Def: An NFA \( N = (Q, \Sigma, \delta, s, F) \) can process a string \( x \in \Sigma^* \) from state \( p \) to state \( q \) if there is a trace computation
\[
(q_0, w_1, q_1, w_2, q_2, \ldots, q_{m-1}, w_m, q_m)
\]
such that \( q_0 = p, \) \( w_1 \ldots w_m = x, \) \( q_m = q, \) and for each \( i, \) \( 1 \leq i \leq m, \) \( (q_{i-1}, w_i, q_i) \in \delta. \)

- Works as is for a DFA \( M \) in place of an NFA \( N, \) and also for "strict NFAs" (no \( \varepsilon \)-arcs).
- If no \( \varepsilon \)-arcs, then each \( w_i \) is a char and so \( m = n = \text{det } 1 \times 1. \)

Proof → Tve.

Theorem: For every NFA \( N \) there is a DFA \( M \) such that \( L(M) = L(N). \) Idea previewed last lecture with the subset trick.
Formal Defn of Regular Expressions &
And Their NFA's. Definition By Induction.

Basis: \( \emptyset \) is a regexp, \( L(\emptyset) = \emptyset \) \( N_\emptyset : \emptyset \)

\( \varepsilon \) is a regexp, \( L(\varepsilon) = \{ \varepsilon \} \) \( N_\varepsilon \varepsilon \) \( \emptyset \)

\( N_\emptyset \) cannot process any string from \( s \) to \( f \).
\( N_\varepsilon \) can process \( \varepsilon \) from \( s \) to \( f : (s, \varepsilon, f) \)

Neither one can process any char(s) from \( s \) to \( f \). For any \( c \in \Sigma : \varepsilon \) is a regexp, \( L(\varepsilon) = \{ c, f \} \) \( N_\varepsilon \varepsilon \) \( s \)

Defn: \( L(N) = \{ x \in \Sigma^* : \text{for some } f \in F, \exists N \text{ can process } x \text{ from } s \text{ to } f \} \)

\( L(N_\emptyset) = \emptyset \), \( L(N_\varepsilon) = \{ \varepsilon \} \), and \( L(N_\varepsilon) = \{ c, f \} \)
Induction: Let any regexps $r_1$ and $r_2$ along with "their" NFAs $N_1$ and $N_2$ be given. Their means $L(N_1) = L(r_1)$, $L(N_2) = L(r_2)$ and each has a unique accepting state $f \neq s$. Then $r_3 = \text{def } (r_1 \cup r_2)$ is a regexp, with language $L(r_3) = L(r_1) \cup L(r_2)$ $N_3 = \ldots$ and its NFA is

Construction: $N_3 = (Q_3, \Sigma, S_3, S_3, F_3)$ where:

$Q_3 = Q_1 \cup Q_2 \cup \{s_3, f_3, f_3\}$ new states $S_3$ is the new start state $F_3 = \{f_3, f_3\}$ on $f_3$.

$S_3 = S_1 \cup S_2 \cup \{(s_3, \varepsilon, S_1), (s_3, \varepsilon, S_2), (f_1, \varepsilon, f_3), (f_2, \varepsilon, f_3)\}$.

Verification: We need $L(N_3) = L(r_3)$, using $L(r_3) = \text{def } L(r_1) \cup L(r_2)$ by induction hypothesis, $L(r_1) = L(N_1)$ and $L(r_2) = L(N_2)$.

So we need to show $L(N_3) \subseteq L(N_1) \cup L(N_2)$. We do by showing both $L(N_3) \subseteq L(N_1) \cup L(N_2)$ and $L(N_1) \cup L(N_2) \subseteq L(N_3)$.
Next case: Let $r_1, r_2, N_1, N_2$ be given... 

- $r_3 = (r_1 \cup r_2)$ is a regexp,
- $L(r_3) = L(r_1) \circ L(r_2)$, and the NFA $N_3$ is:

**Series Circuit.**

The first and third $\Sigma$ arcs are unnecessary; we could define $S_3 = S$, and $f_3 = f_2$ and inductively abide by the rules. The middle $\Sigma$ is needed: "Fusing" $S_2 = f_1$ can cause backfeed.

$x \in L(r_1) \circ L(r_2) \iff x$ can be broken as $x = \gamma \cdot z$

Ind. hip. $L(N_1) \circ L(N_2)$ such that $\gamma \in L(N_1)$ and $z \in L(N_2)$.

(note: either $\gamma$ or $z$ could be $\epsilon$.)

We want this to be equivalent to $x \in L(N_3)$.

$L(N_1) \circ L(N_2) \subseteq L(N_3)$ is clear.

$L(N_3) \subseteq L(N_1) \circ L(N_2)$: the only way $N_3$ can process $x$ is for $N_1$ to process an initial part $\gamma$ and $N_2$ does the rest.

$\therefore L(N_3) = L(N_1) \circ L(N_2)$

If we allowed $S_2 = f_1$ and $f_1$ had back arcs, then we could get "other stuff" in $L(N_3)$.
Last Case: We only need \( r_i \) and \( N_i \).

\[ r_3 = (r_i^*) \] is a regex.

\[ L(r_3) = \{ \epsilon \} \cup L(r_i)^* \]

\[ L(r_3) = \{ x \in \Sigma^* : x \text{ can be broken into zero or more parts } y_1, \ldots, y_m \text{ such that each } y_i (\text{if any}) \text{ belongs to } L(r_i) \} \]

Feedback Circuit...

\[ N_3 \]

\[ S_3 \]

\[ \epsilon \]

\[ f_3 \]

Cycle zero times

\[ L(N_3) = L(N_1)^* \]

Added: To verify \( L(N_3) = L(N_1)^* \), enough to see how \( t \) cycles \( N_i \) zero or more times. Analogy to a for-loop for \( \text{int } i = 0; i < n; i++ \) \( t \ldots 3 \) which "falls thru" if \( n = 0 \).