**Defn:** A Generalized Non-deterministic Finite Automaton (GNFA) is a 5-tuple \( N = (Q, \Sigma, S, s, F) \) where \( s, q \in Q \), \( s \in Q \) as with NFAs, but 

\[
S \subseteq (Q \times \text{Regexp}(\Sigma)) \times Q
\]

\[
\text{Regexp}(\Sigma) = \text{the set of regular expressions with } \Sigma \text{ as its character set}.
\]

**Defn:** A computation trace of a GNFA on an input \( x \in \Sigma^* \) of length \( n \) is a sequence \( \tau = (q_0, q_1, q_2, \ldots, q_{n-1}, q_n) \) s.t. \( q_0 = s \), \( q_n = q \), and \( x \) can be broken into substrings \( x =: U_1 \cdot U_2 \cdots U_m \) s.t. for each \( j \), \( 1 \leq j \leq m \):

\[
(q_{j-1}, q_j, q_j) \in S \text{ and } U_j \in L(q_j), \text{ i.e. matches } q_j.
\]

\[
L(p) = \{ x \in N \text{ has a valid trace with } q_0 = s, q_m = q \}.
\]

Theorem: For every GNFA \( N \) we can build a regexp \( R \) s.t. \( L(N) = L(R) \).

And vice-versa: Given any regexp \( R \), there is the trivial GNFA

\[
N_R = (Q, \Sigma, S, s, F).
\]

Text proof arranges that you get this kind of GNFA at the end, but it gets nasty so we will use a shortcut.
Proof: Use a general 2-state GNFA as a basis:

Then \( L_{11} = (\alpha \cup (\beta \gamma \eta) \cdot \eta) \)  

one time around the track  
(or in the gutter)

defines zero or more times around the track 

Above: \( \alpha = \epsilon \)  
\( L_{11} = (0 + \# (0+\#)^*) \)  

Example: \( \beta = \# \)  
\( r = \# + 0 \)  
no outgoing \( n = D \)

\[ L_{12} = L_{11} \cdot (\beta \cdot \gamma^*) \cdot (\beta \eta) \]  

home stretch and victory spins  
\[ L_{12} = L_{11} \cdot ((0+\#)^*) \]  

\[ L(N) = L_{11} \cup L_{12} = L_{11} + L_{11} \cdot (0+\#)^* \]  

\[ = (0 + \# (0+\#)^* D) \cdot (\epsilon + \# (0+\#)^*) \]  

answer in the Turing Machine

Alternative: \( L_{11} = \alpha^* \beta \cdot L_{22} = \alpha^* \beta \cdot (\gamma + \# \alpha^* \beta)^* \)  

"\( L_{22} \) once"

This works for any 2-state GNFA as the basis.

Induction: \( N \) has \( n \geq 3 \) states.

Let's assume \( \epsilon \) is the only accepting state different from 5 (if any).

Algorithm for \( k = n \) down to 3:

- eliminate state \( k \).

- done \( \Rightarrow \) read answer using basis for 2-state machine.

Strategy: Eliminate all non-accepting states different from 5 until we get \( n \geq 2 \).

Never need a new start state \( \Rightarrow \) Unless there are 2 accepting states different from 5 we're good.

If so, then add a new accepting state \( \epsilon \) with arcs from all the old ones. Numbers 2.

Because \( k \& F \), any processing that goes into \( k \) from some state \( i \) must go out via some state \( j \) in \( F \). This says for all \( i \) and \( j \), express \( \epsilon (i,k) \) to \( j \).
Diagram for Bypass

\[ \alpha_{\text{new}} = \alpha_{\text{old}} + \beta \gamma^N \]

Doing for all exits \( j \) from \( i \) bypasses edge \( \beta \).

\( \beta \)-passing all incoming edges allows you to eliminate state \( k \).

Added: Here is an example done "graphically"-Tuesdays lecture will do it "in code style". Consider the following DFA. It has only one accepting state beyond the start state, so we need not add any more state.

M =

\[ \begin{align*}
\alpha &= a + cc'a \\
\eta &= a + bc'a \\
\beta &= b + cc'b \gamma^N \\
\gamma &= a + cbb + cbb \gamma^N \\
\end{align*} \]

Eliminate (1): 
Incoming: \( (3, b, 1, 1, c) \) Out: \( (a, 1, 1, c, 2) \)
\[ \eta = a \quad \alpha = a + cc'a \quad \beta = b \quad \gamma = a + cbb + cbb \gamma^N \quad \eta = a + bc'a \]

New loop at (1) = old old \( \beta \gamma^N + CC' a = a + CC'a \)
New (3, 2) = old (3, 2) * b cc'b = c + bc'b
New (1, 2) = old 11, 27 + CC' b = b + CC' b. Then elim (4)

Eliminate (2): In terms (1) and (3) Out only to (3).
So we only need to update (3, 3) and (3, 3). There was no (1, 3!) but now there's (b + cc'b) (b + c)' a. And (3, 3) becomes (b + cc'b) a. Now do...