Equivalence of Regular Languages, Expressions, and Automata

CSCI 3136: Principles of Programming Languages

Agenda

- 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

- Theorem: 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

Operation	Regular Language	Regular Expression
Empty Language	A	A
Empty String	{ε}	ε
Single character	{a}, a E Σ	a
Disjunction	L1 ν L2	R1|R2
Concatenation	L1L2	R1R2
Kleene-*	LX	R*

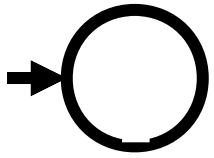
Regular Expressions are Equivalent to NFAs

- 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: **A**

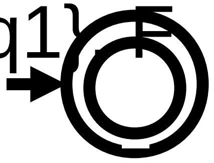
NFA for each RE Base Case.

Recall: An NFA *M = (Q,Σ,δ,qS,F)*

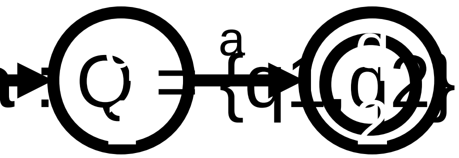
- Empty Language: **A** : Q = {q1}, F = A, δ = A



- Empty String: **ε** : Q = {q1}, F = {q1}, δ = A

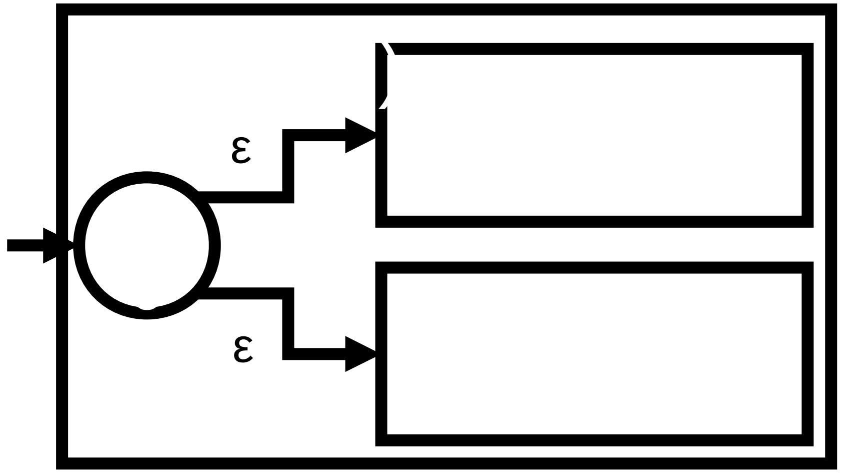


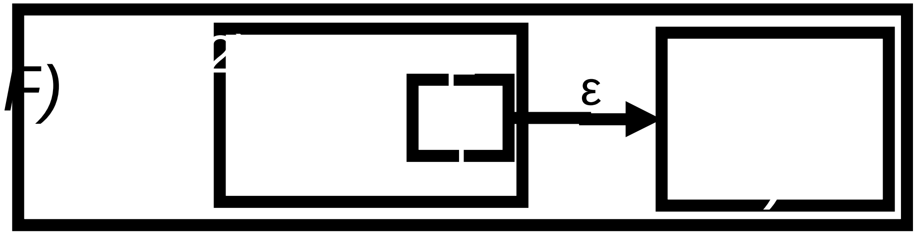
- Single character: **a** : Q = {q1,q2}, F = {q2}, δ(q1,a) = q2

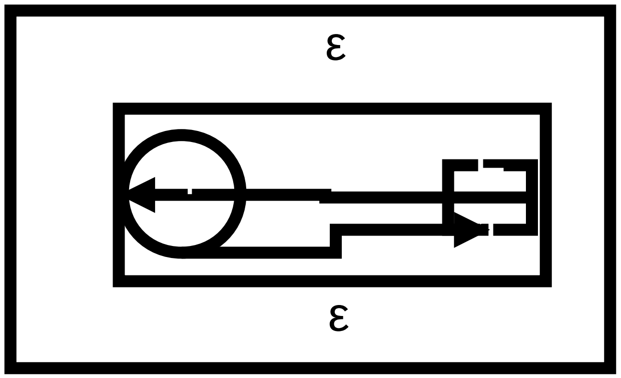


NFAs for each RE Inductive Step

- Notation:
  - *M(R1) = (Q1,Σ,δ1,q1,F1)*
  - *M(R2) = (Q2,Σ,δ2,q2,F2)*
- **Disjunction:** R1|R2 :
  - *M(*R1|R2) = (*Q,Σ,δ,q0,F)*
  - Q = Q1 νQ2 ν{q0},
  - F = F1 νF2,
  - δ = δ1 νδ2 ν{δ(q0,ε) = {q1,q2}}
- **Concatenation**: R1R2:
  - *M(*R1R2) = (*Q,Σ,δ,q1,F2)*





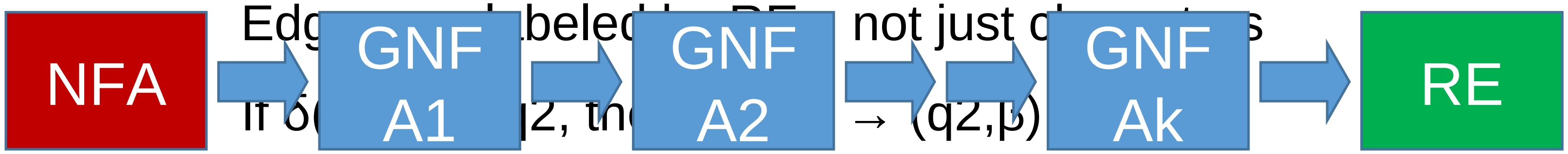


Back to Regular Expressions are Equivalent to NFAs

- 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)

Edg[illegible]beled [illegible] not just c[illegible]s

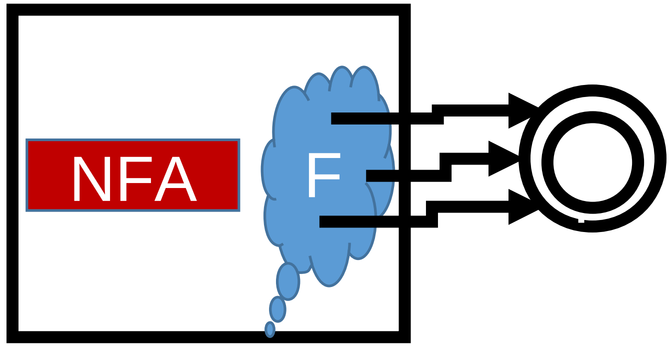
If δ([illegible]2, th[illegible] → (q2,β)

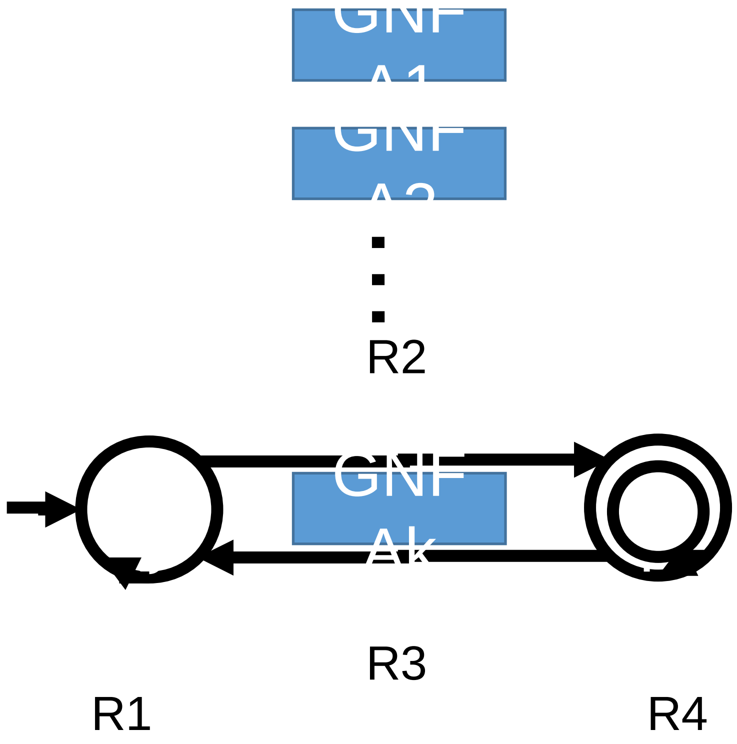


- Start with the NFA (which is a GNFA)

NFA to RE To Do List

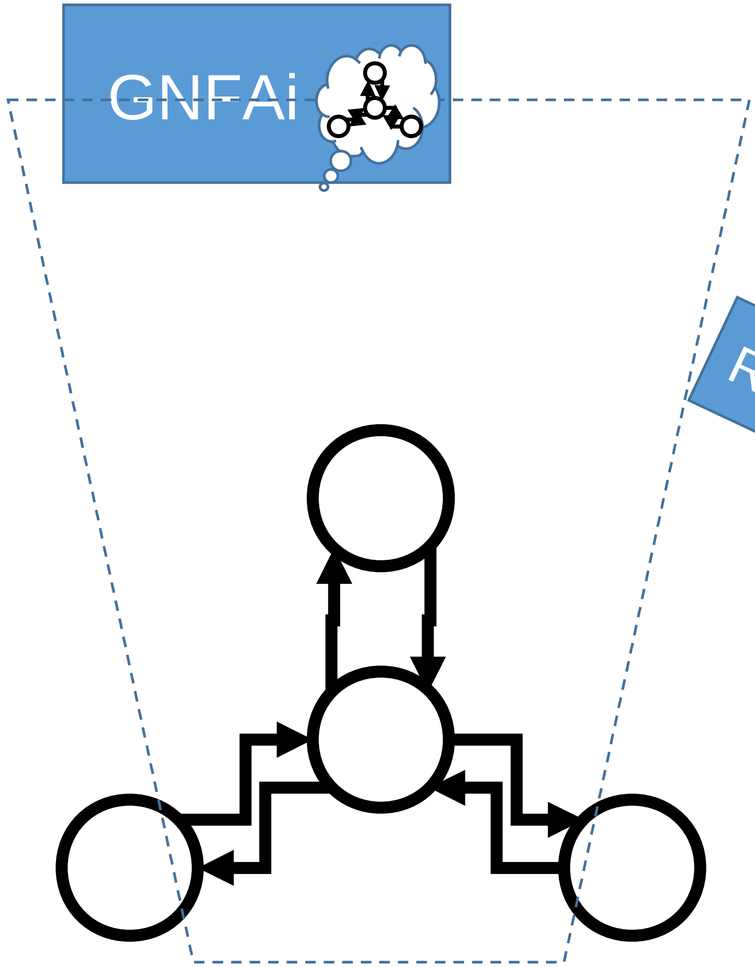
- Normalize NFA by ensuring only one final state.
  - Add ε transitions and a new final state if needed
- Collapse GNFA to a two state start/finish GNFA
  - one state at a time
- Transform two the state GNFA to an RE

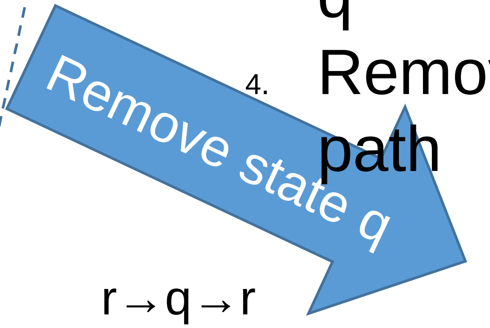




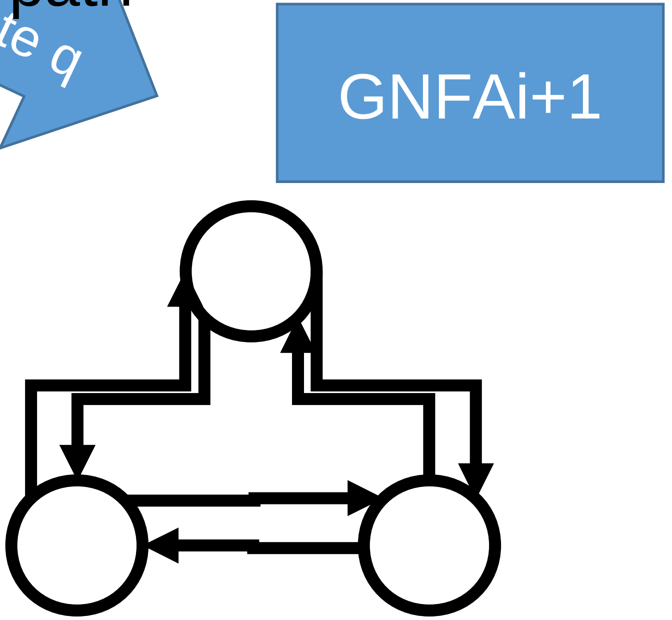
Collapsing the GNFA

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

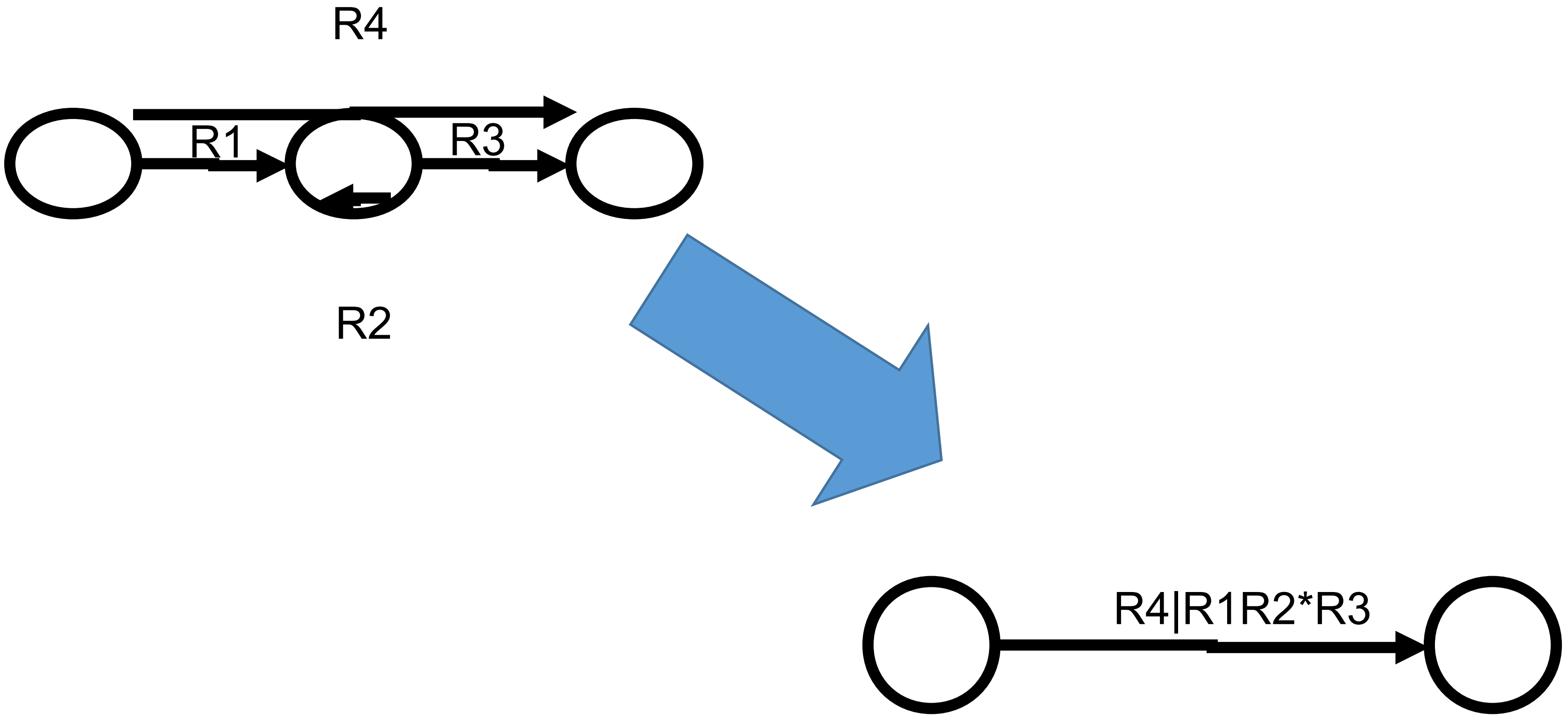




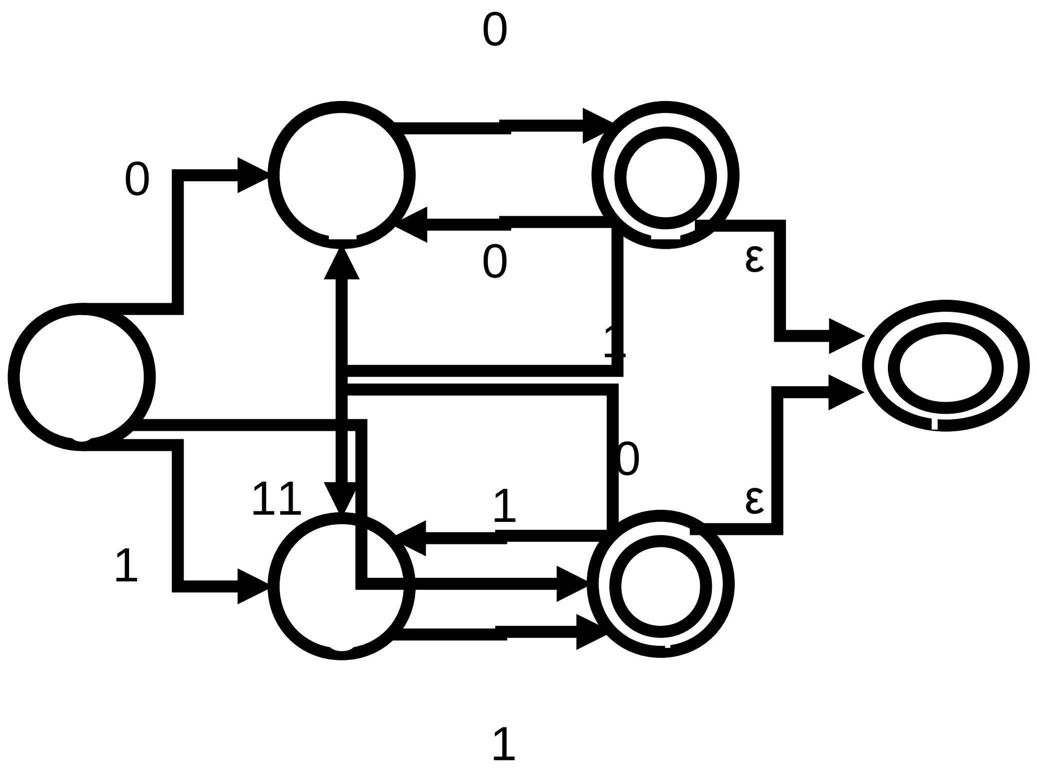
$r \to q \to r$
$r \to q \to s$
$r \to q \to t$
$s \to q \to r$
$s \to q \to s$
$s \to q \to t$
$t \to q \to r$
$t \to q \to s$
$t \to q \to t$



Removing a State from a Path



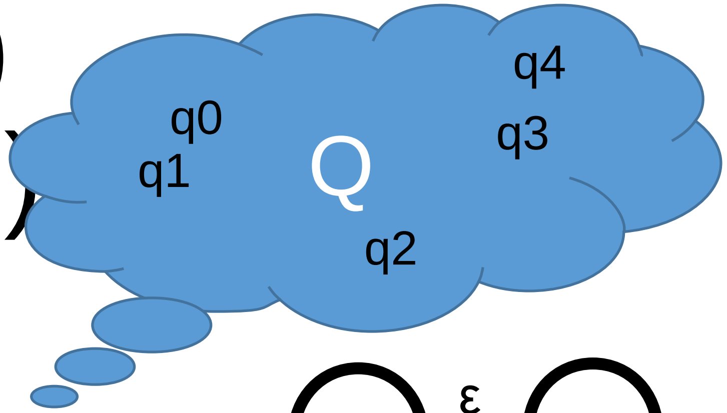
Example for NFA to RE Process



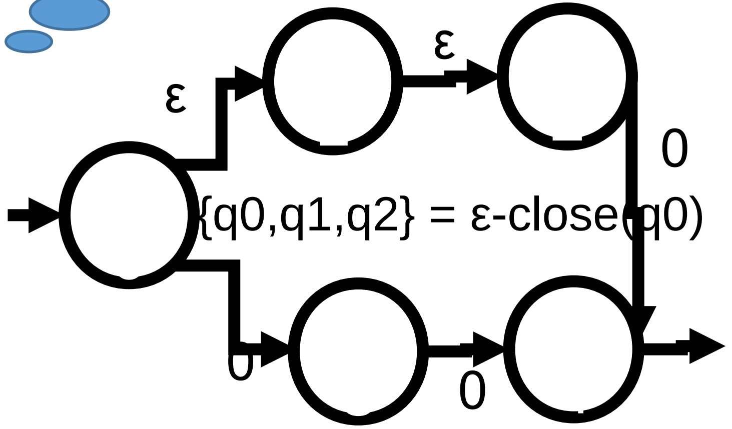
NFAs are Equivalent to DFAs

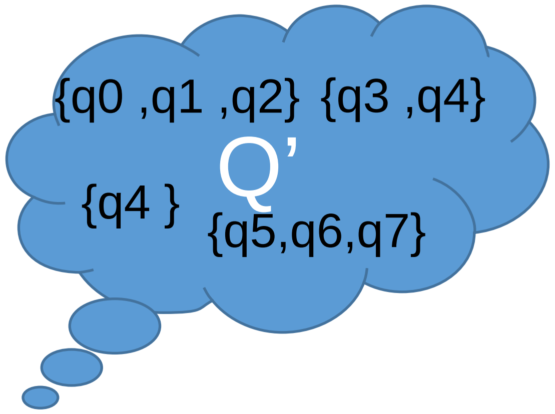
- Proof: We will show that
  1. For each DFA M that accepts L there is an NFA N that recognizes L
  2. For each NFA N that accepts L there is an DFA M that recognizes L
- We do part 1 first.
  - This is easy. Every DFA is by definition also an NFA.
- The second part is a bit trickier. ☺

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



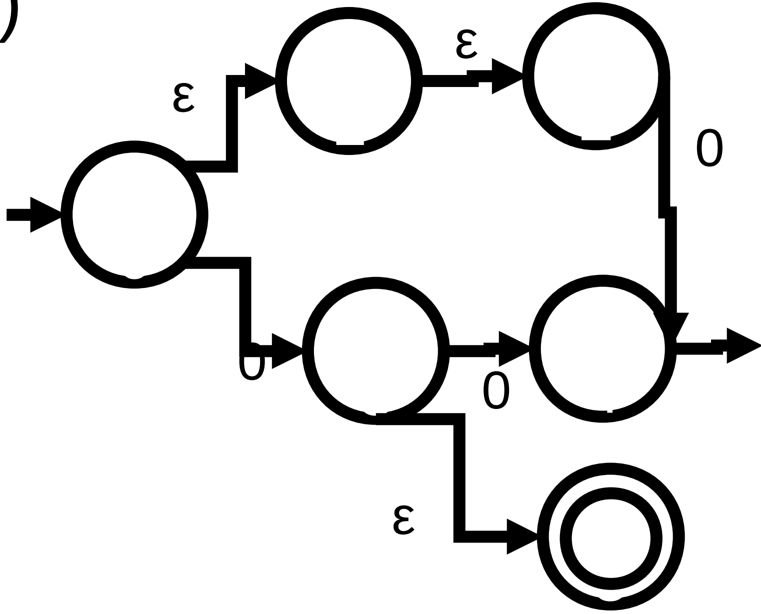
- Define
  - NFA N = (Q,Σ,δ,q0,F)
  - DFA M = (Q′,Σ,δ′,q0′,F′)
- Where
  - q0′ = ε−close(q0)
    - ε−close(q) = {p E Q | δ(q, ε) = p}
    - Set of states form q0 reachable by ε transitions
    - Note: ε−close(P) = νpEP ε−close(p)
- Each state in Q’ is represented by a subset of Q





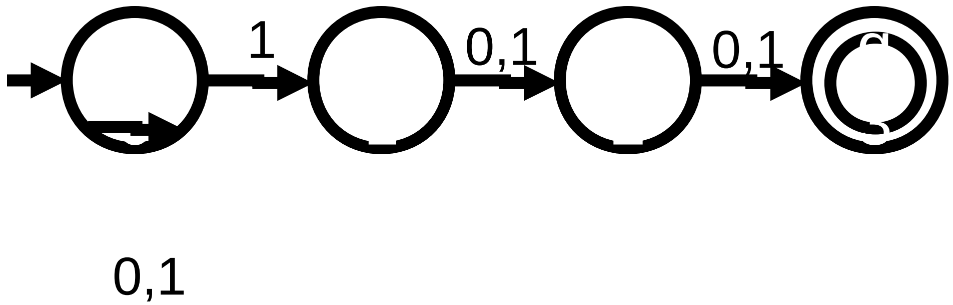
The δ’ Function for DFA M(L)

- The transition function δ’(q’,a) = p’ where
  - p‘ = ε−close(P)
  - P = {δ(q,a) | q E q’,}
  - Example: δ’({q0,q1,q2},0) = {q3,q4,q5}
- Lastly, F′ ={q′EQ′|F∩q′ ≠ A}
  - Every state in F ′ contains a state of an NFA that was in its final set.
  - Example: {q2,q3,q5}EF’



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

- q0’ = {q0}
- Q’ = (see table)
- δ’ = (see table)
- F’ = **(bolded states)**

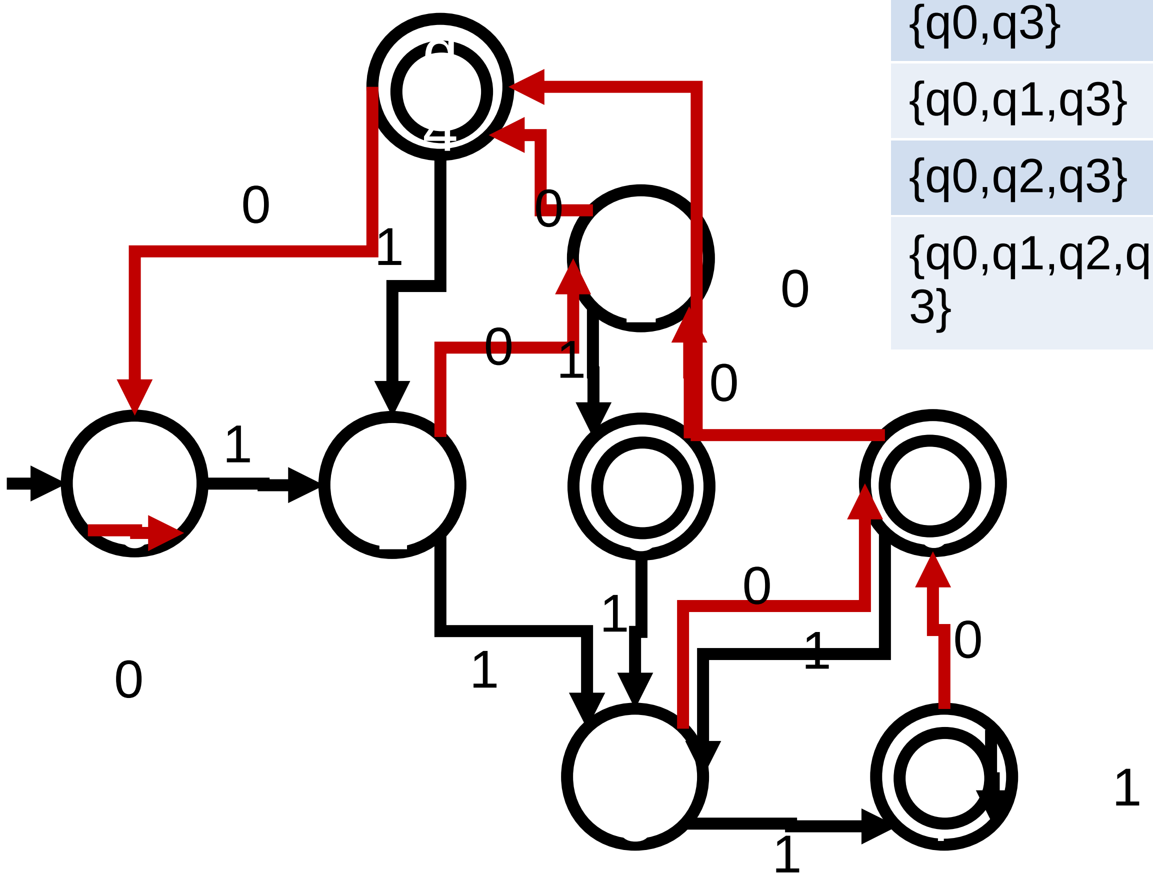


State	0	1
{q0}	{q0}	{q0,q1}
{q0,q1}	{q0,q2}	{q0,q1,q2}
{q0,q2}	{q0,q3}	{q0,q1,q3}
{q0.q1,q2}	{q0,q2,q3}	{q0,q1,q2,q3}
{q0,q3}	{q0}	{q0,q1}
{q0,q1,q3}	{q0,q2}	{q0,q1,q2}
{q0,q2,q3}	{q0,q3}	{q0,q1,q3}
{q0,q1,q2, q3}	{q0,q2,q3}	{q0,q1,q2,q3}

Example

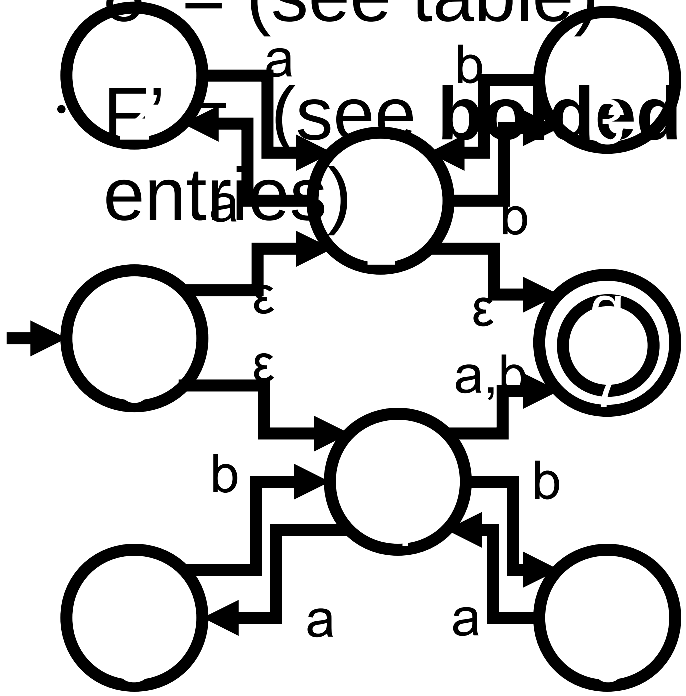
L=(0|1)*1(0|1)(0|1)

State	Q’	0	1
{q0}	q0’	{q0}	{q0,q1}
{q0,q1}	q1’	{q0,q2}	{q0,q1,q2}
{q0,q2}	q2’	{q0,q3}	{q0,q1,q3}
{q0.q1,q2}	q3’	{q0,q2,q3}	{q0,q1,q2,q3}
{q0,q3}	q4’	{q0}	{q0,q1}
{q0,q1,q3}	q5’	{q0,q2}	{q0,q1,q2}
{q0,q2,q3}	q6’	{q0,q3}	{q0,q1,q3}
{q0,q1,q2,q3}	q7’	{q0,q2,q3}	{q0,q1,q2,q3}



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

- q0’ = {q0,q1,q4,q7}
- Q’ = (see table)
- δ’ = (see table)
- F’ = (see bolded entries)

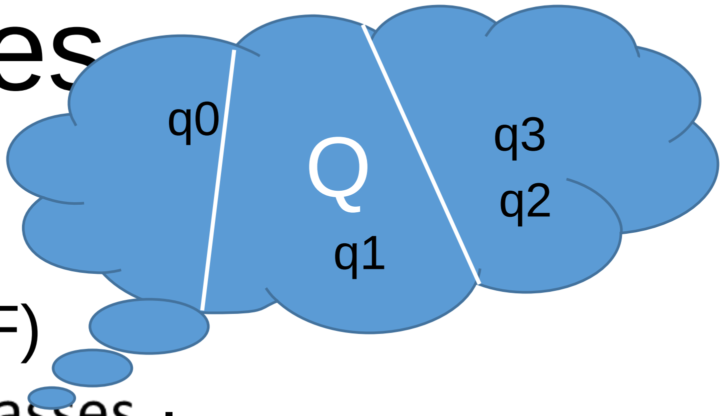


State	a	b
{q0,q1, q4,q7}	{q2,q5,q7}	{q3,q6,q7}
{q2,q5,q7}	{q1,q7}	{q4}
{q3,q6,q7}	{q4}	{q1,q7}
{q1,q7}	{q2}	{q3}
{q4}	{q5,q7}	{q6,q7}
{q2}	{q1,q7}	A
{q3}	A	{q1,q7}
{q5,q7}	A	{q4}
{q6,q7}	{q4}	A

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

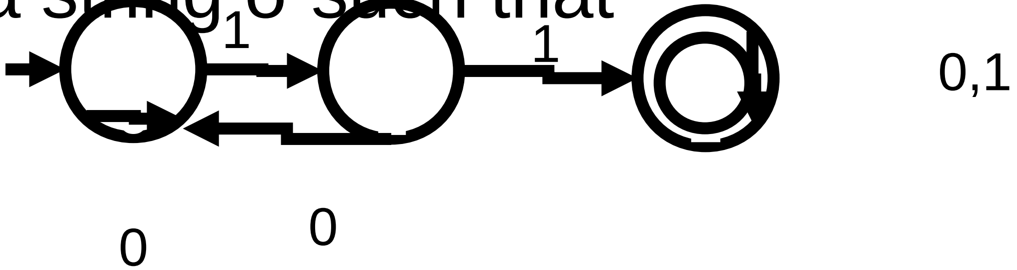
Equivalence Classes



- Start with a DFA M = (Q,Σ,δ,q0,F)
- **Idea**: Divide Q into equivalence classes.
- The classes represent the states of the minimal DFA
- **Definition**: q1 and q2 are *equivalent* (in the same class) means for all σ E ΣX, δ(q1,σ) E F if and only if δ(q2,σ) E F
- I.e., If there exists a string σ such that
  - δ(q1,σ) E F
  - δ(q2,σ) F

- 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



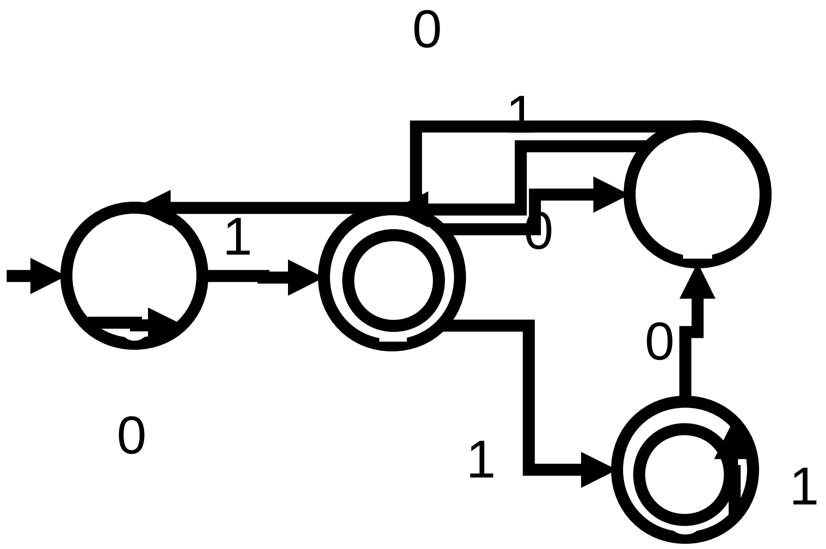
Minimization Procedure

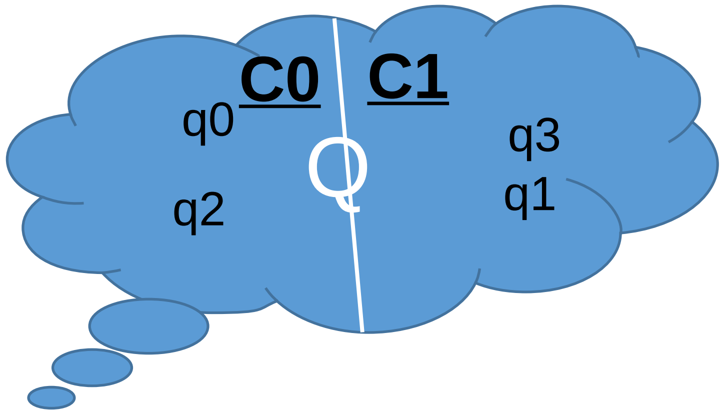
- Initially all states are either accepting or not
- **If** there is a class C and character a E Σ such that

  {δ(qi,a)|qi E C} are in k > 1 equivalence classes
- **Then** Split C into k classes Cj such that

  δ(qi, a), where qi E Ck, are in the same equivalence class.
- Repeat until no more splits are needed.

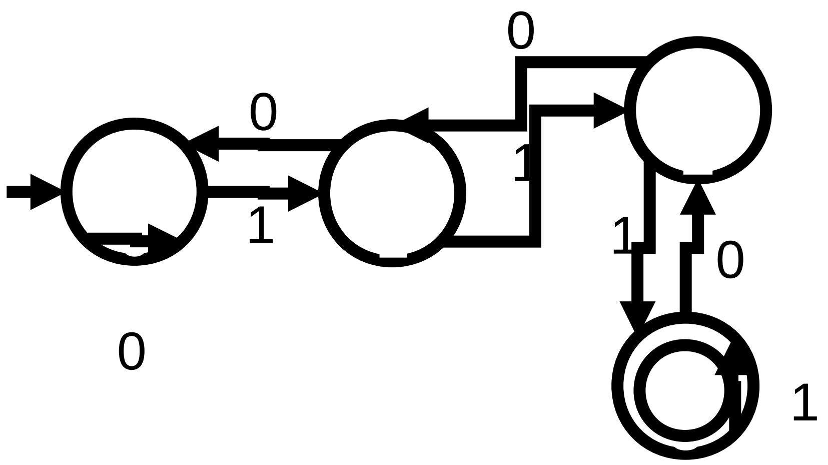
Example 1

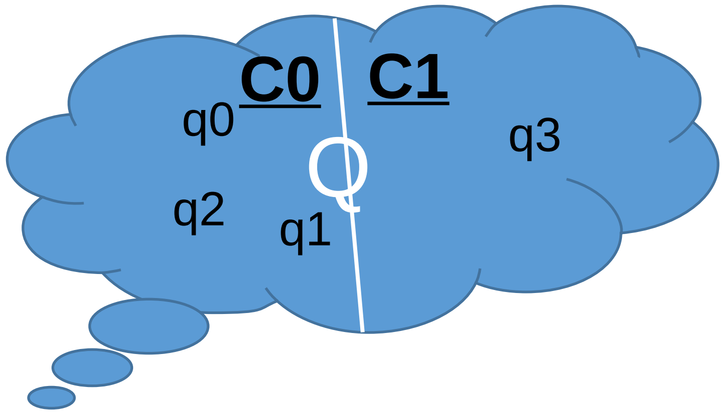




	Q	0	1
C0	q0	C0	C1
	q2	C0	C1
C1	q1	C1	C0
	q3	C1	C0

Example 2





	Q	0	1	0	1	0	1
C0	q0	C0	C0	C0	C0		
	q2	C0	C0	C0	C2	🟨	🟨
	q1	C0	C1	🟩	🟩	🟩	🟩
C1	q3						

C3

C2