Finishing Up GNFA
Review: GNFAs Generally

A GNFA is a special kind of NFA that uses regular expressions as its *transition alphabet*
- A GNFA has a single start state and a single accept state
- Nothing can transition *into* the start state, and nothing can transition *out of* the accept state

First, we convert our DFA to a GNFA
- This is the easy part

We then convert that GNFA to a regular expression by *state ripping* and *repair*
- One by one, we remove states from the GNFA, or *rip* the states out
- After each rip, we expand the expressions on the transitions surrounding the removed state, so that the GNFA still recognizes the same language

We know we’re done when there are only two states left—the start and accept states
- ...and the transition regular expression between them has to be the regular expression recognizing the original language
Review: Making the GNFA

First, we:

- Add specific start and accept states
- Add an empty-string transition from the start state to the old start state
- Add empty transitions from the old accept states to the accept state
- Convert all the multiple-symbol transitions to use the union operator
Review: Making the GNFA

First, we:
- Add specific start and accept states
- Add an empty-string transition from the start state to the old start state
- Add empty transitions from the old accept states to the accept state
- Convert all the multiple-symbol transitions to use the union operator
Review: Making the GNFA

Now we rip out a state
  ◦ It actually doesn’t matter which

1 transitioned to the accept state *through* 2, so...
  ◦ We need to repair that transition
Review: Making the GNFA

Now we rip out a state
  ◦ It actually doesn’t matter which

1 transitioned to the accept state through 2, so…
  ◦ We need to repair that transition
  ◦ The concatenation is obvious
  ◦ Can you see why we need the star closure?
Review: Making the GNFA

Now we rip out a state

- It actually doesn’t matter which

1 transitioned to the accept state through 2, so...

- We need to repair that transition
- The concatenation is obvious
- Can you see why we need the star closure?

One more state, and we’re done
Review: Making the GNFA

Now we rip out a state
  ◦ It actually doesn’t matter which

1 transitioned to the accept state through 2, so...
  ◦ We need to repair that transition
  ◦ The concatenation is obvious
  ◦ Can you see why we need the star closure?

One more state, and we’re done
Reliable Ripping and Repair

Ripping is the easy part: Just pick a state \( q_r \) that isn’t the start or accept state.

Repair is the hard part. Consider every pair of states \( q_a \) and \( q_b \) so that:

- \( q_a \) can transition to \( q_r \) on regular expression \( R_{ar} \)
- \( q_r \) can transition to \( q_b \) on regular expression \( R_{rb} \)
- (If the transition can go the other way too, it counts as two pairs)

Since we aren’t picking the start or accept state, it is both necessary and sufficient to repair every such transition.

Three cases to consider:

- \( q_a \) can always transition to \( q_b \) on regular expression \( (R_{ar})(R_{rb}) \)
- If \( q_r \) has a self-loop on \( R_r \) then we concatenate with \( (R_r)^* \) to get \( (R_{ar})(R_r)^*(R_{rb}) \)
- And finally, if \( q_a \) can transition to \( q_b \) on regex \( R_{ab} \) without \( q_r \) involved, we union with \( (R_{ab}) \) to get:

\[
(R_{ar})(R_r)^*(R_{rb}) \cup (R_{ab})
\]
Definition: Generalized Nondeterministic Finite Automaton

A GNFA is a 5-tuple $G = \{Q, \Sigma, \delta, q_s, q_f\}$ where:
- $Q$ is the set of states,
- $\Sigma$ is the input alphabet,
- $\delta: (Q - \{q_a\}) \times (Q - \{q_s\}) \to R$ (with $R$ as the set of all regular expressions over $\Sigma$) is the transition function,
- $q_s$ is the start state, and
- $q_f$ is the (single) accept state

A GNFA accepts string $w$ on $\Sigma$ if:
- $w = w_1w_2 \ldots w_k$, and...
- ...state sequence $q_0q_1 \ldots q_k$ exists, so that $q_0 = q_s$ and $q_k = q_f$, and...
- $w_i \in L(\delta(q_{i-1}, q_i))$ for $i$ from 1 to $k$
Recursive Conversion

Let RIP(G) be a function that accepts a GNFA $G = \{Q, \Sigma, \delta, q_s, q_f\}$. It returns $G_R = \{Q_R, \Sigma, \delta_R, q_s, q_f\}$ so that:

- $Q_R = Q - \{q_r\}$ for some $q_r \notin \{q_s, q_f\}$, and
- For every $q_a \in Q_R - \{q_f\}$ and $q_b \in Q_R - \{q_s\}$,

$$
\delta_R(q_a, q_b) = (R_{ar})(R_r)^*(R_{rb})\cup(R_{ab})
$$

where: $R_{ar} = \delta(q_a, q_r)$, $R_{rb} = \delta(q_r, q_b)$, $R_r = \delta(q_r, q_r)$, and $R_{ab} = \delta(q_a, q_b)$

Now Let CONVERT(G) be a function that accepts a GNFA $G = \{Q, \Sigma, \delta, q_s, q_f\}$. It returns:

- The regular expression $\delta(q_s, q_f)$ if $|Q| = 2$, and
- CONVERT(RIP(G)) otherwise.
A Little Convincing

We can show RIP($G$) is equivalent to $G$:

- If $G$ accepts $w$, then $G$ enters states $q_s, q_1, q_2, \ldots, q_f$
  - If none of these are $q_r$, obviously RIP($G$) accepts $w$
  - If $q_r$ does appear, then let the states before and after it be $q_a$ and $q_b$, and our construction shows that $\delta_r$ provides a regular expression transition between them equivalent to all transitions through $q_r$

- If RIP($G$) accepts $w$, then RIP($G$) enters states $q_s, q_1, q_2, \ldots, q_f$
  - If none of the transitions previously involved $q_r$, obviously $G$ accepts $w$
  - If a transition did previously involve $q_r$, we just reverse our construction to observe that $G$ can make the same transition through $q_r$

- $G$ and RIP($G$) each accept everything the other does; therefore, they are equivalent.
Lemma: DFAs to Regular Expressions

Suffices to show that for a GNFA $G = \{Q, \Sigma, \delta, q_s, q_f\}$, CONVERT($G$) returns a regular expression describing $L(G)$.

- **Proof:** Induction on $|Q|$.
  - **Basis:** $|Q| = 2$. Then $G$ has a singular transition from $q_s, q_f$ for strings described by a regular expression $\delta(q_s, q_f) = R$, which CONVERT($G$) returns as desired.
  - **Induction Hypothesis:** Assume that for any $G_k = \{Q_k, \Sigma, \delta_k, q_{sk}, q_{fk}\}$ with $|Q_k| < |Q|$, CONVERT returns a regular expression describing $L(G_k)$.
  - **Induction:** Consider RIP($G$) = \{$Q_R, \Sigma, \delta_R, q_s, q_f$\}.
    - By definition of RIP($G$), $|Q_R| = |Q| - 1$.
    - Therefore, by the induction hypothesis, CONVERT(RIP($G$)) returns a regular expression describing $L(RIP(G))$.
    - We have already shown that $L(RIP(G)) = L(G)$.
    - CONVERT(RIP($G$)) returns a regular expression describing $L(G)$, as desired.
Closure of Regular Languages
Proof: Closure of Regular Languages – Union

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages $A$ and $B$. 
Proof: Closure of Regular Languages – Union

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages $A$ and $B$.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup Q_B \cup \{ s \}$
- Start state $s$
- $F = F_A \cup F_B$
Let \( N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \} \) and \( N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \} \) be NFAs recognizing regular languages \( A \) and \( B \).

Construct a new NFA \( N = \{ Q, \Sigma, \delta, s, F \} \) with:

- \( Q = Q_A \cup Q_B \cup \{ s \} \)
- Start state \( s \)
- \( F = F_A \cup F_B \)
- \( \delta(q, a) = \begin{cases} 
\delta_A(q, a) & q \in Q_A \\
\delta_B(q, a) & q \in Q_B \\
\{q_{0A}, q_{0B}\} & q = s \text{ and } a = \varepsilon \\
\emptyset & \text{otherwise}
\end{cases} \)
Proof: Closure of Regular Languages – Union

- $N$ clearly accepts everything $N_A$ or $N_B$ accept, and nothing else.
- Therefore by recognition, $N$ accepts everything in $A$ or in $B$, and nothing else.
- Therefore by union, $N$ accepts everything in $A \cup B$, and nothing else.
- Therefore by recognition, $N$ recognizes $A \cup B$.
- Therefore, there is an NFA recognizing $A \cup B$.
- Therefore, $A \cup B$ is regular. □
Proof: Closure of Regular Languages – Concatenation

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages $A$ and $B$. 
Proof: Closure of Regular Languages – Concatenation

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages $A$ and $B$.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup Q_B \cup \{ s \}$
- Start state $q_{0A}$
- $F = F_B$
Proof: Closure of Regular Languages – Concatenation

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ and $N_B = \{ Q_B, \Sigma, \delta_B, q_{0B}, F_B \}$ be NFAs recognizing regular languages $A$ and $B$.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup Q_B \cup \{ s \}$
- Start state $q_{0A}$
- $F = F_B$

$$\delta(q, a) = \begin{cases} 
\delta_B(q, a) & q \in Q_B \\
\delta_A(q, a) & q \in Q_A \text{ and } q \notin F_A \\
\delta_A(q, a) & q \in F_A \text{ and } a \neq \epsilon \\
\delta_A(q, \epsilon) \cup \{q_{0B}\} & q \in F_A \text{ and } a = \epsilon 
\end{cases}$$
Proof: Closure of Regular Languages – Concatenation

- $N$ clearly accepts every string consisting of a string accepted by $N_A$ followed by a string accepted by $N_B$, and only those strings.
- Therefore by recognition, $N$ accepts every string that is a string in $A$ followed by a string in $B$, and nothing else.
- Therefore by concatenation, $N$ accepts every string in $AB$, and nothing else.
- Therefore by recognition, $N$ recognizes $AB$.
- Therefore, there is an NFA recognizing $AB$.
- Therefore, $AB$ is regular. □
Proof: Closure of Regular Languages – Star

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ be an NFA recognizing regular language $A$. 
Proof: Closure of Regular Languages – Star

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ be an NFA recognizing regular language $A$.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup \{ s \}$
- Start state $s$
- $F = F_A \cup \{ s \}$
Proof: Closure of Regular Languages – Star

Let $N_A = \{ Q_A, \Sigma, \delta_A, q_{0A}, F_A \}$ be an NFA recognizing regular language $A$.

Construct a new NFA $N = \{ Q, \Sigma, \delta, s, F \}$ with:

- $Q = Q_A \cup \{ s \}$
- Start state $s$
- $F = F_A \cup \{ s \}$

$$
\delta(q, a) = \begin{cases} 
\delta_A(q, a) & q \in Q_A \text{ and } q \notin F_A \\
\delta_A(q, a) & q \in F_A \text{ and } a \neq \epsilon \\
\delta_A(q, \epsilon) \cup \{ q_{0A} \} & q \in F_A \text{ and } a = \epsilon \\
\{ q_{0A} \} & q = s \text{ and } a = \epsilon \\
\emptyset & \text{otherwise}
\end{cases}
$$
Proof: Closure of Regular Languages – Star

- \( N \) clearly accepts every string consisting of zero or more strings accepted by \( N_A \).
- Therefore by recognition, \( N \) accepts every string consisting of zero or more strings in \( A \).
- Therefore by star, \( N \) accepts every string in \( A^* \).
- Therefore by recognition, \( N \) recognizes \( A^* \).
- Therefore, there is an NFA recognizing \( A^* \).
- Therefore, \( A^* \) is regular. \( \square \)
Regular Expressions: Formal Cleanup
Regular Expression Equivalence

We split the set equivalence proof as normal. We need to prove two things:

If a language is regular, it is described by a regular expression

- Handled by the ability to create a GNFA for any DFA, and subsequently describe the language recognized by that GNFA with a regular expression

If a language is described by a regular expression, then that language is regular

- Recall the definition of regular languages

\[ R \] is a regular expression over the alphabet \( \Sigma \) if it is:

1. \( a \) for some \( a \in \Sigma \)
2. \( \varepsilon \)
3. \( \emptyset \)
4. \( (R_1 \cup R_2) \) where \( R_1 \) and \( R_2 \) are both regular expressions
5. \( (R_1 \circ R_2) \) where \( R_1 \) and \( R_2 \) are both regular expressions
6. \( (R_1^*) \) where \( R_1 \) is a regular expression

- 1-3 represent the languages \( \{a\} \), \( \{\varepsilon\} \) and the empty language, respectively
- 4-6 represent the union, concatenation and star closure of the language(s) described by the regular expression operand(s)
Expression-to-Language Equivalence

Consider each case of the definition of regular expressions

1. \( R = a \) for some \( a \in \Sigma \)
2. \( R = \varepsilon \)
3. \( R = \emptyset \)

...and for 4-6, we just use the same constructions from the regular class closure proofs
Next Time: NFA to DFA, Revisited