilon, 01, 0101, 010101, \dots\}$

$(0 \cup 1)1 \Rightarrow \{01, 11\}$

Σ, $+$: $R^+ = RR^*$, $\varepsilon \Rightarrow \{\varepsilon\}$, $\emptyset \Rightarrow \{\}$.

Order of operations clarification:

0) parentheses.

1) star and plus. (R^+ is just shorthand for RR^*)

2) concatenation

3) union

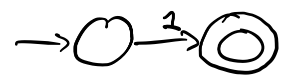
$a + b \times c = (b \times c) + a$.

$(a+b) \times c =$

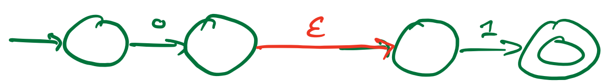
4. Any regular expression can be converted to an NFA.

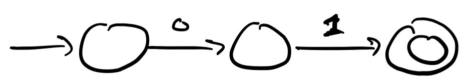
Example: $(01)^* 1 \cup \varepsilon$.

0: 

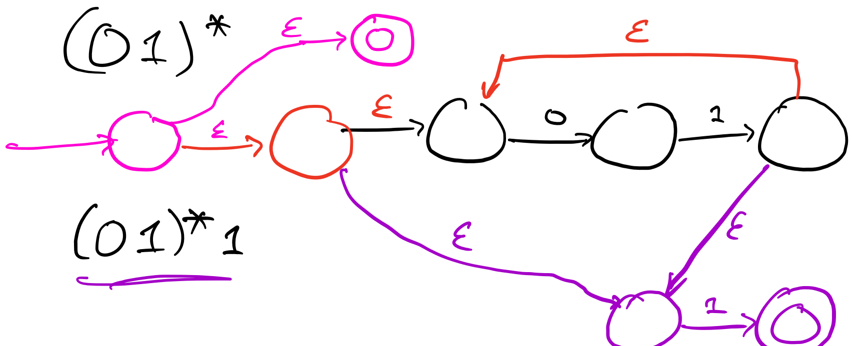
1: 

ε: 

01: 

= 

$(01)^*$



$(01)^* 1$

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

2. DFAs → Regular Expressions

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

Follows from two lemmas:

Lemma 1: ($\Leftarrow$). Any regular expression has an equivalent NFA.

✓ Proved last time.

Lemma 2: ($\Rightarrow$). Any DFA has an equivalent regular expression.

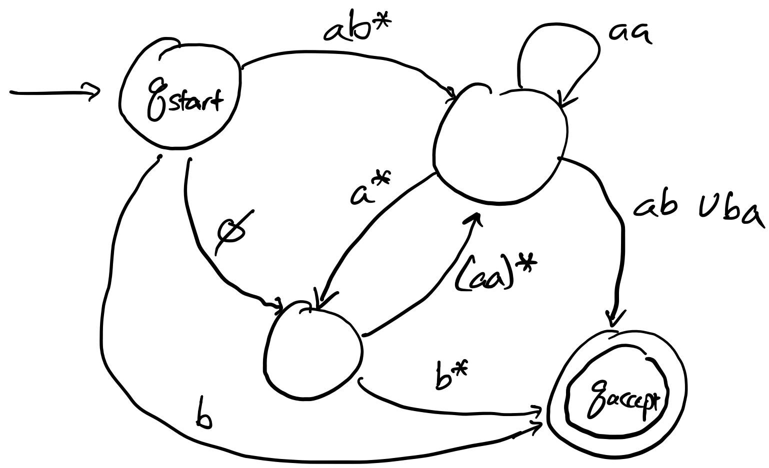
Proof Idea: 1. Start with some DFA.

2. Convert our DFA to a new kind of automaton, called a GNFA (Generalized NFA), which has a convenient structure.

3. "Boil down" our GNFA to an equivalent regular expression.

GNFAs — like NFAs, but with transitions labeled by regular expressions.

Picture of a GNFA: (On $\Sigma = \{a, b\}$):



Special rule: Exactly one start/end state, q_{start} and q_{accept}.
(Exactly) One transition between every ordered pair of states, except the start state (no incoming transitions) and the accept state (no outgoing transitions.)

Def. (GNFA, formally.) 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,

$\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \longrightarrow \mathcal{R}$,
where $\mathcal{R}$ is the set of all regular expressions on Σ.

q_{start} and q_{accept} denote the start and accept states.

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

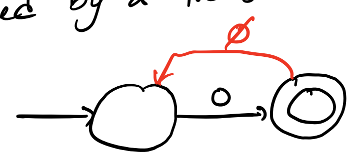
$q_0 = q_{start}$, $q_k = q_{accept}$, and

for each $i \in [k]$, we have $w_i \in L(R_i)$, where $R_i = \underline{\delta(q_{i-1}, q_i)}$.
(That is, R_i is the regular expression on the arrow from q_{i-1} to q_i.

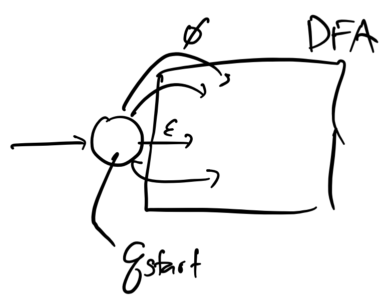
Now: How to convert a DFA into an equivalent GNFA.

Let D be a DFA.

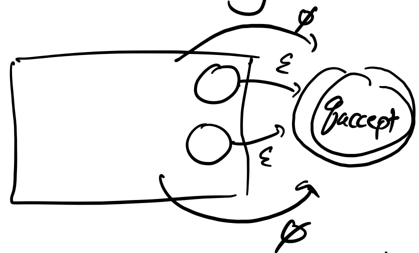
1. Add $\emptyset$ arrow between any ordered pair of states in our DFA not linked by a transition.



2. Create a new start state with no incoming edges, connected by an ε-arrow to the old start state (and by $\emptyset$-arrows to other states.)

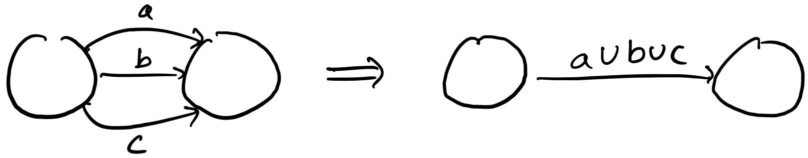


3. Create a new end state q_{accept} w/ ε-arrows from the old end states (and incoming $\emptyset$ transitions)

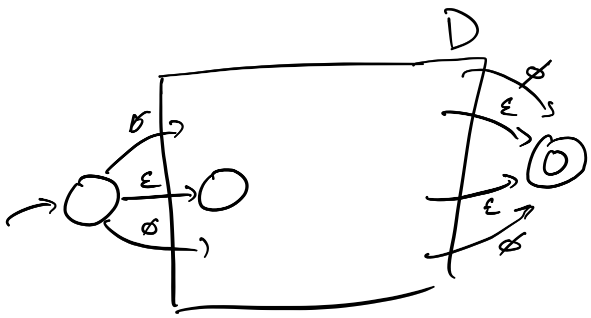


4. If there are multiple transitions between any ordered pair

of states, merge them with union ($\cup$).



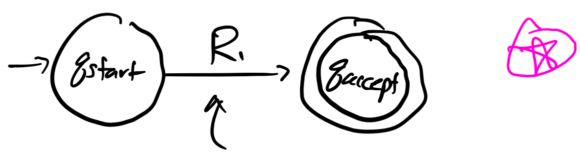
Result:



Now we have a GNFA: one arrow between each ordered pair of states, except the start state (no incoming) and the accept state (no outgoing.)

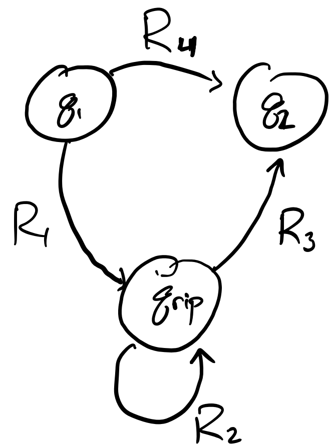
Last step: "boil down" a GNFA into an equivalent regular expression.

Idea: Given any GNFA, we can remove one state at a time by combining transitions using regular expressions. Once we have a two-state GNFA, we are done.

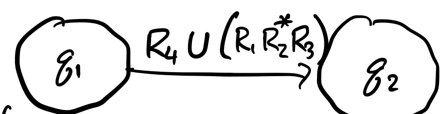


R_1 evaluates to the language recognized by this GNFA.

How do we remove a state? Consider any pair of states q_1 and q_2, and a third state q_{rip} that we want to "rip out" of our GNFA.



Goal: remove q_{rip} and replace R_4 with a new regular expression that captures all strings that might have gone from $q_1 \to q_2$ through q_{rip}.



Claim: these two pictures are equivalent. If we perform this replacement operation for all state pairs (q_1, q_2) not including q_{rip}, we have an equivalent GNFA with one fewer state.

Put it all together.

Lemma 2. Any DFA has an equivalent regular expression.

Proof. Let D be any 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/accept states with no incoming/outgoing edges, connected to the old start/accept states with ε-edges.
2. Add dummy $\emptyset$-edges where necessary.
3. Merge multiple edges between the same two states using union ($\cup$).

Second, we repeatedly replace G with an equivalent GNFA G'

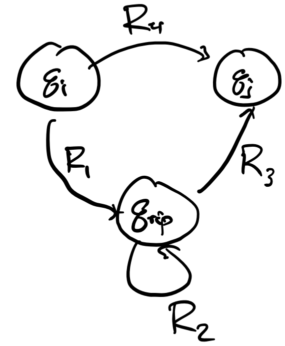
that has one fewer state, using the following procedure CONVERT(G):

CONVERT(G):

- Suppose G has $|Q| = k$ states.
- If $k > 2$, select some state $q_{rip} \neq q_{start}, q_{accept}$.
- Let $G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$ be a new GNFA such that $Q' = Q - \{q_{rip}\}$.

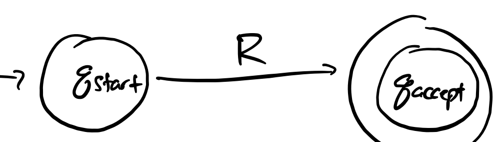
For each $(q_i, q_j) \in Q \times Q$ such that $q_i \neq q_{accept}$ and $q_j \neq q_{start}$ define:

$$\delta'(q_i, q_j) = R_1 R_2^* R_3 \vee R_4,$$



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

- Then return G'.
- If $k = 2$, G looks like this:



We return R. $\blacksquare$

Thus we return R that evaluates to the language recognized by G, which is the language recognized by D.

<u>Punchline</u>: A language is regular if and only if some regular expression evaluates to it. DFAs, NFAs, and regular expressions recognize/describe the same class of languages (maybe with more or less states.)

Regex. $\longrightarrow$ NFA.

DFA $\longrightarrow$ GNFA $\longrightarrow$ Reg. Ex.

<u>Next up:</u> Thinking about languages that are <u>not regular</u>.