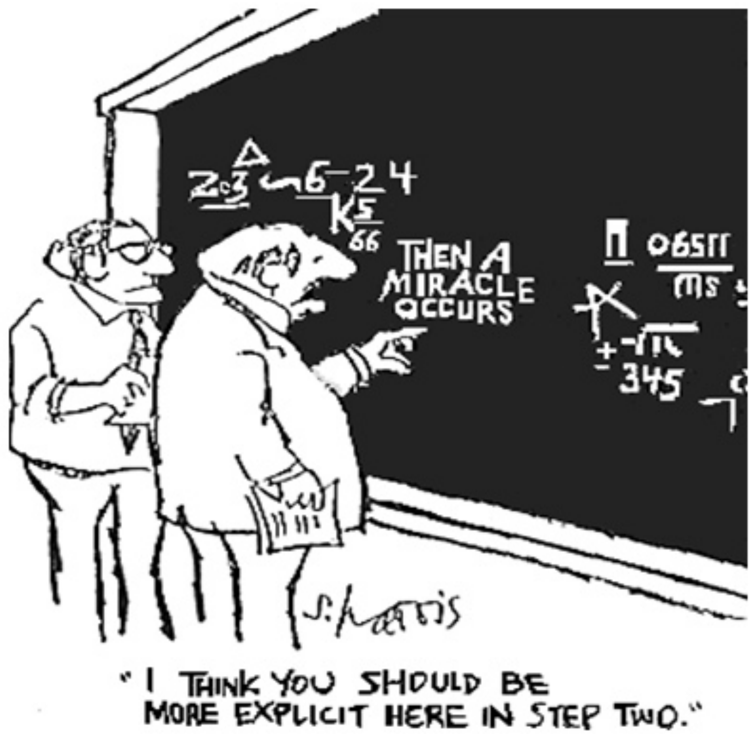
CSE 311: Foundations of Computing

Lecture 23: Finite State Machine Minimization & NFAs

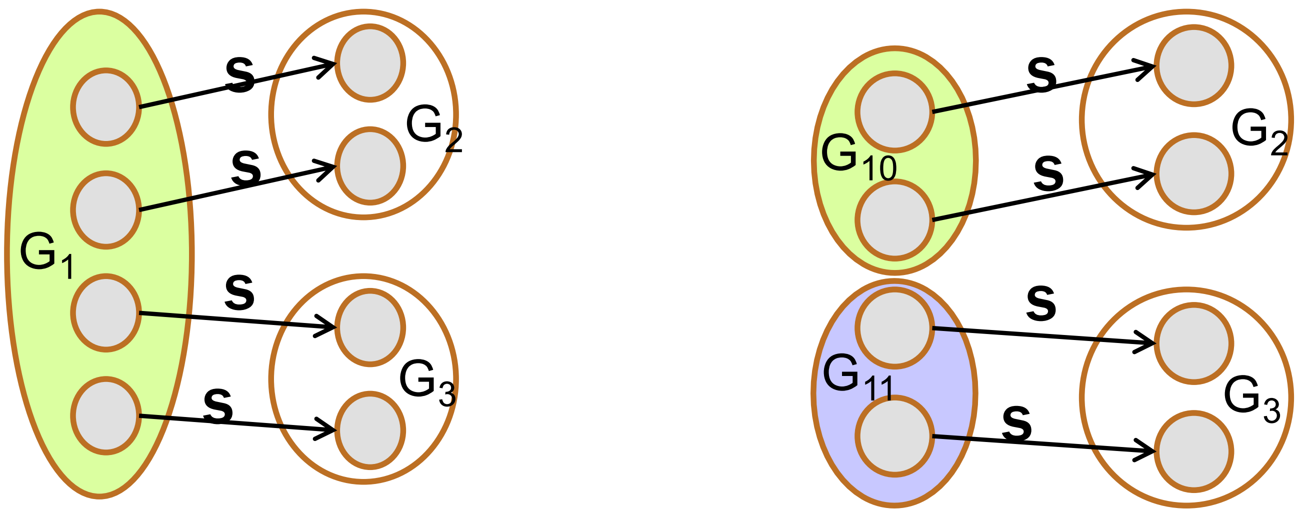


State Minimization

- Many different FSMs (DFAs) for the same problem
- Take a given FSM and try to reduce its state set by combining states
  - Algorithm will always produce the unique minimal equivalent machine (up to renaming of states) but we won’t prove this

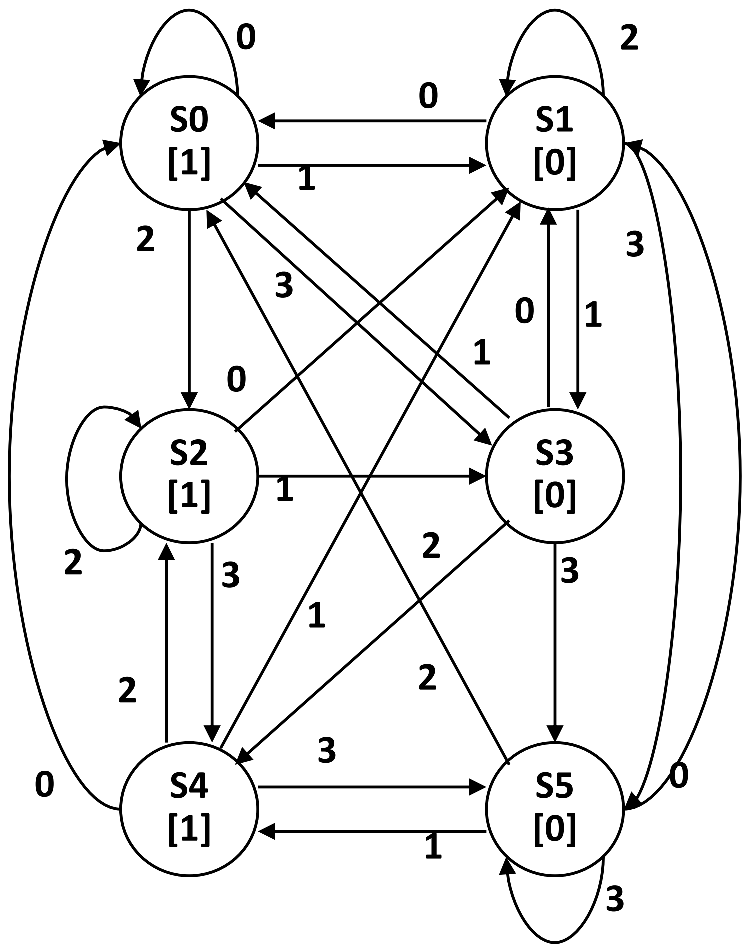
State Minimization Algorithm

1. Put states into groups based on their outputs (or whether they are final states or not)
2. Repeat the following until no change happens
   a. If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** into smaller groups based on which group the states go to on **s**



3. Finally, convert groups to states

State Minimization Example

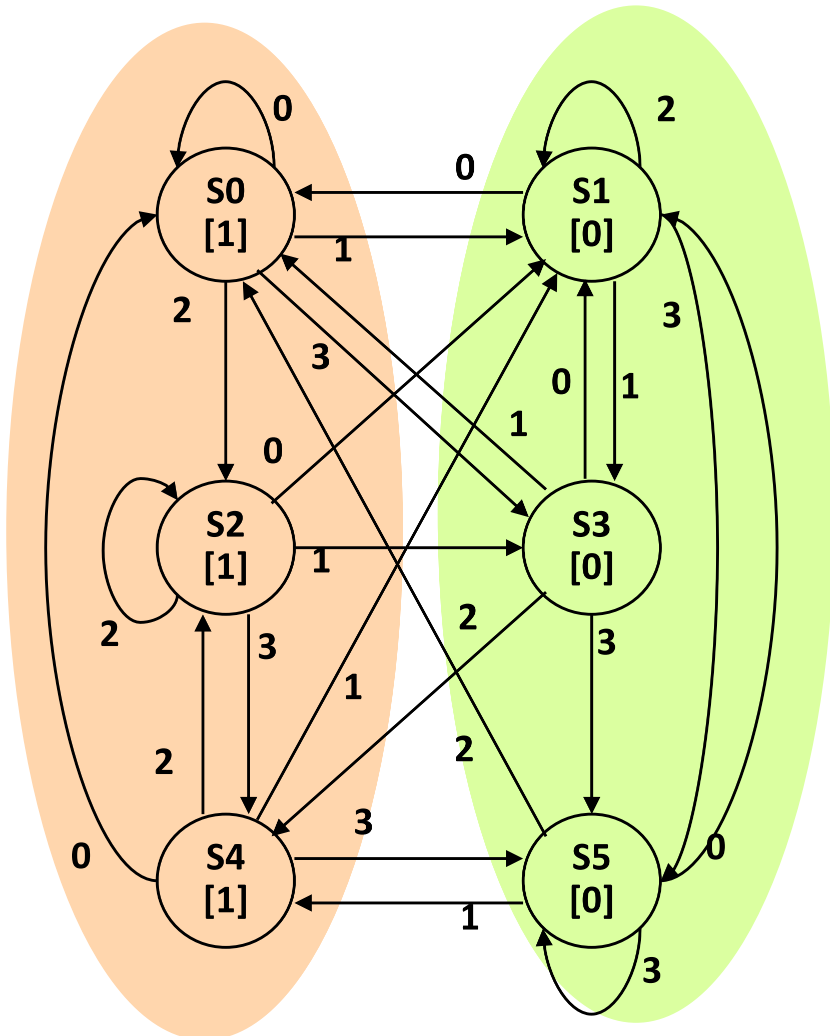


present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state
transition table

Put states into groups based on their outputs (or whether they are final states or not)

State Minimization Example

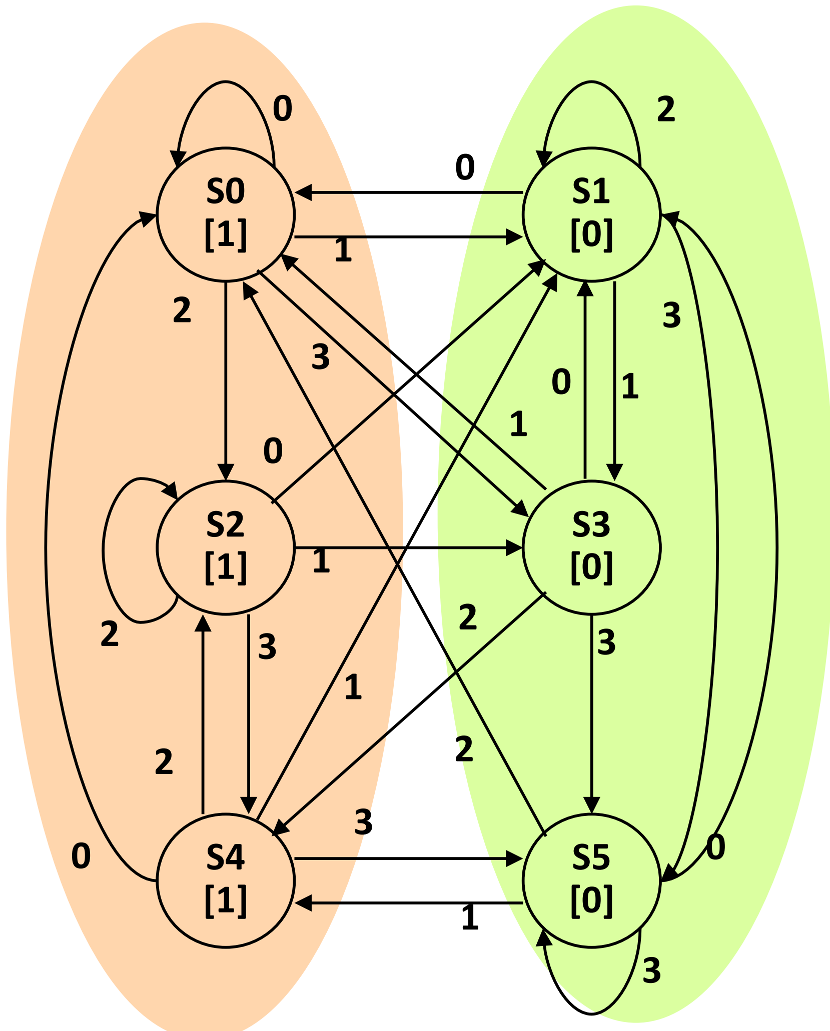


present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state
transition table

Put states into groups based on their outputs (or whether they are final states or not)

State Minimization Example



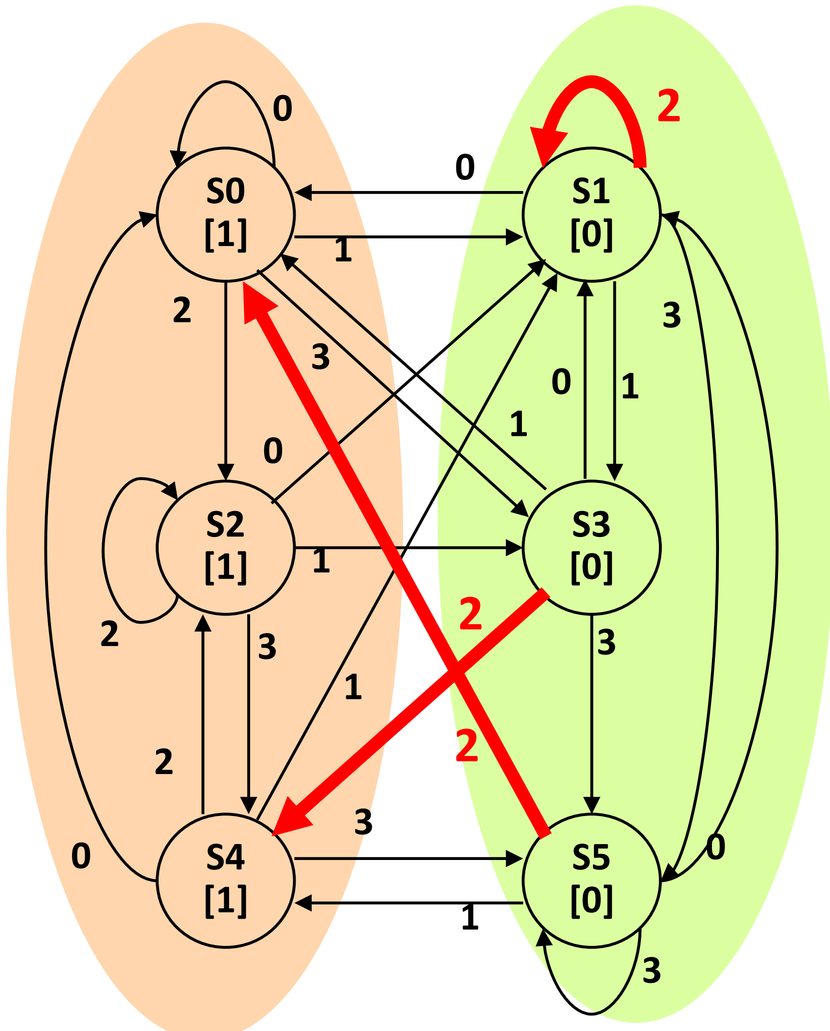
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



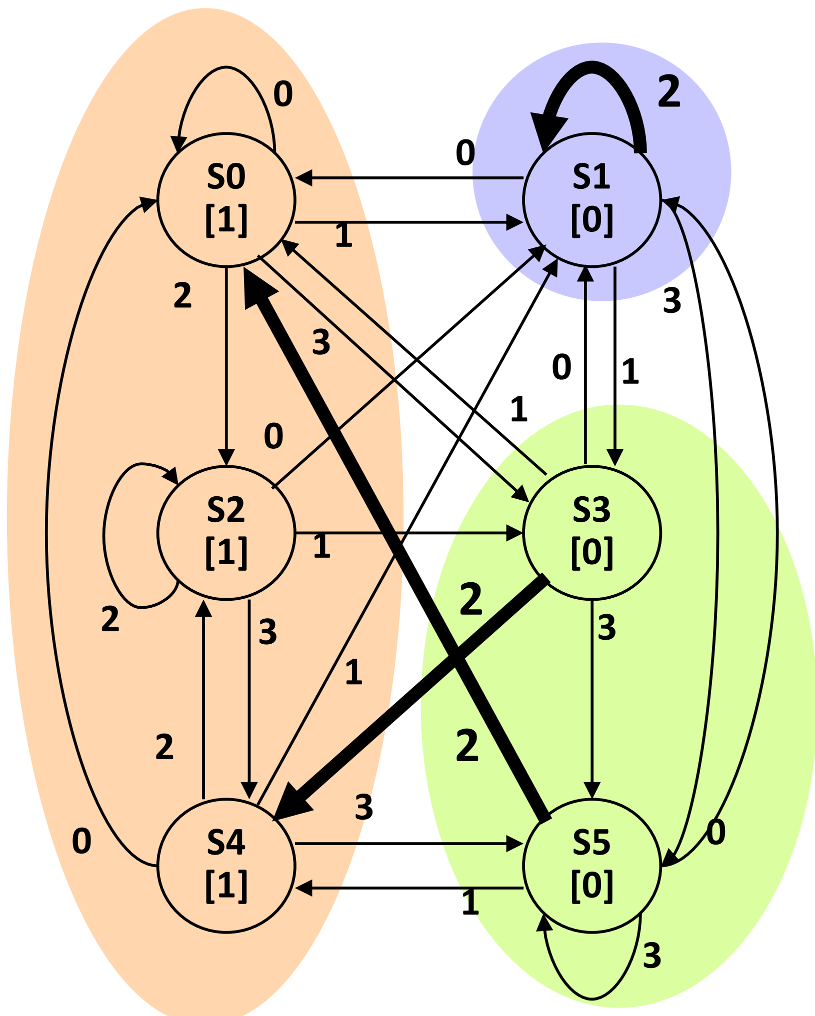
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



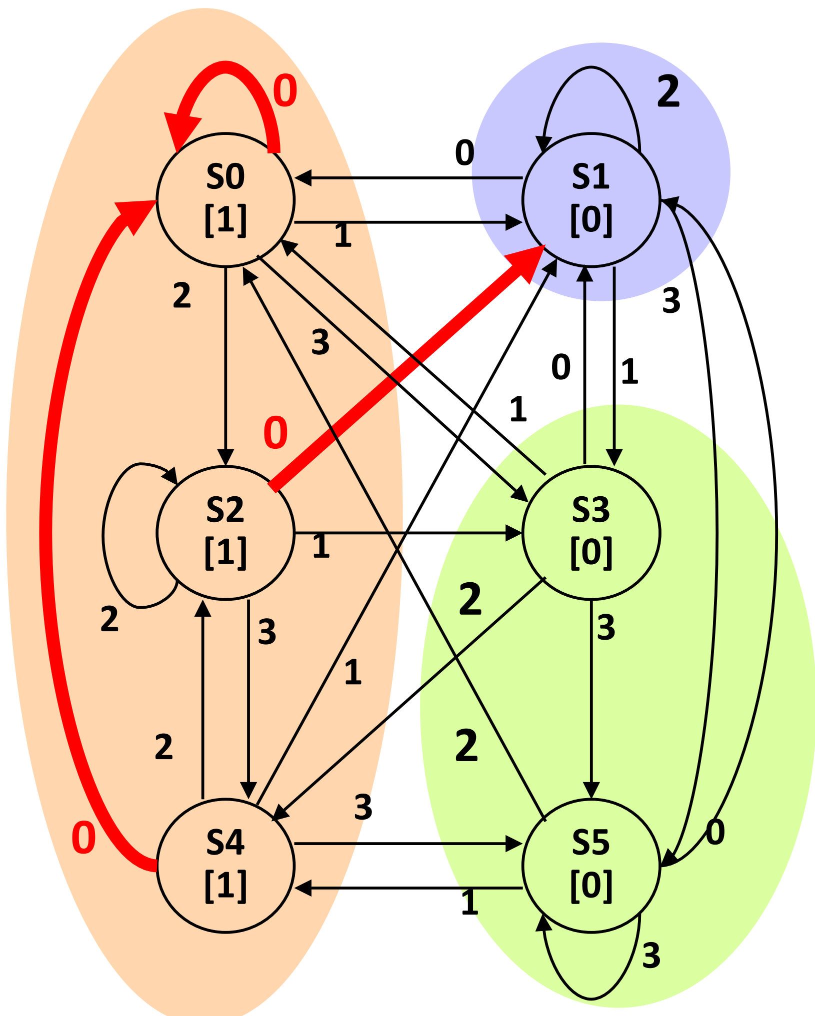
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state
transition table

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



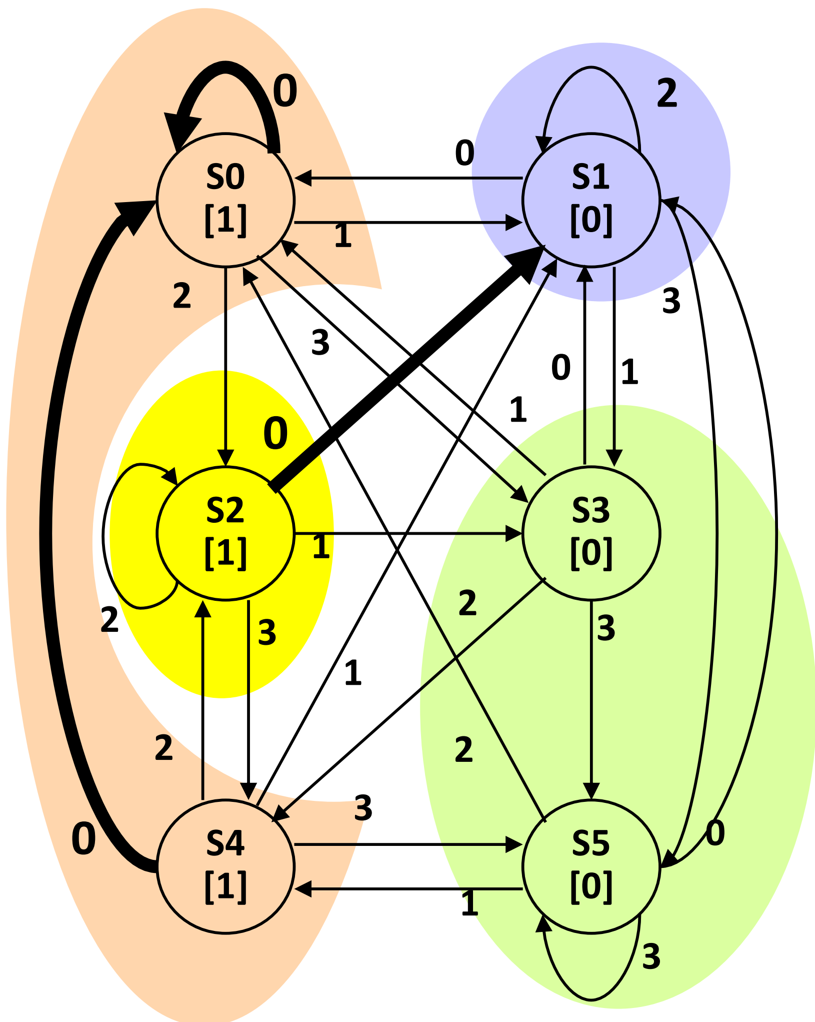
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



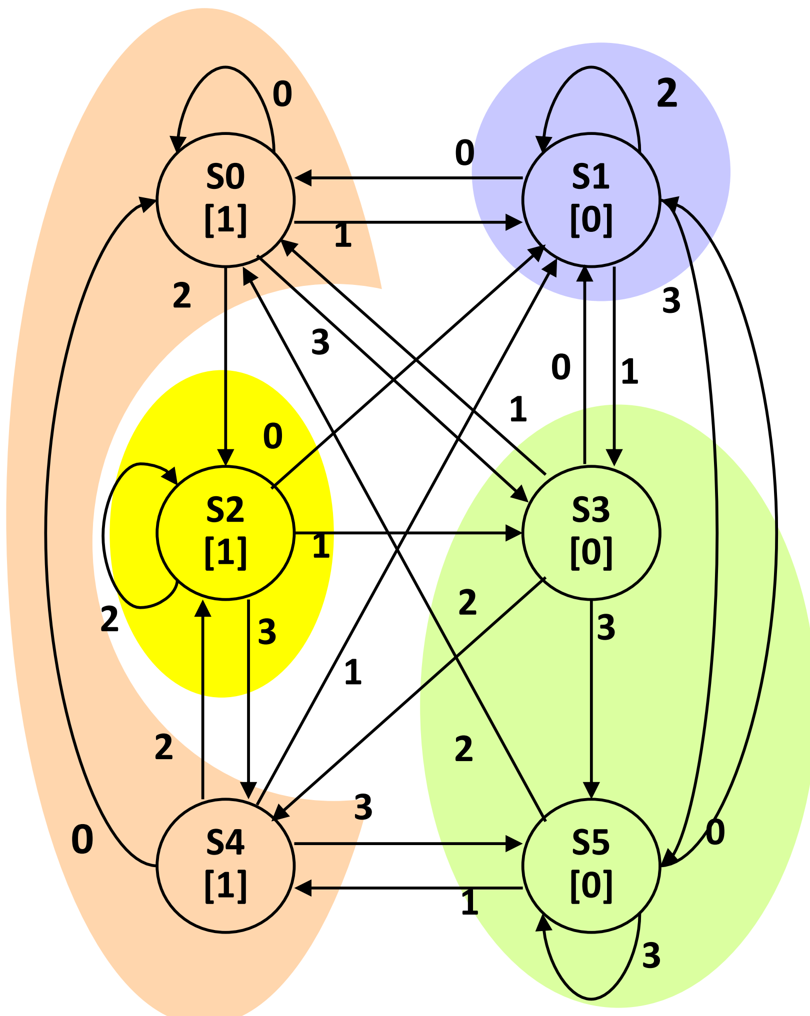
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

Put states into groups based on their outputs (or whether they are final states or not)

If there is a symbol **s** so that not all states in a group **G** agree on which group **s** leads to, split **G** based on which group the states go to on **s**

State Minimization Example



present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

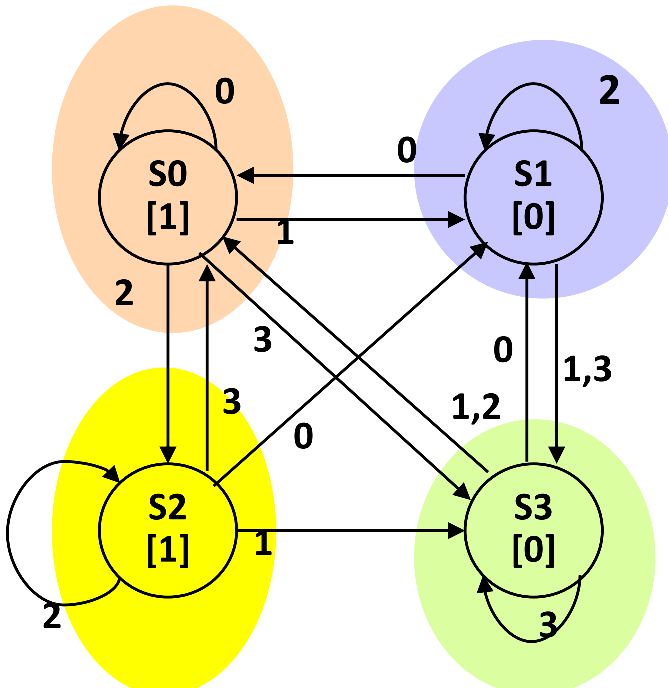
state transition table

Finally convert groups to states:

Can combine states S0-S4 and S3-S5.

In table replace all S4 with S0 and all S5 with S3

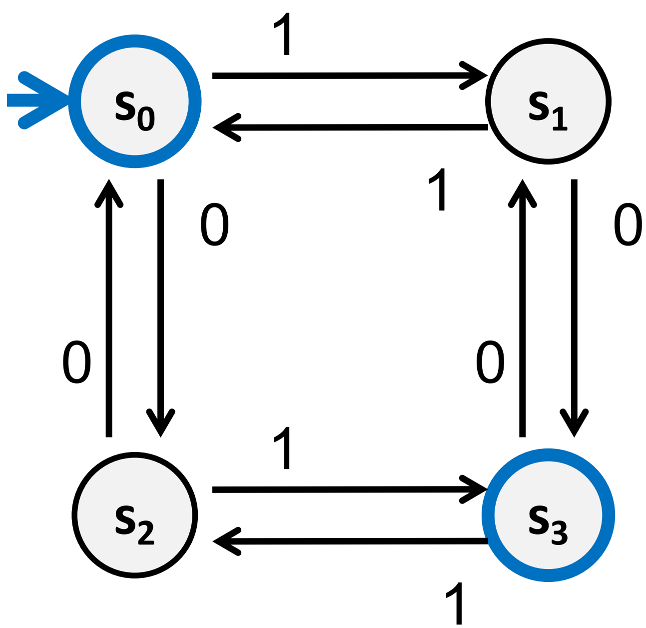
Minimized Machine



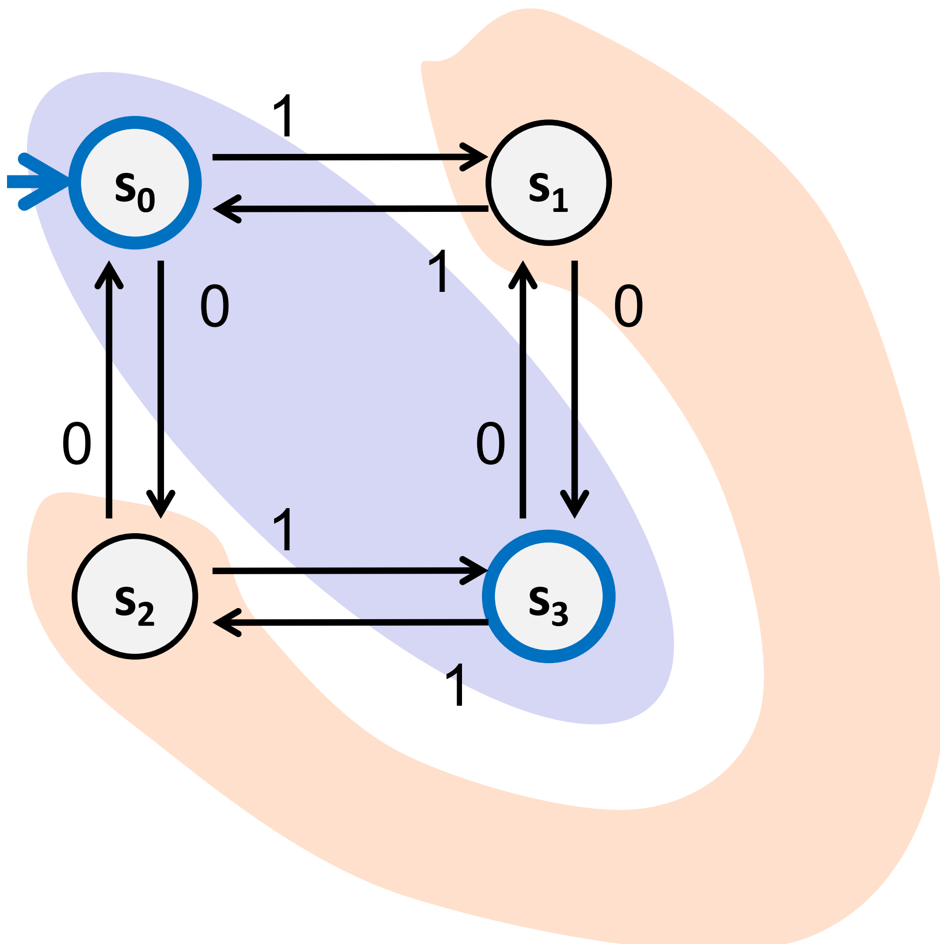
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S3	0
S2	S1	S3	S2	S0	1
S3	S1	S0	S0	S3	0

state
transition table

A Simpler Minimization Example



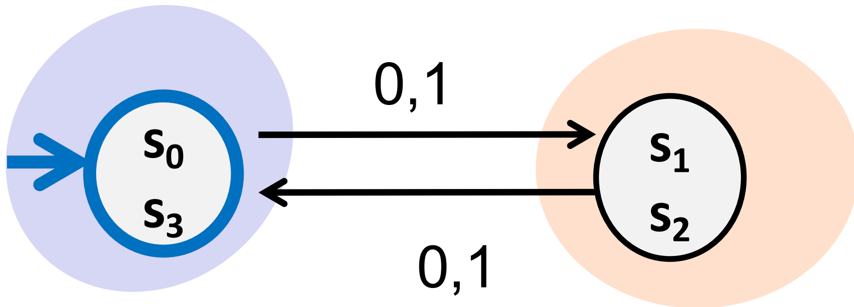
A Simpler Minimization Example



Split states into final/non-final groups

Every symbol causes the DFA to go from one group to the other so neither group needs to be split

Minimized DFA



Partial Correctness of Minimization Algorithm

- **Prove this claim: after processing input x, if the old machine was in state q, then the new machine is in the state S with $q \in S$**
  - **True after 0 characters processed**
  - **If true after k characters processed, then it's true after k+1 characters processed:**

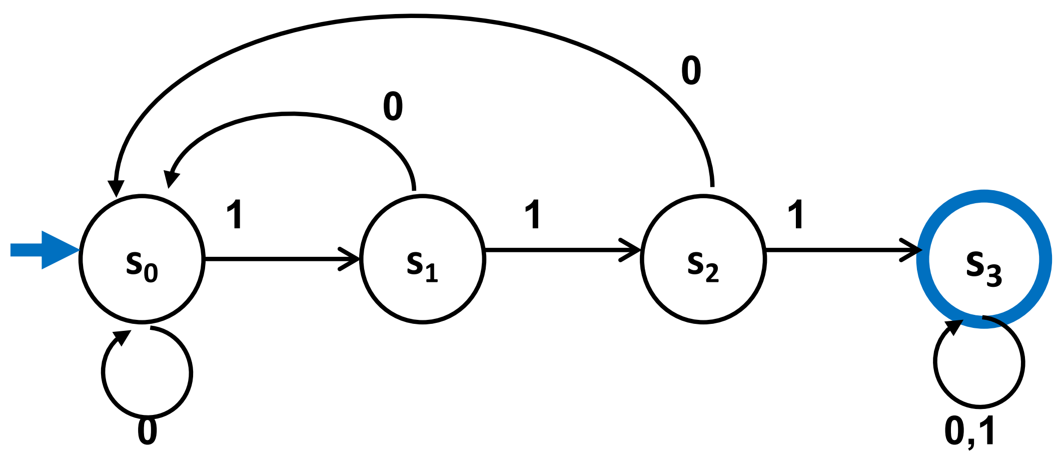
    By inductive hypothesis, after k steps, old machine is in state q and new one in state S with $q \in S$

    By construction, every $r \in S$ is taken to the same state S' on input x_{k+1}, so q is taken to some $q' \in S'$.
  - **At end, since every $r \in S$ is accepting or rejecting, new machine gives correct answer.**

Another way to look at DFAs

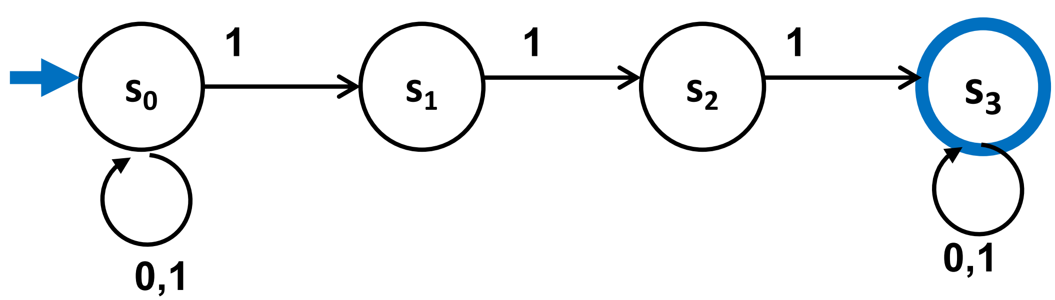
Definition: The label of a path in a DFA is the concatenation of all the labels on its edges in order

Lemma: x is in the language recognized by a DFA iff x labels a path from the start state to some final state

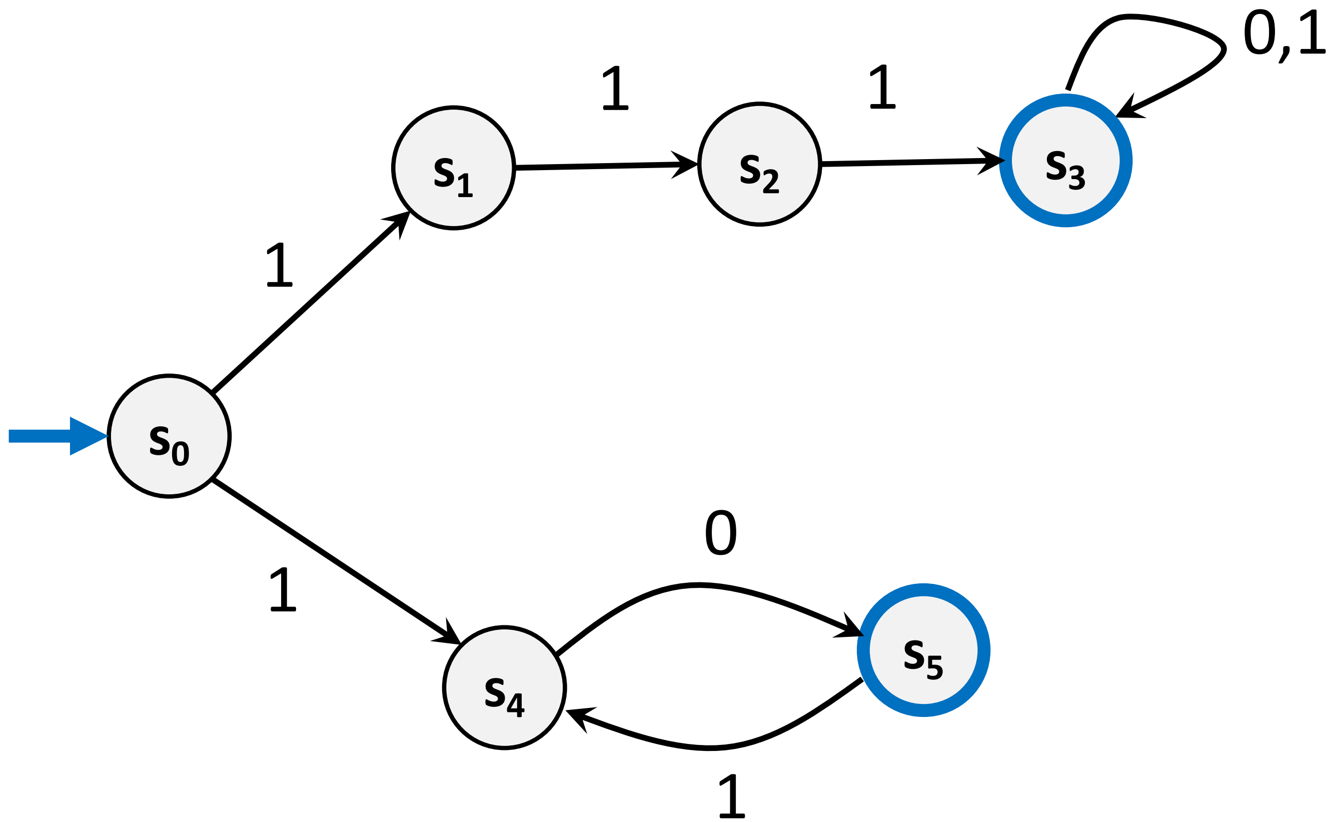


Nondeterministic Finite Automata (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol— can have 0 or >1
  - Also can have edges labeled by empty string ε
- **Definition:** x is in the language recognized by an NFA if and only if x labels some path from the start state to a final state

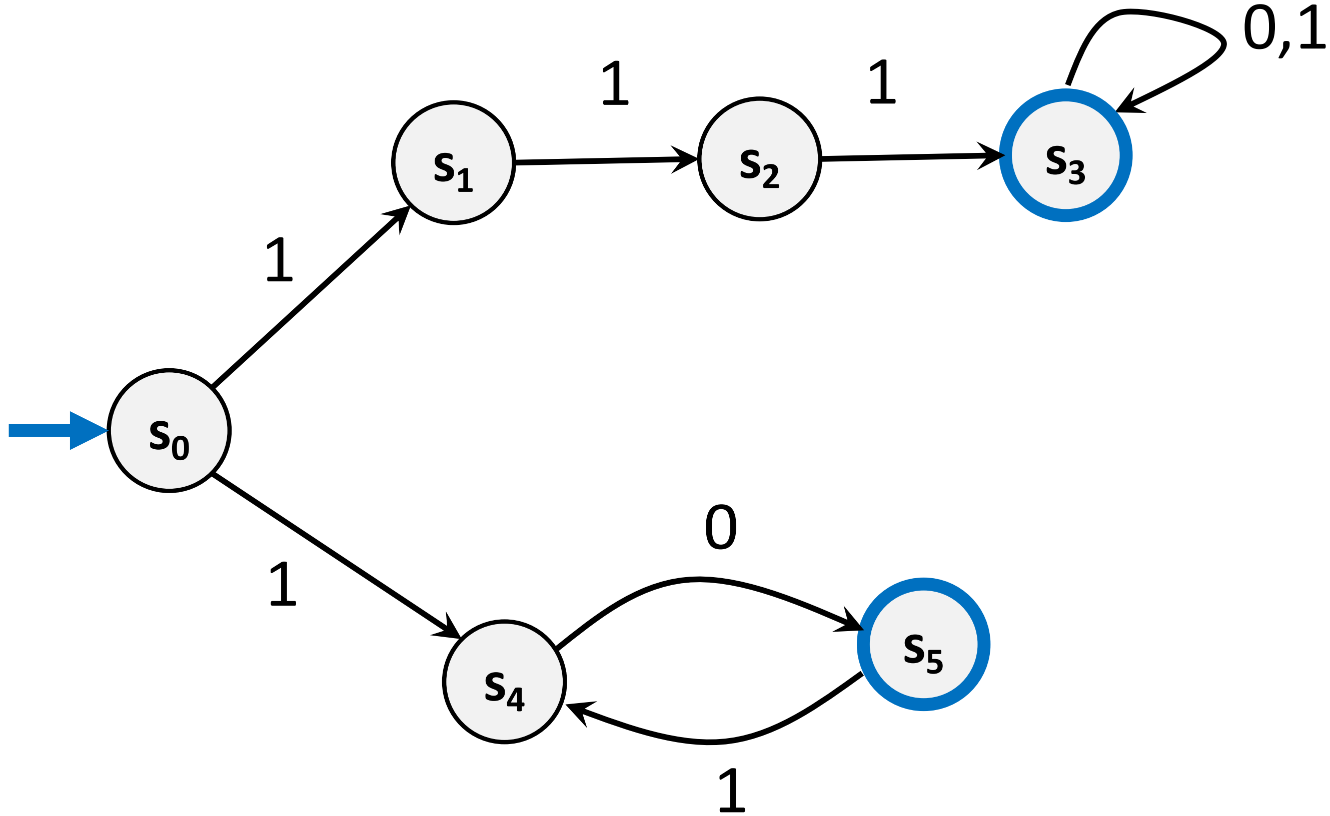


Consider This NFA



What language does this NFA accept?

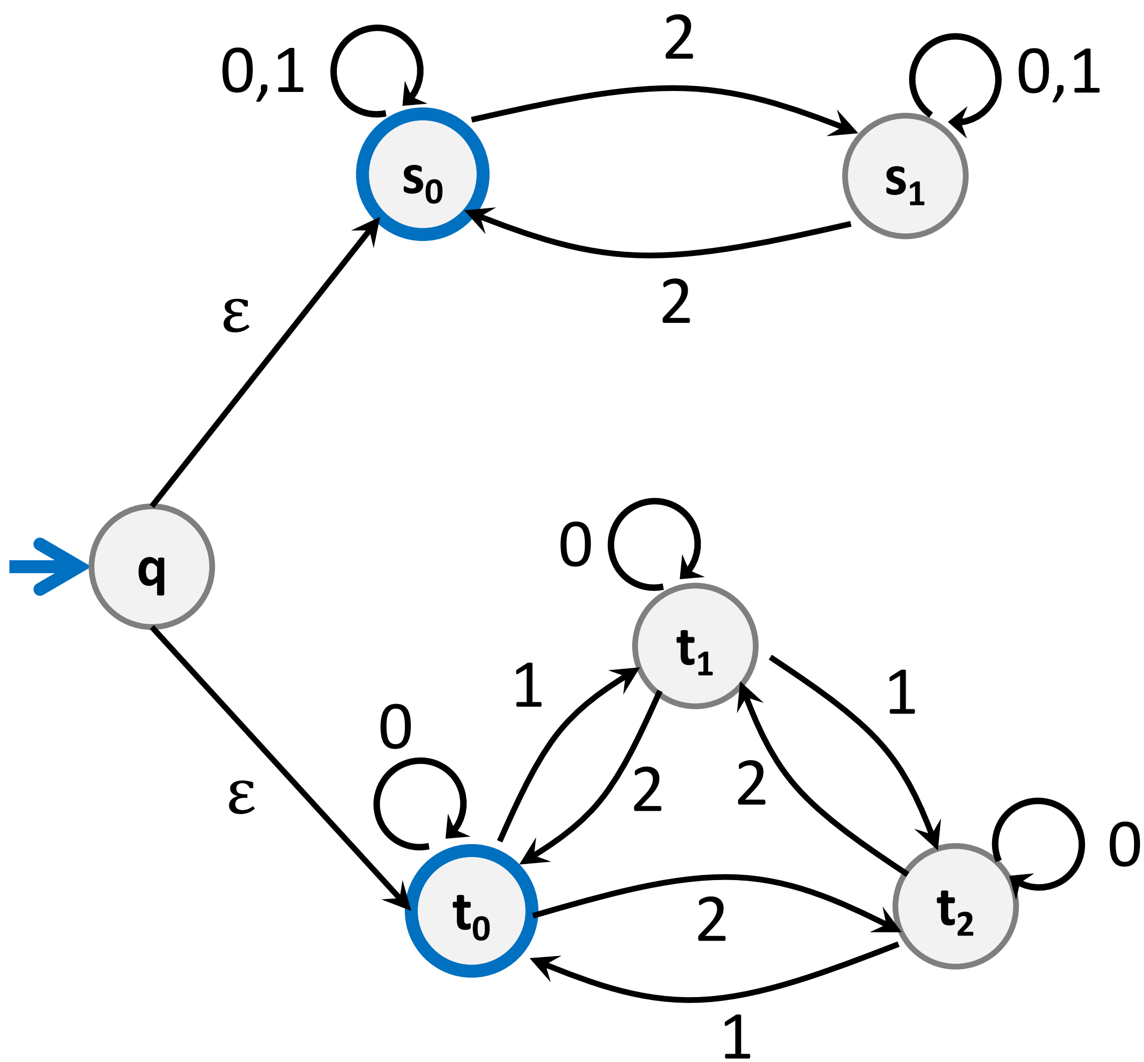
Consider This NFA



What language does this NFA accept?

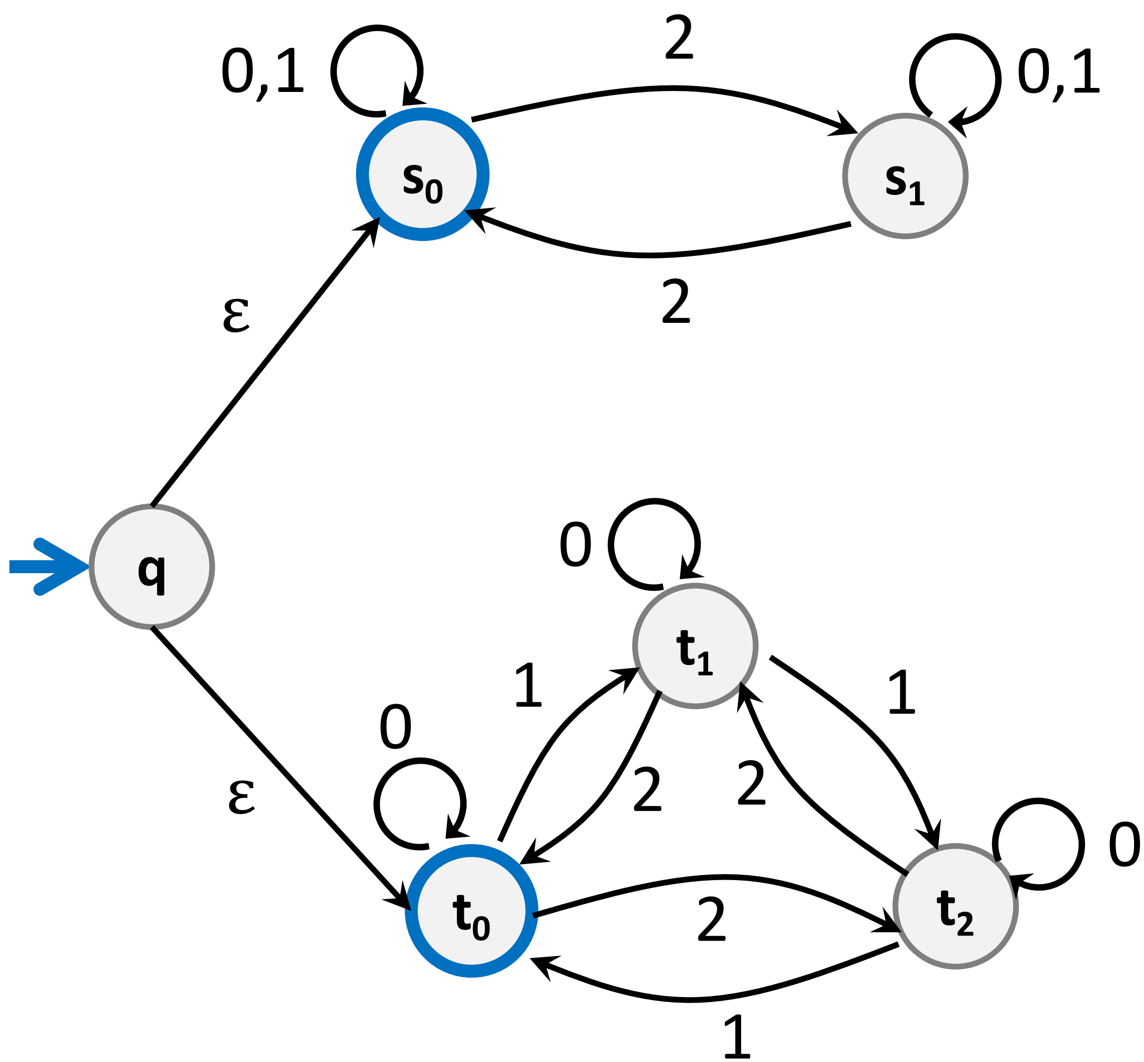
10(10)* U 111 (0 U 1)*

NFA ε-moves



NFA ε-moves

Strings over {0,1,2} w/even # of 2's OR sum to 0 mod 3



Three ways of thinking about NFAs

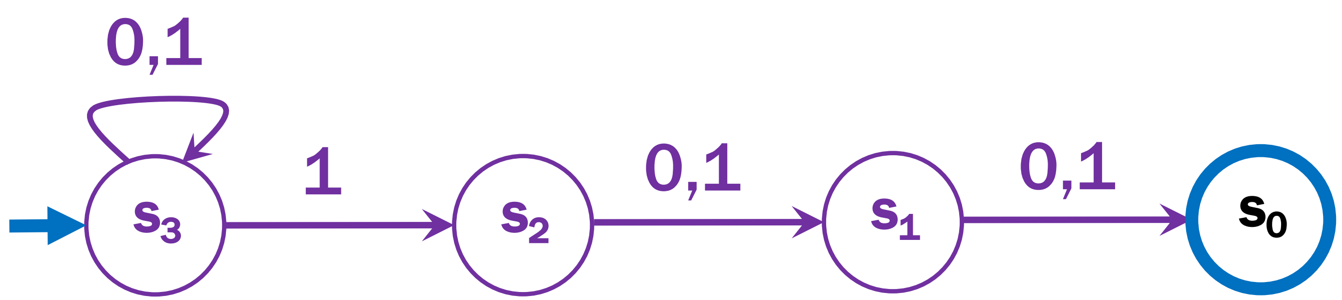
- Outside observer: Is there a path labeled by x from the start state to some final state?

- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)

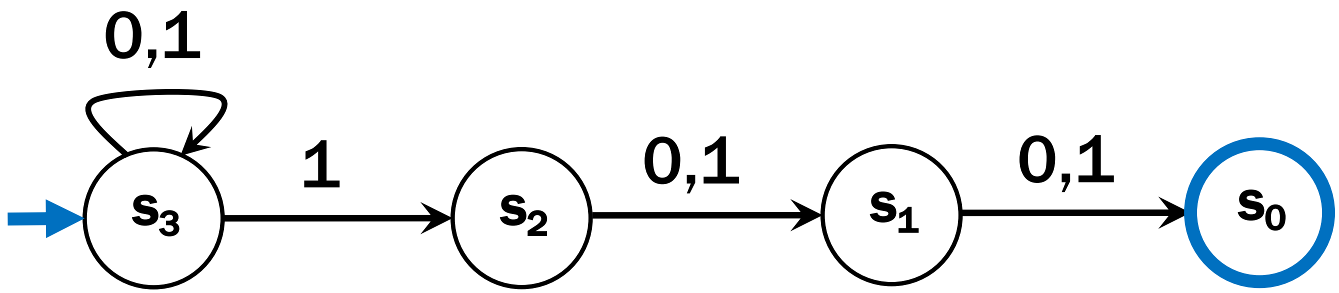
- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

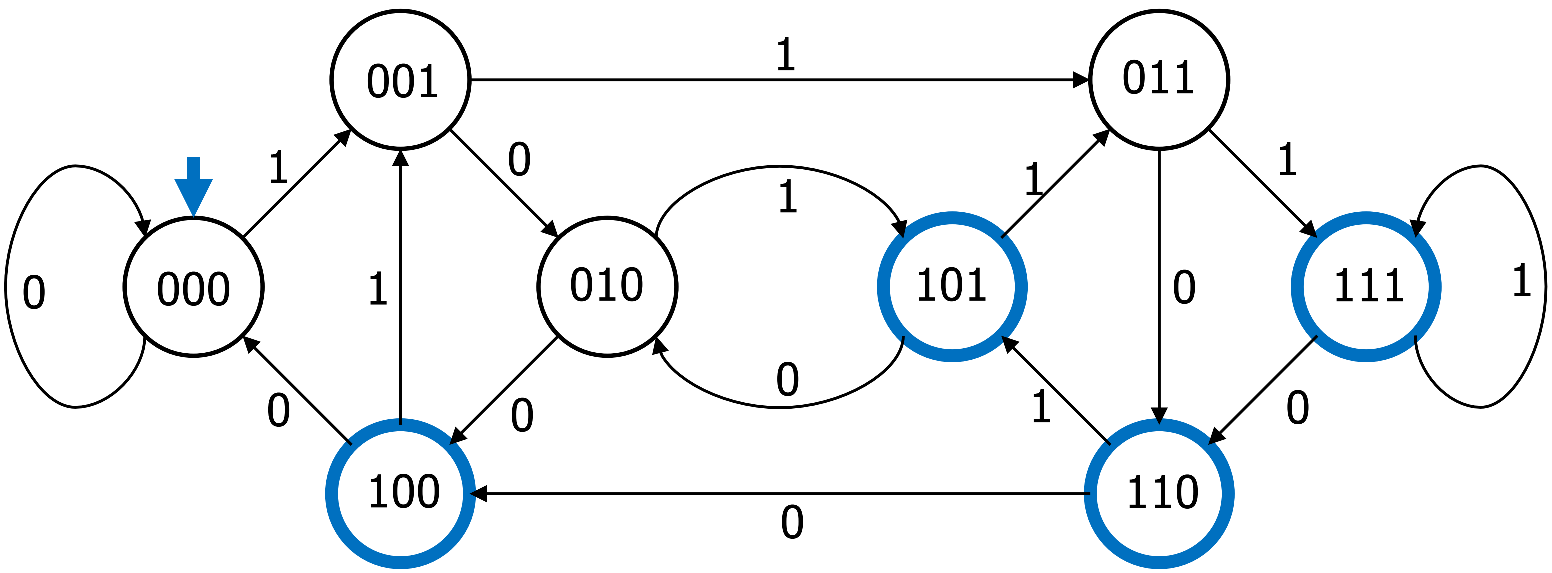
NFA for set of binary strings with a 1 in the 3rd position from the end

NFA for set of binary strings with a 1 in the 3rd position from the end

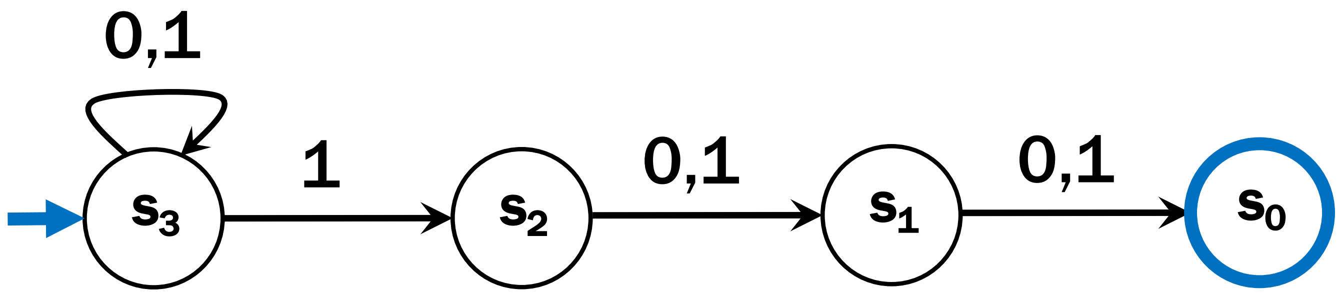


Compare with the smallest DFA

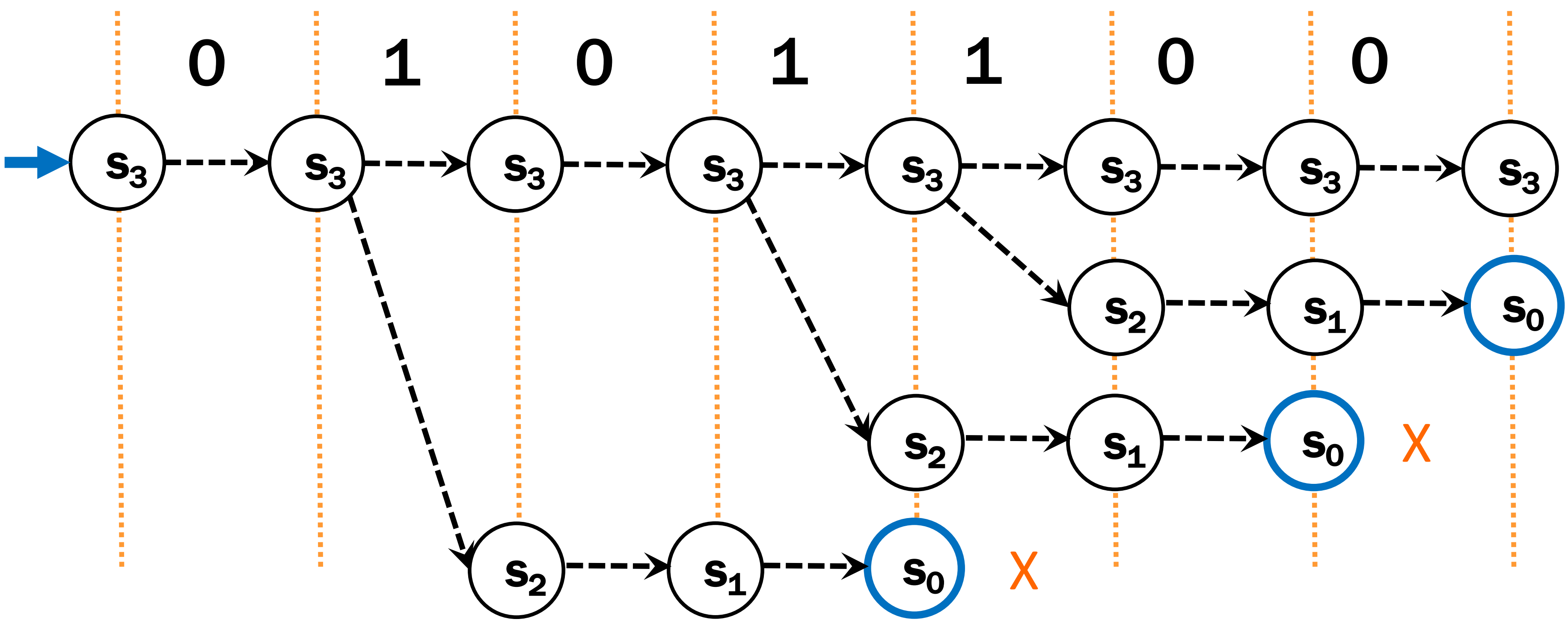




Parallel Exploration view of an NFA



Input string 0101100



NFAs and regular expressions

Theorem: For any set of strings (language) A described by a regular expression, there is an NFA that recognizes A.

Proof idea: Structural induction based on the recursive definition of regular expressions...

Regular Expressions over Σ

- **Basis:**
  - $\varnothing$, ε are regular expressions
  - ***a*** is a regular expression for any *a* $\in \Sigma$
- **Recursive step:**
  - If **A** and **B** are regular expressions then so are:

    (A $\cup$ B)

    (AB)

    A*

Base Case

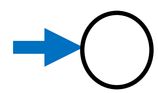
- **Case $\varnothing$:**

- **Case ε:**

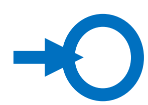
- **Case *a*:**

Base Case

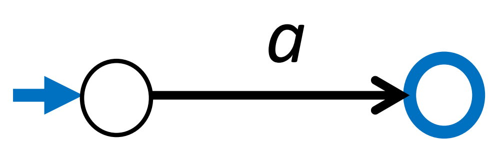
- **Case ∅:**



- **Case ε:**

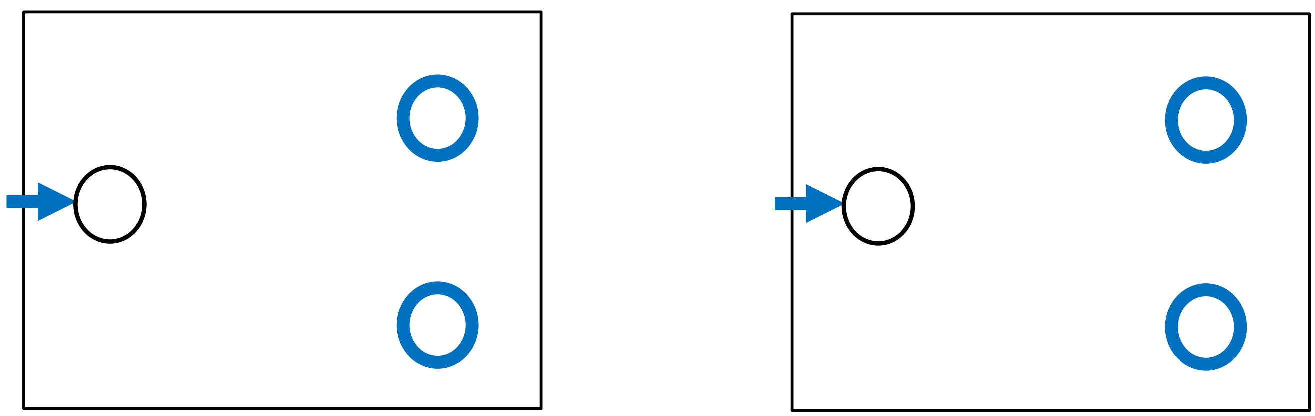


- **Case *a*:**



Inductive Hypothesis

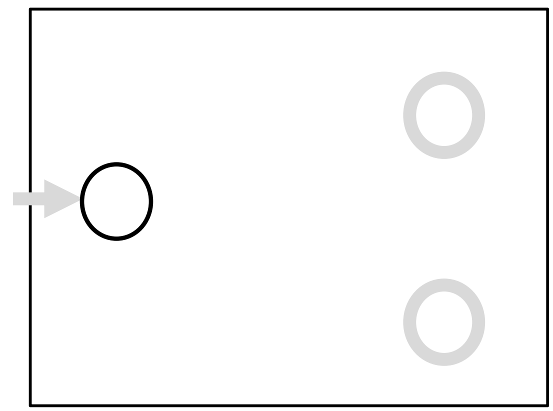
- Suppose that for some regular expressions A and B there exist NFAs N_A and N_B such that N_A recognizes the language given by A and N_B recognizes the language given by B



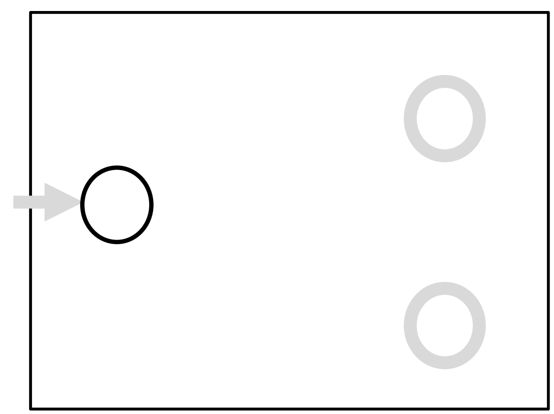
N_A N_B

Inductive Step

Case (A ∪ B):



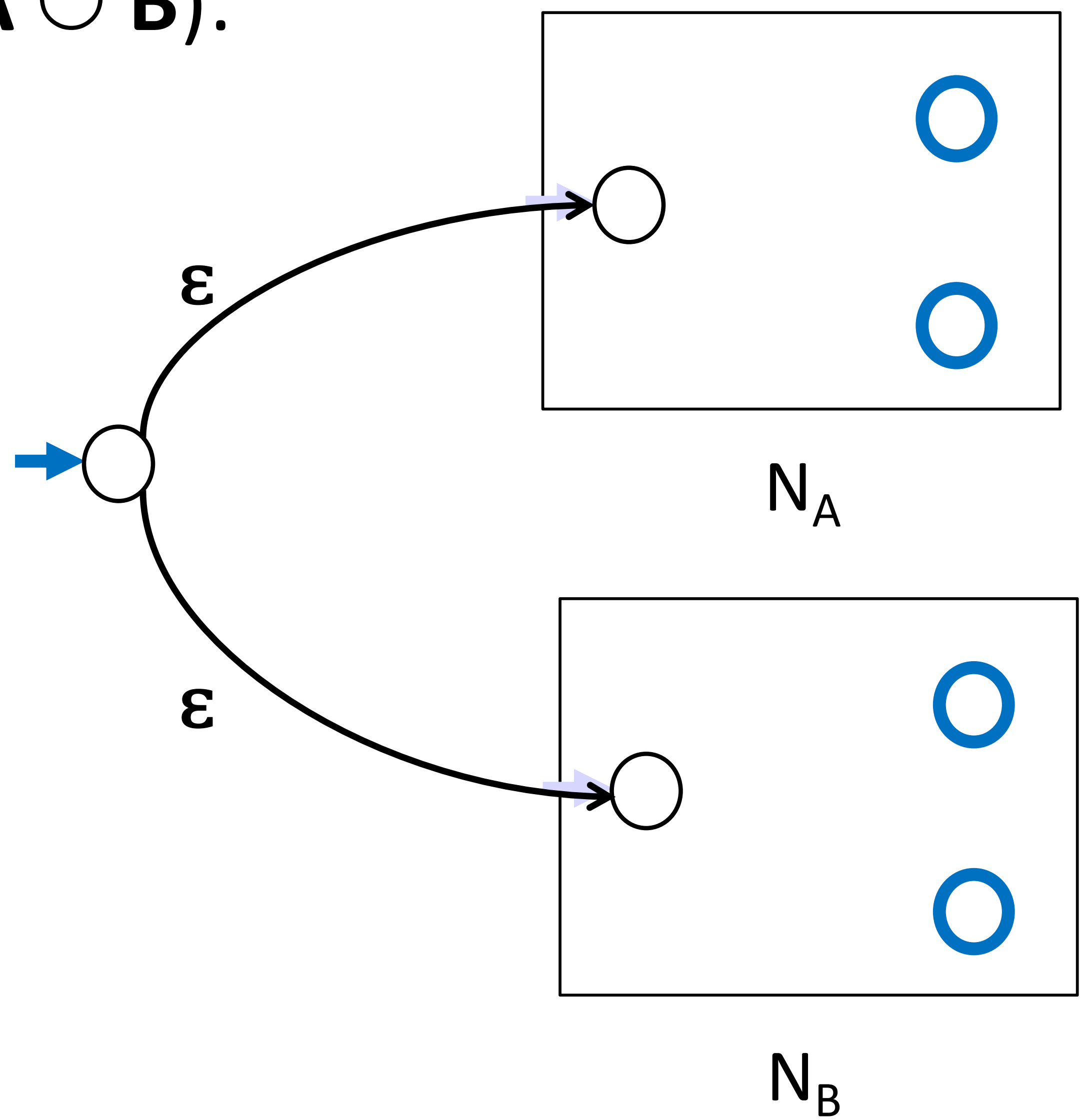
N_A



N_B

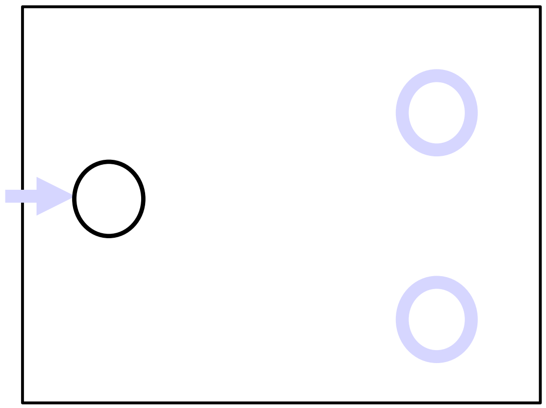
Inductive Step

Case (A ∪ B):

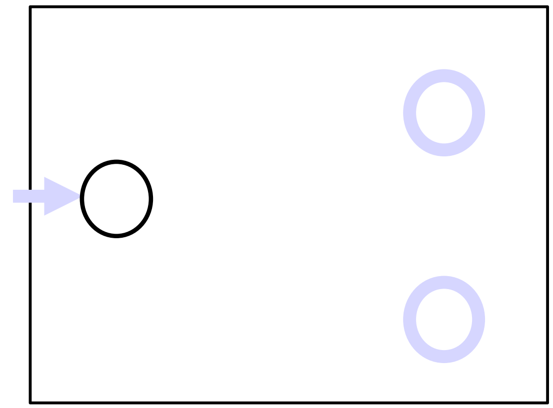


Inductive Step

Case (AB):



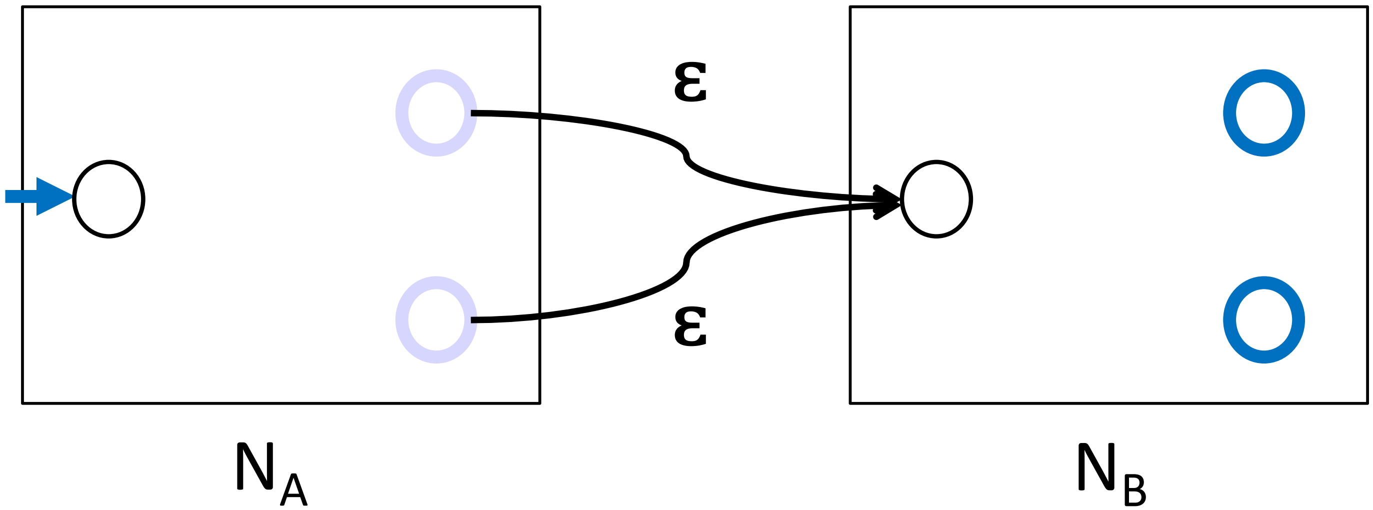
N_A



N_B

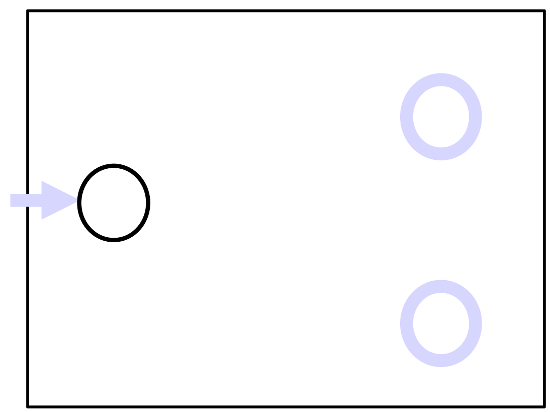
Inductive Step

Case (AB):



Inductive Step

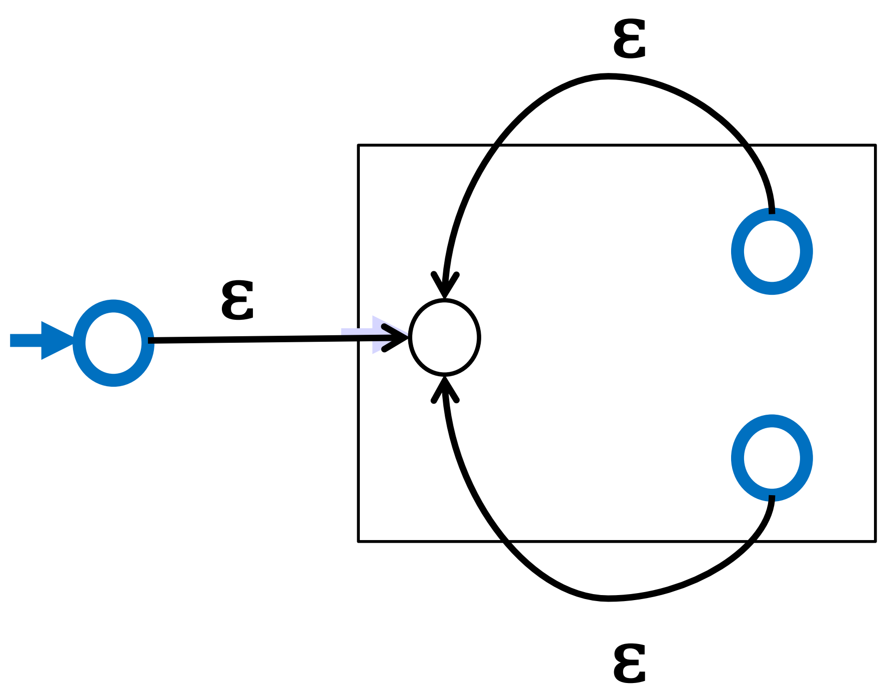
Case A*



N_A

Inductive Step

Case A*



N_A

Build an NFA for $(01 \cup 1)^*0$

Solution

(01 ∪1)*0