Equivalence of Regular Languages, Expressions, and Automata

CSCI 3136: Principles of Programming Languages

Agenda

- Announcements
  - Assignment 1 is out and due May 24
- Readings:
  - Today: 2.2.1
  - Next: 2.2.1
  - Note: I recommend using alternative texts for this part of the course:
  - E..g, Hopcorft et al, “Introduction to Automata Theory”
- Lecture Contents
  - Regular Languages Equivalence Theorem
  - Equivalence between RLs and Res
  - Equivalence between RE’s and NFAs
  - Equivalence between NFAs and DFAs
  - Minimization of DFAs (time permitting)

Are these all the same?

- We have discussed a variety of specifications: RLs, RE, DFAs, NFAs
  - RLs: a class of languages
  - RE a way to specify RLs
  - DFAs: a way to implement scanners for RLs
  - NFAs: a simpler way to implement scanners for RLs
- Questions:
  - Are these all of equal power?
  - Are NFAs same as DFAs?
  - Do REs specify only regular languages?

Regular Languages Equivalence Theorem

- Thm: The following statements are equivalent:
  i. L is a regular language.
  ii. L is the language described by a regular expression.
  iii. L is recognized by an NFA.
  iv. L is recognized by a DFA.
- We will prove: (i) ≡ (ii) ≡ (iii) ≡ (iv)

Regular Languages are equivalent to Regular Expressions

- Every regular language can be specified by a regular expression.
- Every regular expression specifies a regular language.
- Idea: There is a one-to-one correspondence between the 2 definitions.
- Apart from notation, the recursive definitions are identical.

Operation	Regular Language	Regular Expression
Empty Language	$\emptyset$	$\emptyset$
Empty String	$\{\varepsilon\}$	ε
Single character	$\{a\}, a \in \Sigma$	a
Disjunction	$L_1 \cup L_2$	$R_1 \vert R_2$
Concatenation	L_1L_2	R_1R_2
Kleene-*	L^*	R*

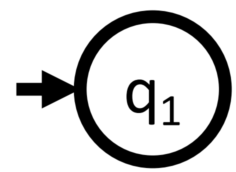
Regular Expressions are Equivalent to NFAs

- Proof: We will show that
  1. For each RE R there is an NFA M that recognizes L(R)
  2. For each NFA M there is an RE that specifies L(M)
- We do part 1 first.
  - Idea: For each RE base case and inductive step we can construct a corresponding NFA, hence for any RE, we can construct an NFA.
- Recall the base cases:
  - Empty Language: **∅**
  - Empty String: **ε**
  - Single character: **a**
- And inductive steps:
  - Disjunction: $\mathbf{R_1|R_2}$
  - Concatenation: $\mathbf{R_1R_2}$
  - Kleene-*: **R***

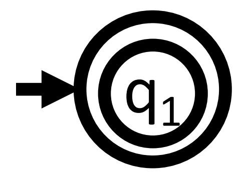
NFA for each RE Base Case.

Recall: An NFA $M = (Q,\Sigma,\delta,q_S,F)$

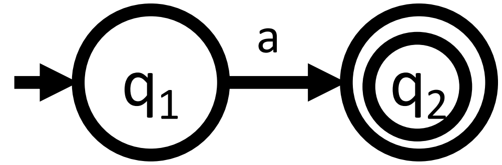
- Empty Language: **Ø** : $Q = \{q_1\}$, $F = \emptyset$, $\delta = \emptyset$



- Empty String: **ε** : $Q = \{q_1\}$, $F = \{q_1\}$, $\delta = \emptyset$

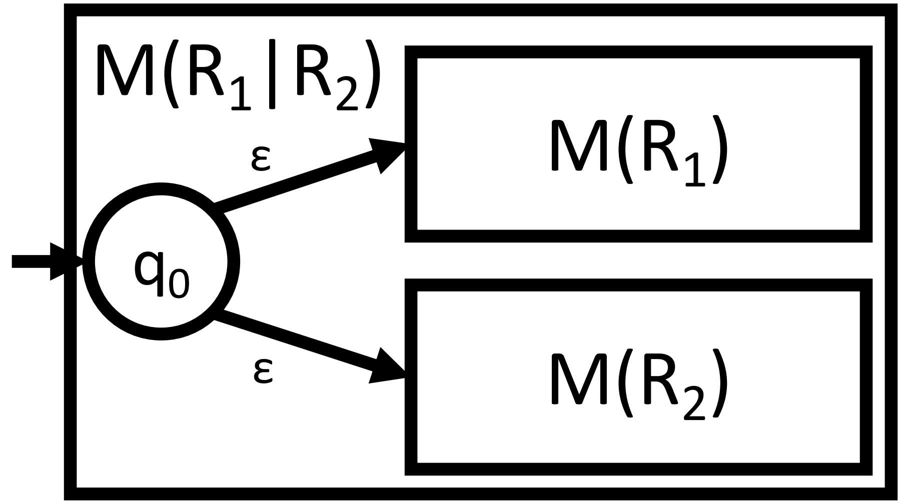


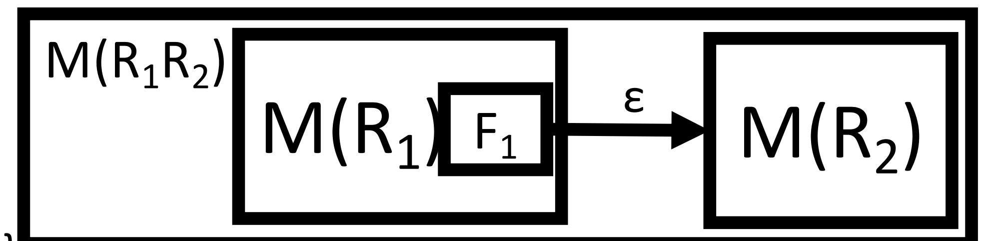
- Single character: **a** : $Q = \{q_1,q_2\}$, $F = \{q_2\}$, $\delta(q_1,a) = q_2$

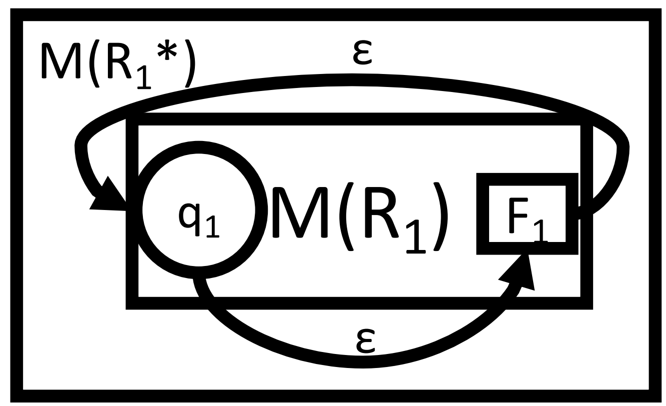


NFAs for each RE Inductive Step

- Notation:
  - $M(R_1) = (Q_1, \Sigma, \delta_1, q_1, F_1)$
  - $M(R_2) = (Q_2, \Sigma, \delta_2, q_2, F_2)$
- **Disjunction:** $R_1|R_2$:
  - $M(R_1|R_2) = (Q, \Sigma, \delta, q_0, F)$
  - $Q = Q_1 \cup Q_2 \cup \{q_0\}$,
  - $F = F_1 \cup F_2$,
  - $\delta = \delta_1 \cup \delta_2 \cup \{\delta(q_0,\varepsilon) = \{q_1,q_2\}\}$
- **Concatenation**: R_1R_2:
  - $M(R_1R_2) = (Q, \Sigma, \delta, q_1, F_2)$
  - $Q = Q_1 \cup Q_2$
  - $\delta = \delta_1 \cup \delta_2 \cup \{\delta(q,\varepsilon) = \{q_2\} \mid q \in F_1\}$
- **Kleene-*:** $R_1{*}$:
  - $M(R_1{*}) = (Q_1, \Sigma, \delta, q_1, F_1)$
  - $\delta = \delta_1 \cup \{\delta(q_1,\varepsilon) = \{q \in F\} \cup \{\delta(q,\varepsilon) = \{q_1\} \mid q \in F_1\}$

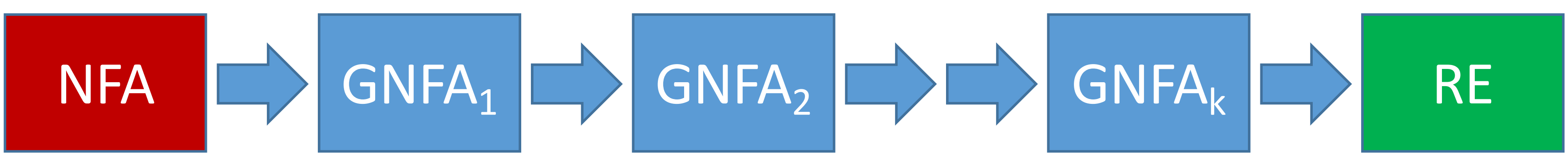






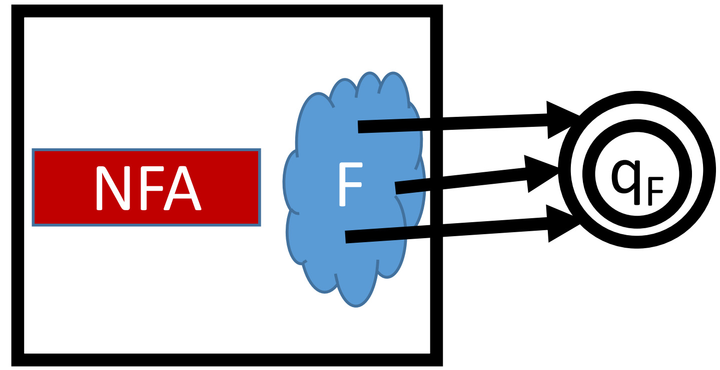
Back to Regular Expressions are Equivalent to NFAs

- Proof: We will show that
  1. For each RE R there is an NFA M that recognizes L(R)
  2. For each NFA M there is an RE the specifies L(M)
- Part 2 is a bit trickier.
- Proof Idea:
  - Treat NFA as a GNFA (Generalized NFA)
    - Edges are labeled by REs, not just characters
    - If $\delta(q_1,\alpha) = q_2$, the $(q_1,\alpha\beta) \rightarrow (q_2,\beta)$
- Start with the NFA (which is a GNFA)
- Collapse the GNFA, one state at a time into an RE

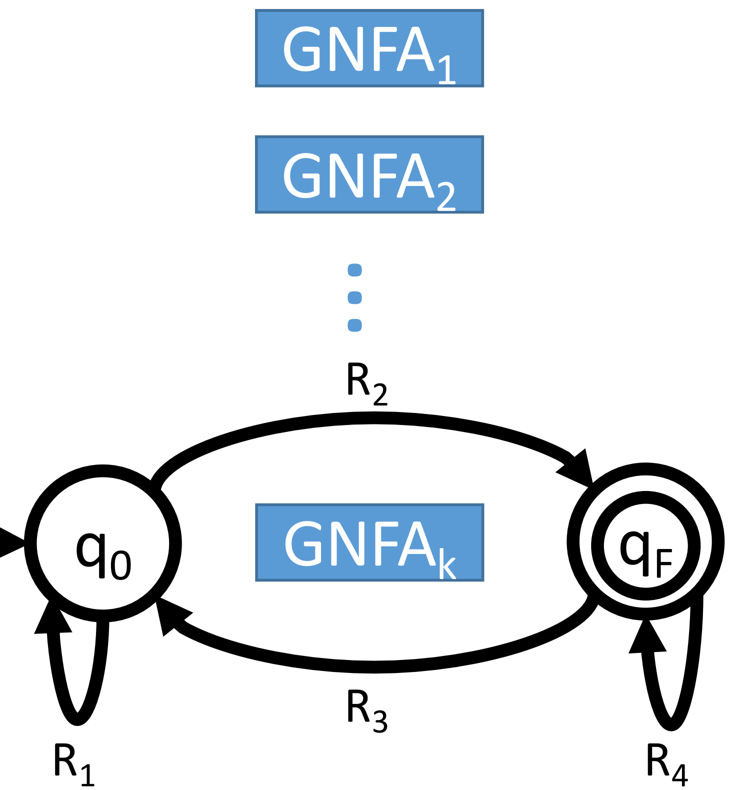


NFA to RE To Do List

- Normalize NFA by ensuring only one final state.
  - Add ε transitions and a new final state if needed
- Collapse GNFA to a two state start/finish GNFA
  - one state at a time
- Transform the two state GNFA to an RE

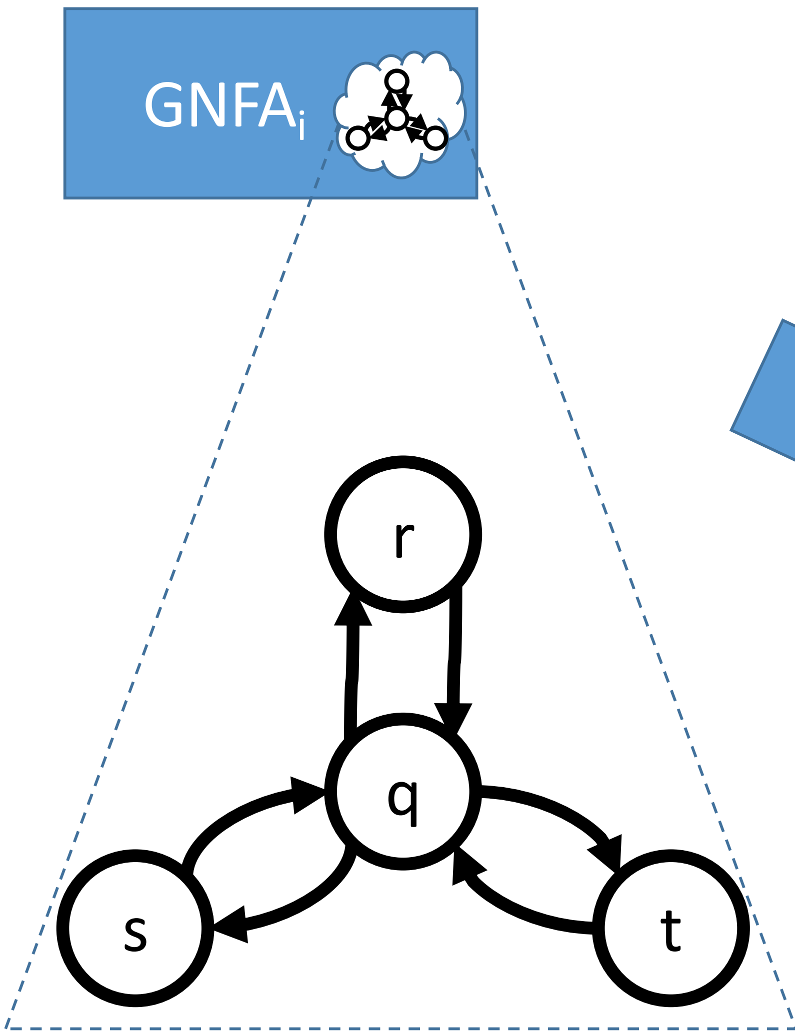


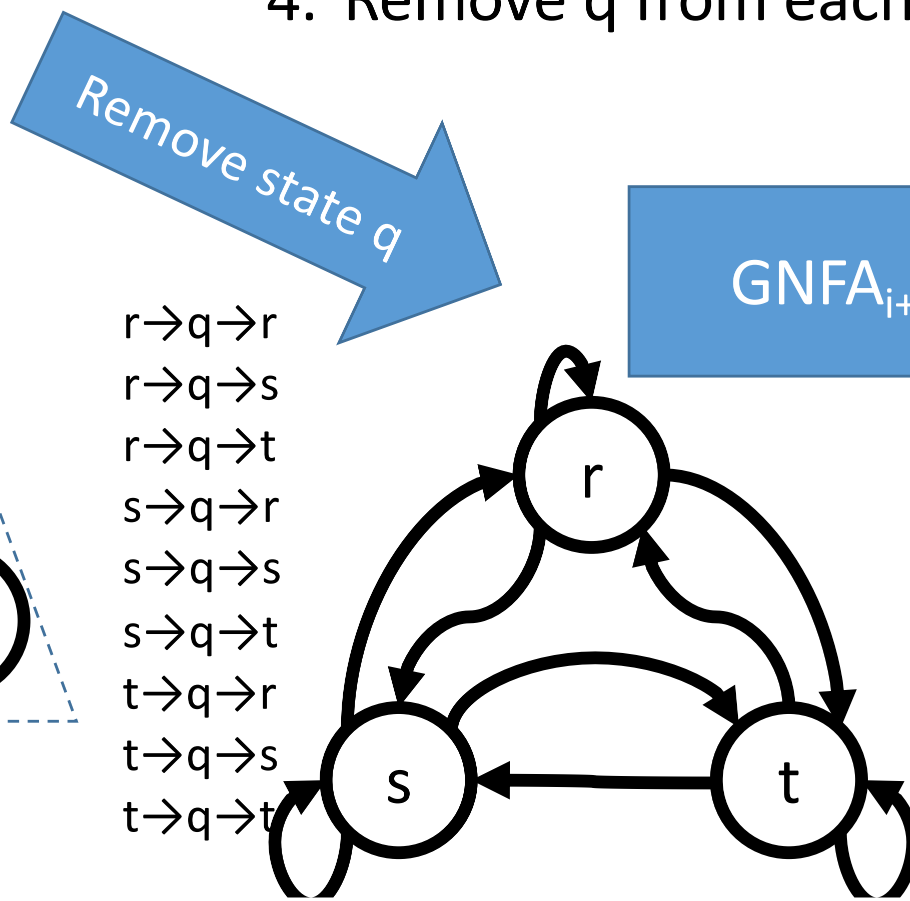
RE = $R_1{*}\, R_2\, ((R_3R_1{*}R_2) \mid R_4){*}$



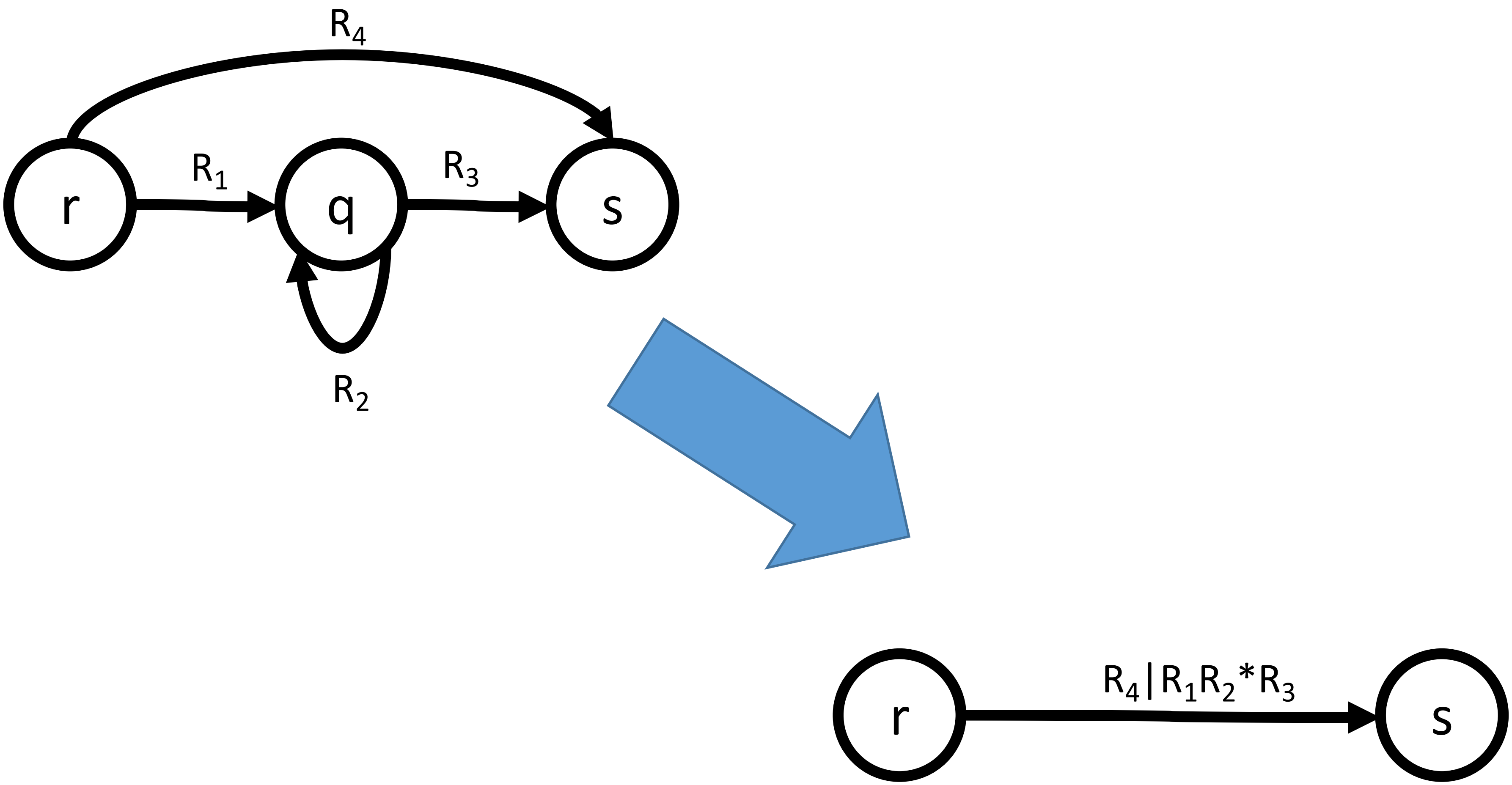
Collapsing the GNFA

1. Select state q to remove
2. Identify adjacent states
3. Identify all paths through q
4. Remove q from each path

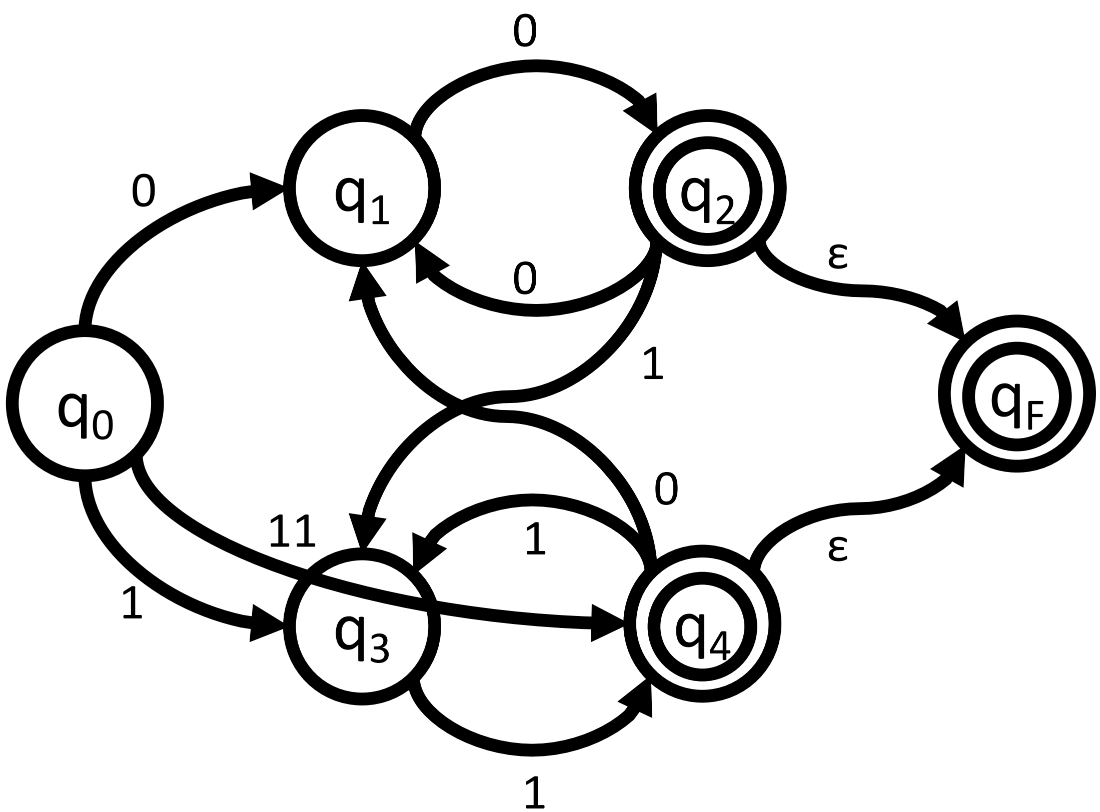




Removing a State from a Path



Example for NFA to RE Process



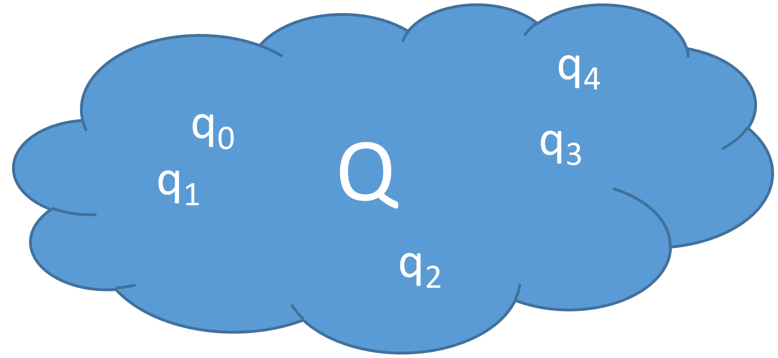
Regular Languages Equivalence Theorem

- Thm: The following statements are equivalent:
  i. L is a regular language.
  ii. L is the language described by a regular expression.
  iii. L is recognized by an NFA.
  iv. L is recognized by a DFA.
- We will prove: (i) ≡ (ii) ≡ (iii) ≡ (iv)

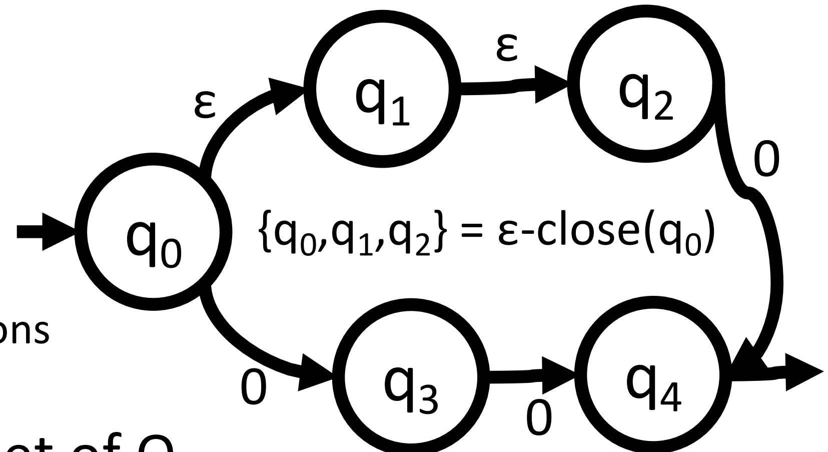
NFAs are Equivalent to DFAs

- Proof: We will show that
    1. For each DFA M that accepts L there is an NFA N that recognizes L
    2. For each NFA N that accepts L there is an DFA M that recognizes L
- We do part 1 first.
    - This is easy. Every DFA is by definition also an NFA.
- The second part is a bit trickier. ☺

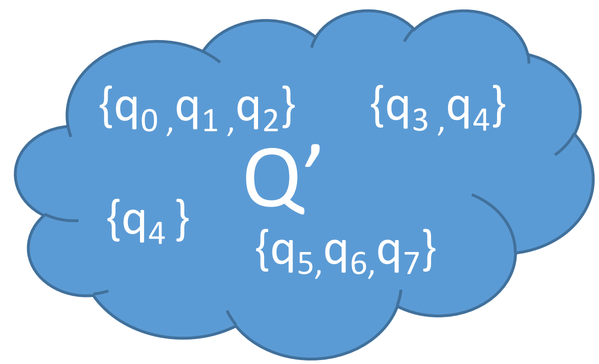
For each NFA N(L) there is a DFA M(L)



- Define
  - NFA $N = (Q,\Sigma,\delta,q_0,F)$
  - DFA $M = (Q',\Sigma,\delta',q_0',F')$
- Where
  - $q_0' = \varepsilon\text{-close}(q_0)$
    - $\varepsilon\text{-close}(q) = \{p \in Q \mid \delta(q, \varepsilon) = p\}$
    - Set of states form q_0 reachable by ε transitions
    - Note: $\varepsilon\text{-close}(P) = \cup_{p\in P}\, \varepsilon\text{-close}(p)$

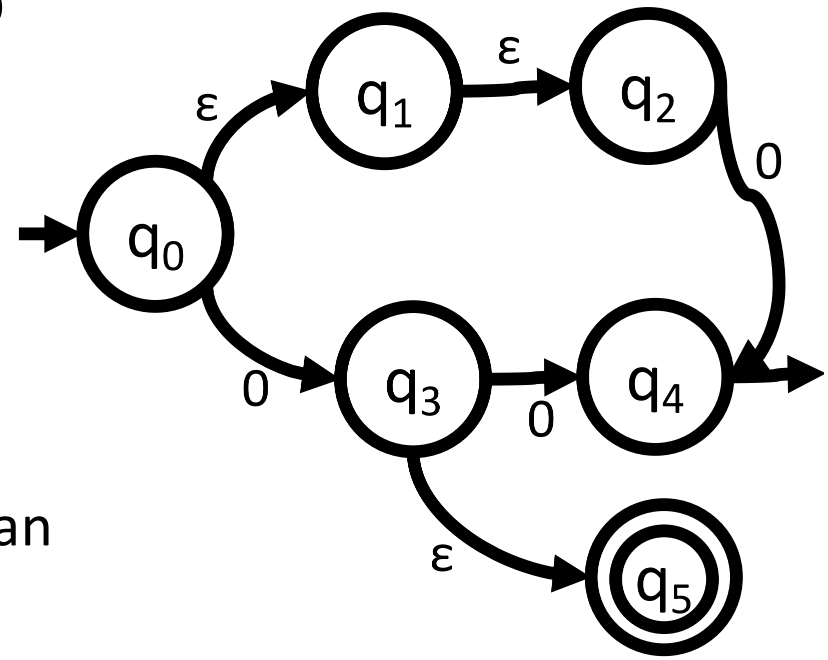


- Each state in Q' is represented by a subset of Q
  I.e., $Q' \subseteq 2^Q$
- We will build Q' iteratively:
  - Start with the start state $q_0' \in Q'$
  - For each $q' \in Q'$ and $a \in \Sigma$ compute $p' = \delta'(q',a)$
  - Add p' to Q' if $p' \notin Q'$
  - Repeat steps until no more states are added.



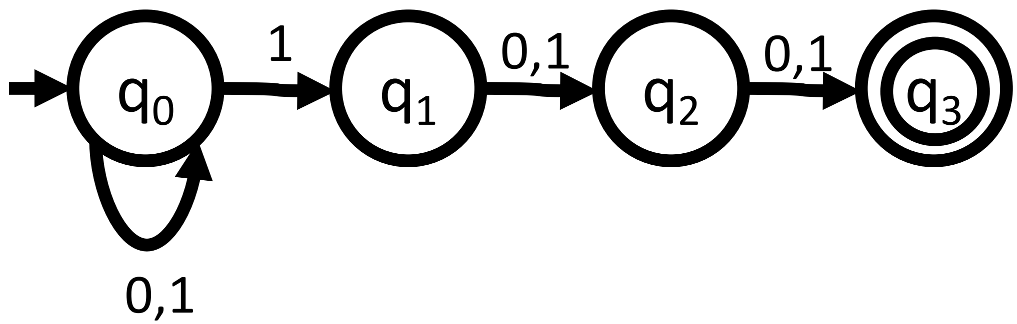
The δ’ Function for DFA M(L)

- The transition function $\delta'(q',a) = p'$ where
  - $p' = \varepsilon\text{-close}(P)$
  - $P = \{\delta(q,a) \mid q \in q'\}$
  - Example: $\delta'(\{q_0,q_1,q_2\},0) = \{q_3,q_4,q_5\}$
- Lastly, $F' = \{q' \in Q' \mid F \cap q' \neq \emptyset\}$
  - Every state in F' contains a state of an NFA that was in its final set.
  - Example: $\{q_2,q_3,q_5\} \in F'$
- In the worst case,the DFA is exponentially bigger than the NFA.



Example L=(0|1)*1(0|1)(0|1)

- $q_0' = \{q_0\}$
- Q' = (see table)
- δ' = (see table)
- F' = **(bolded states)**

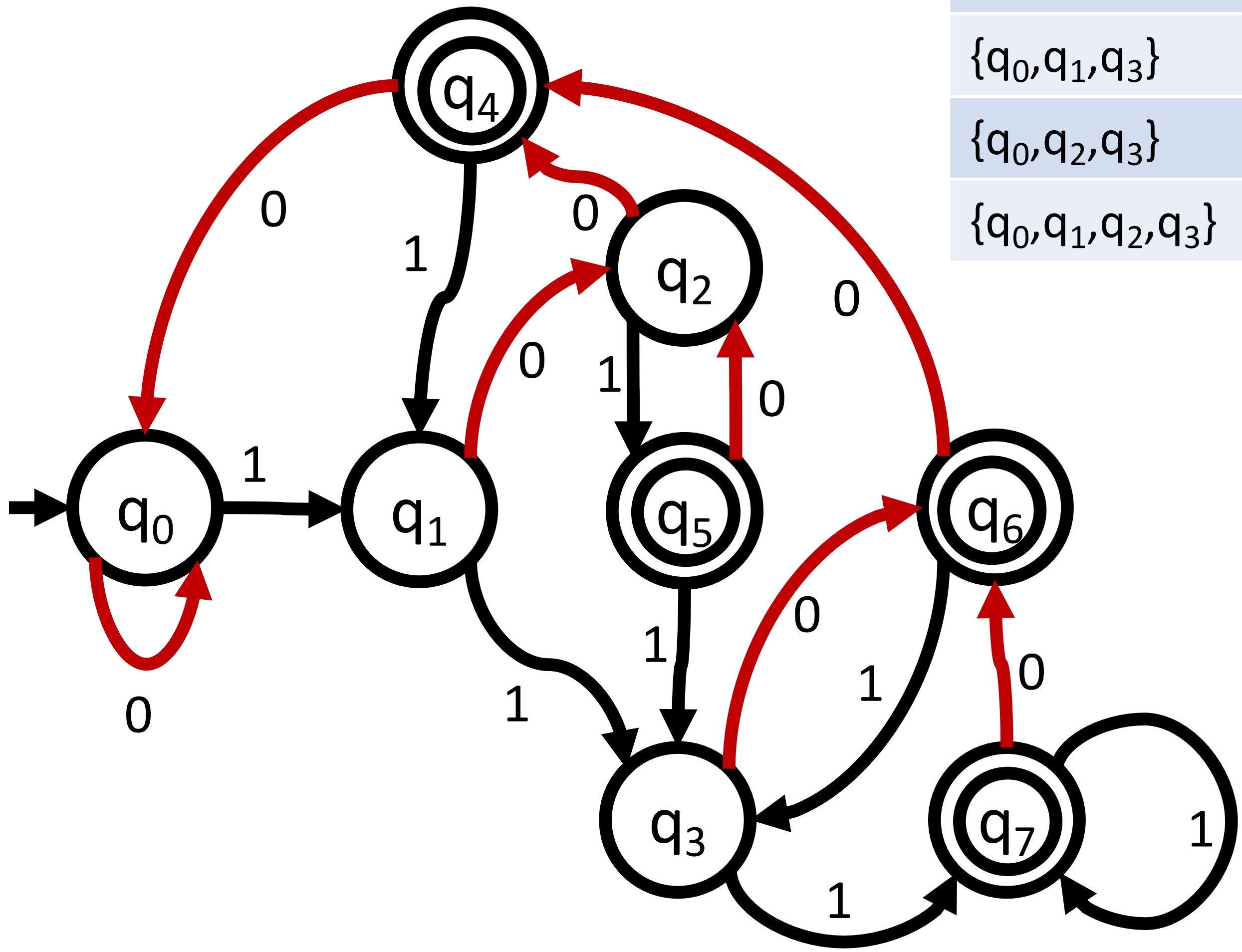


State	0	1
$\{q_0\}$	$\{q_0\}$	$\{q_0,q_1\}$
$\{q_0,q_1\}$	$\{q_0,q_2\}$	$\{q_0,q_1,q_2\}$
$\{q_0,q_2\}$	$\{q_0,q_3\}$	$\{q_0,q_1,q_3\}$
$\{q_0.q_1,q_2\}$	$\{q_0,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$
$\{q_0,q_3\}$	$\{q_0\}$	$\{q_0,q_1\}$
$\{q_0,q_1,q_3\}$	$\{q_0,q_2\}$	$\{q_0,q_1,q_2\}$
$\{q_0,q_2,q_3\}$	$\{q_0,q_3\}$	$\{q_0,q_1,q_3\}$
$\{q_0,q_1,q_2,q_3\}$	$\{q_0,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$

Example

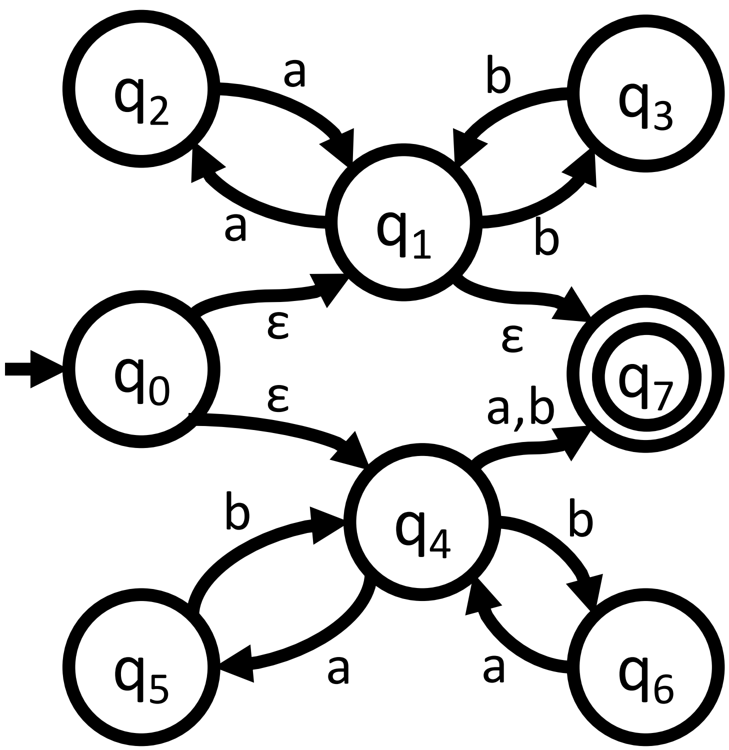
L=(0|1)*1(0|1)(0|1)

State	Q'	0	1
$\{q_0\}$	q_0'	$\{q_0\}$	$\{q_0,q_1\}$
$\{q_0,q_1\}$	q_1'	$\{q_0,q_2\}$	$\{q_0,q_1,q_2\}$
$\{q_0,q_2\}$	q_2'	$\{q_0,q_3\}$	$\{q_0,q_1,q_3\}$
$\{q_0.q_1,q_2\}$	q_3'	$\{q_0,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$
$\{q_0,q_3\}$	q_4'	$\{q_0\}$	$\{q_0,q_1\}$
$\{q_0,q_1,q_3\}$	q_5'	$\{q_0,q_2\}$	$\{q_0,q_1,q_2\}$
$\{q_0,q_2,q_3\}$	q_6'	$\{q_0,q_3\}$	$\{q_0,q_1,q_3\}$
$\{q_0,q_1,q_2,q_3\}$	q_7'	$\{q_0,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$



Example
L=(aa|bb)* | (ab|ba)*(a|b)

- $q_0' = \{q_0,q_1,q_4,q_7\}$
- Q' = (see table)
- δ' = (see table)
- F' = (see **bolded** entries)

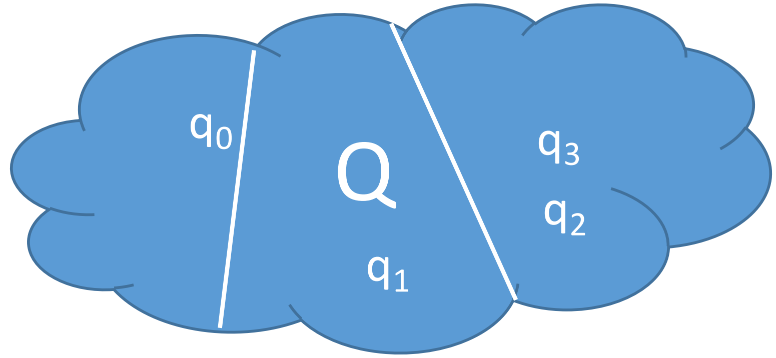


State	a	b
$\{q_0,q_1, q_4,q_7\}$	$\{q_2,q_5,q_7\}$	$\{q_3,q_6,q_7\}$
$\{q_2,q_5,q_7\}$	$\{q_1,q_7\}$	$\{q_4\}$
$\{q_3,q_6,q_7\}$	$\{q_4\}$	$\{q_1,q_7\}$
$\{q_1,q_7\}$	$\{q_2\}$	$\{q_3\}$
$\{q_4\}$	$\{q_5,q_7\}$	$\{q_6,q_7\}$
$\{q_2\}$	$\{q_1,q_7\}$	$\emptyset$
$\{q_3\}$	$\emptyset$	$\{q_1,q_7\}$
$\{q_5,q_7\}$	$\emptyset$	$\{q_4\}$
$\{q_6,q_7\}$	$\{q_4\}$	$\emptyset$

Minimization of Automata

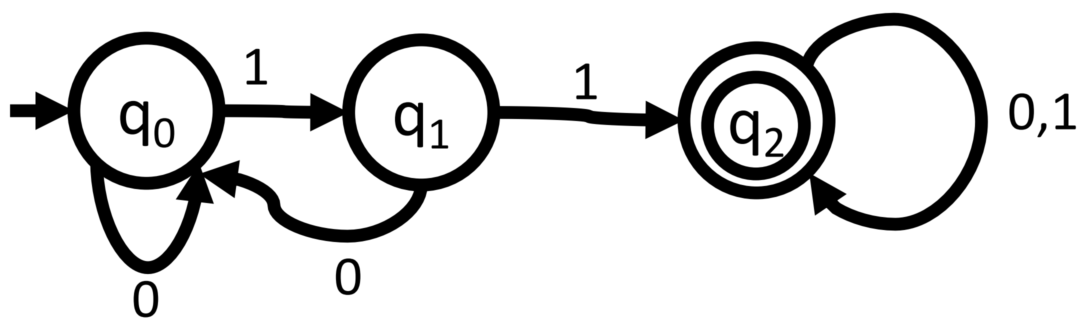
- **Motivation**: To build a scanner, we need to build a DFA
- The simpler a DFA is, the more efficient it is.
- So, we want to build the smallest DFA possible
- **Process**:
  - Build a DFA to recognize L
  - Minimize it.
- A DFA is *minimal* if it has the minimum number of states necessary to recognize L

Equivalence Classes



- Start with a DFA M = (Q,,Σ,δ,q_0,F)
- **Idea**: Divide Q into equivalence classes.
- The classes represent the states of the minimal DFA
- **Definition**: q_1 and q_2 are *equivalent* (in the same class) means for all $\sigma \in \Sigma^*$, $\delta(q_1,\sigma) \in F$ if and only if $\delta(q_2,\sigma) \in F$
- I.e., If there exists a string σ such that
  - $\delta(q_1,\sigma) \in F$
  - $\delta(q_2,\sigma) \notin F$

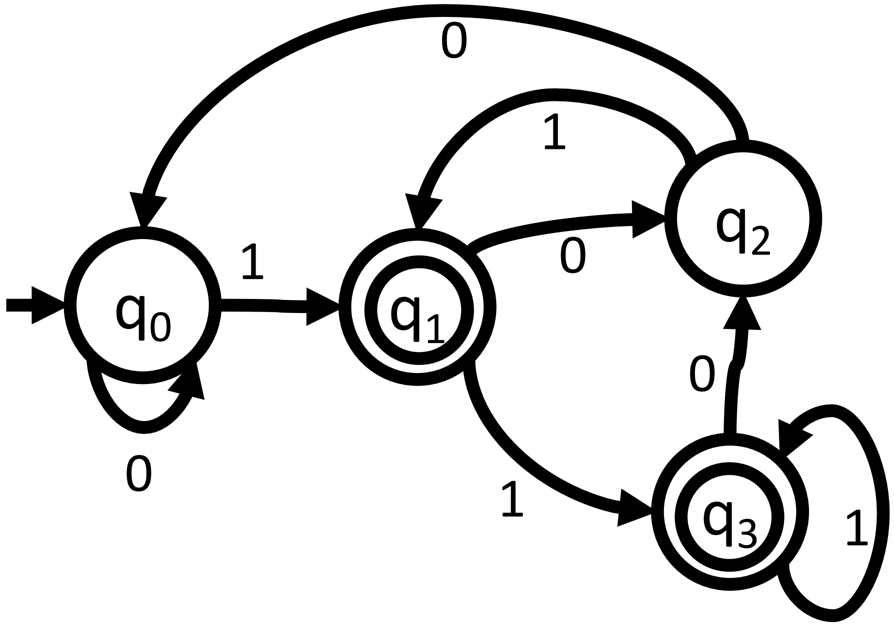
  then the two states are not in the same class.
- Example: q_0 and q_1 are in different classes

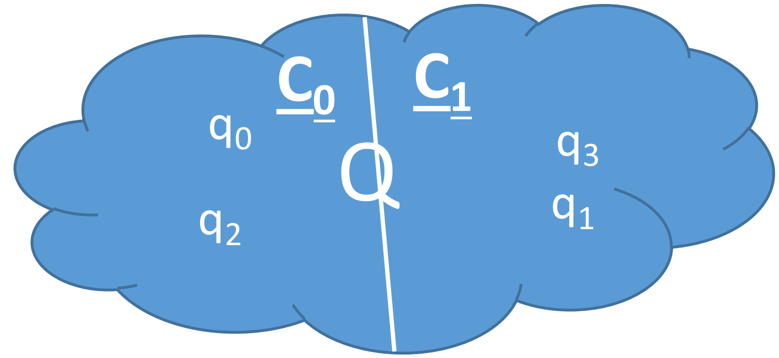


Minimization Procedure

- Initially all states are either accepting or not
- **If** there is a class C and character $a \in \Sigma$ such that
  $\{\delta(q_i,a) | q_i \in C\}$ are in $k > 1$ equivalence classes
- **Then** Split C into k classes C_j such that
  $\delta(q_i, a)$, where $q_i \in C_k$, are in the same equivalence class.
- Repeat until no more splits are needed.

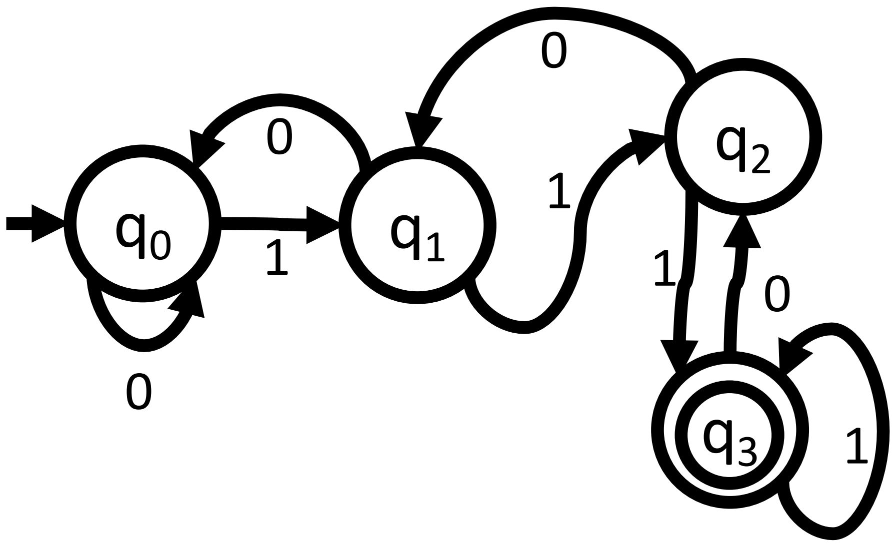
Example 1

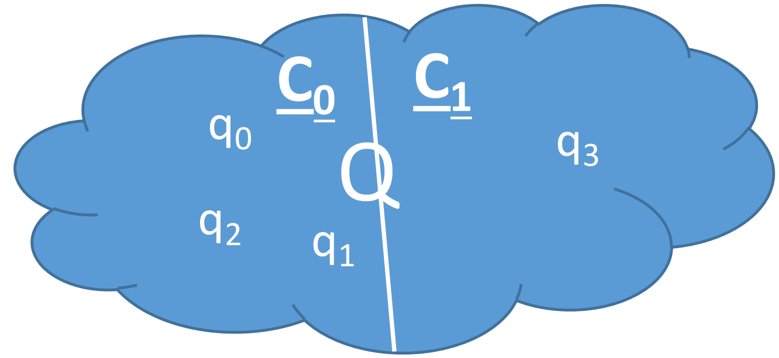




	Q	0	1
C_0	q_0	C_0	C_1
	q_2	C_0	C_1
C_1	q_1	C_1	C_0
	q_3	C_1	C_0

Example 2





	Q	0	1	0	1	0	1	
C_0	q_0	C_0	C_0	C_0	C_0			
	q_2	C_0	C_0	C_0	C_2			C_3
	q_1	C_0	C_1					C_2
C_1	q_3							