cse 311: foundations of computing

Spring 2015

Lecture 23: State minimization and NFAs

induction and grammars

Prove that the following grammar generates all strings with matched (): $\overbrace{S \to SS \mid (S) \mid \varepsilon}^{G}$.

P(n) = " $\forall$ string T len n with matched, G generates T "

Base: P(0) T = ε ✓

IH: For some $k \geq 0$ and any $0 \leq j \leq k$, P(j) holds.

IS: Want to prove P(k+1).

Fix ~~T~~ len(T) = k+1 has matched ().

$$\underbrace{(\overset{A}{\cdots\cdots})}_{j}\ \overset{B}{\cdots\cdots}_{k+1-j-2}$$

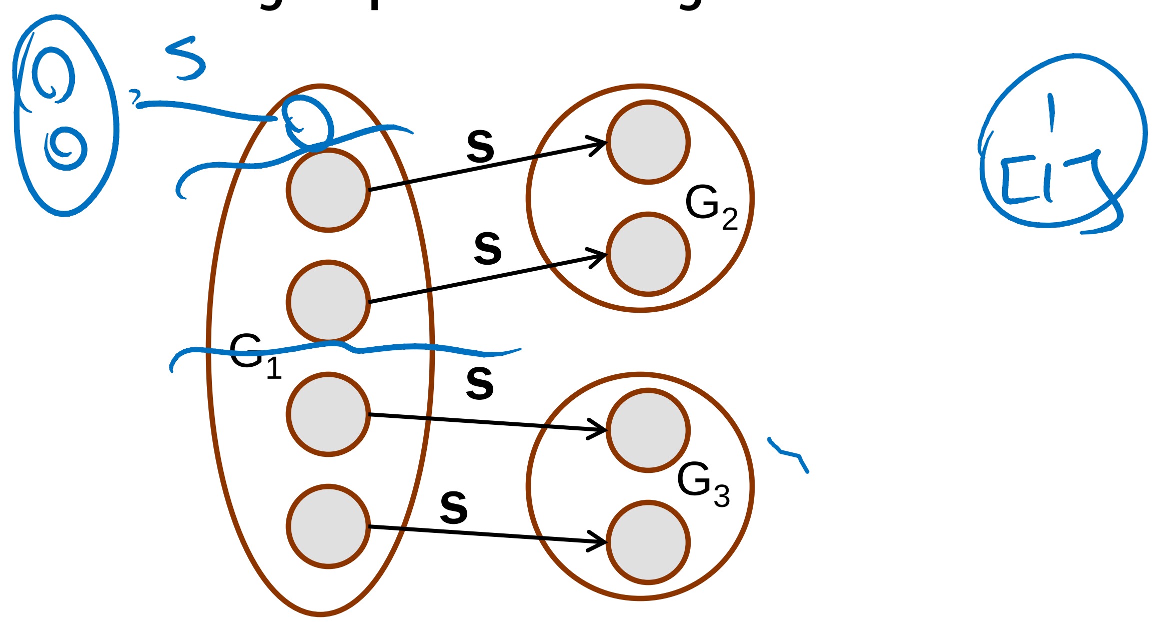
$$S \to SS \to (S)S \to \cdots \xrightarrow{\text{by IH}} (A)S \to \cdots \xrightarrow{\text{by IH}} (A)B$$

state minimization

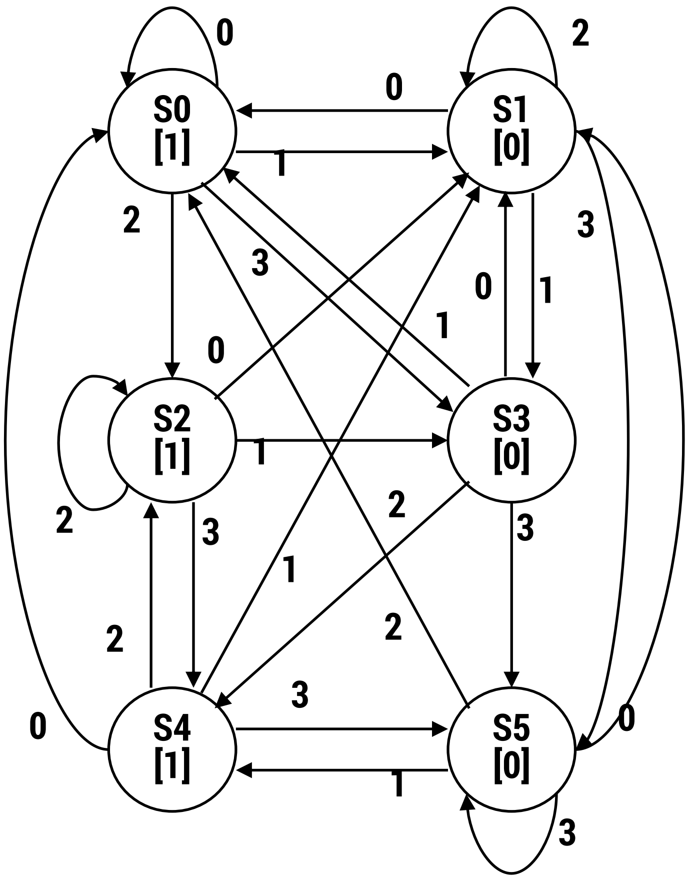
- Many different FSMs (DFAs) for the same problem
- Take a given FSM and try to reduce its state set by combining states
  - Algorithm will always produce the unique minimal equivalent machine (up to renaming of states) but we won't prove this

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***



state minimization example

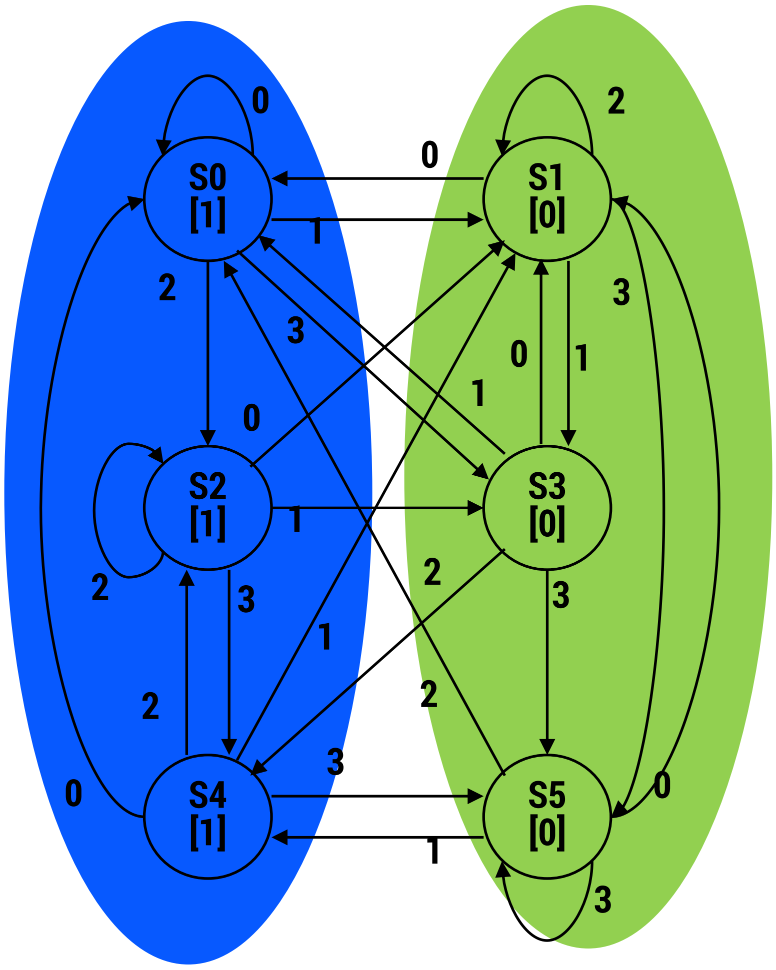


present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state
transition table

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

state minimization example

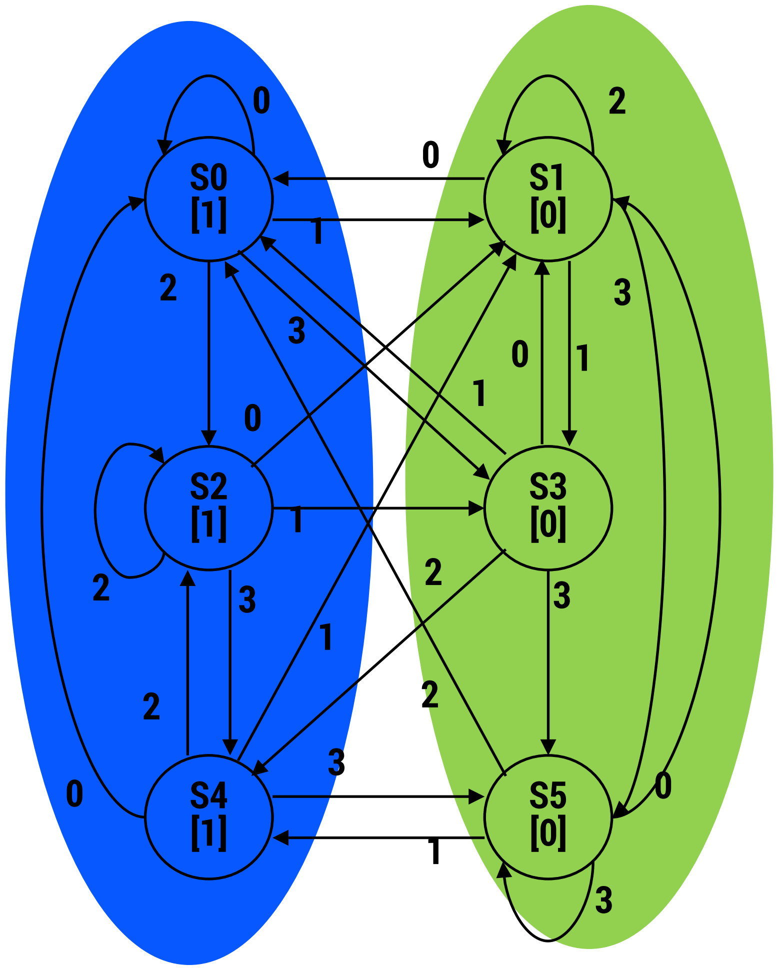


present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state transition table

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

state minimization example



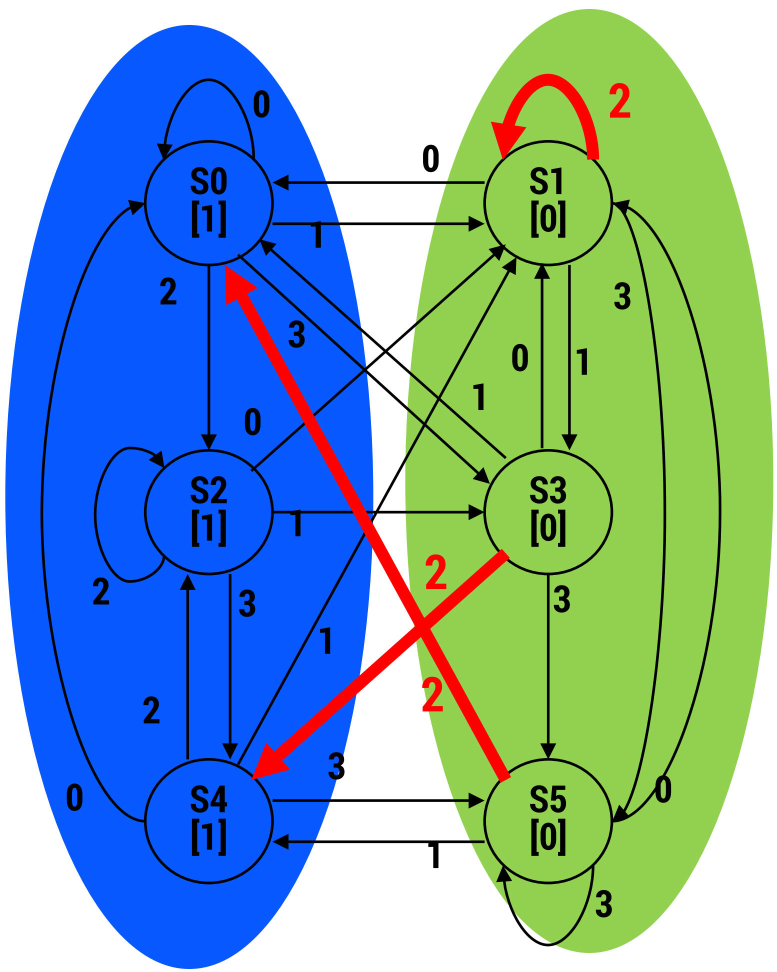
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

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



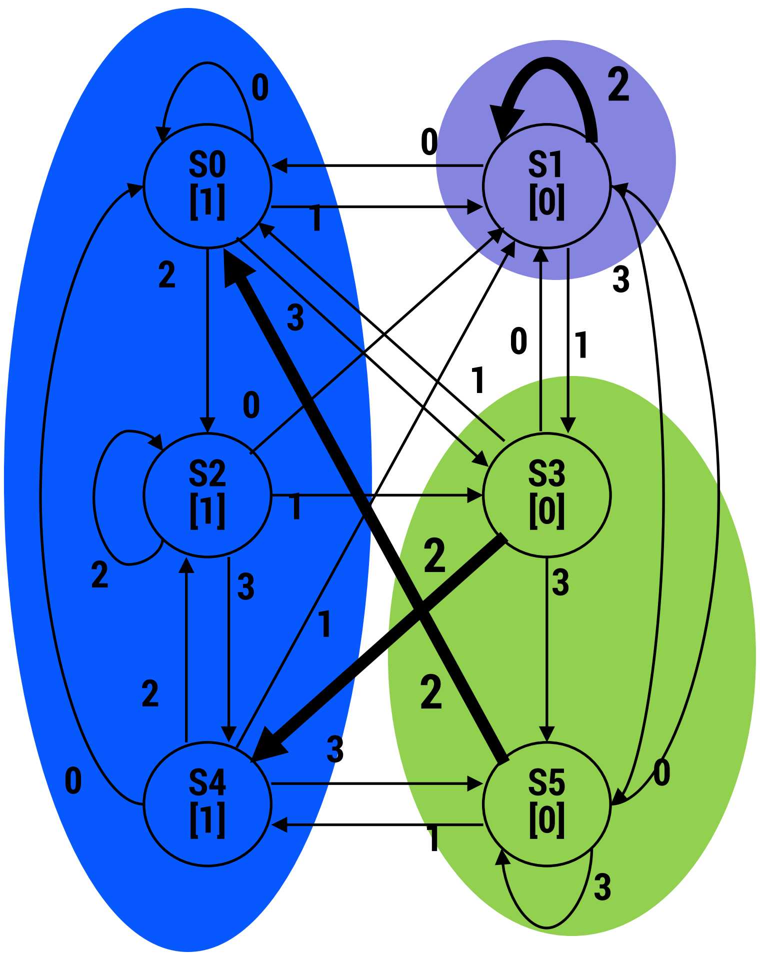
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

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



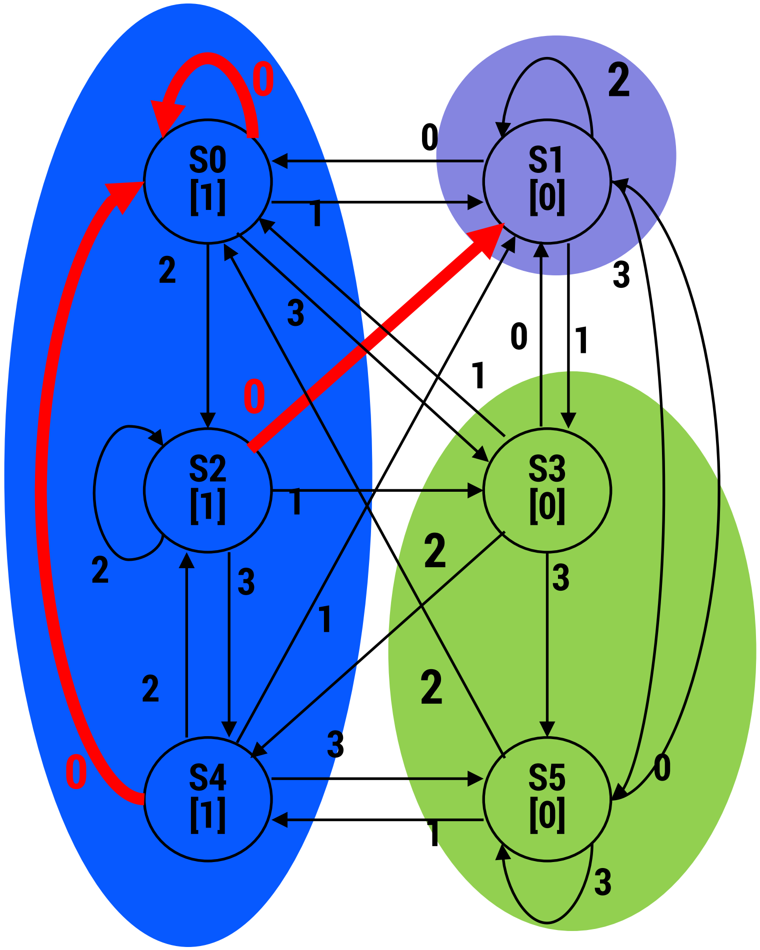
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

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



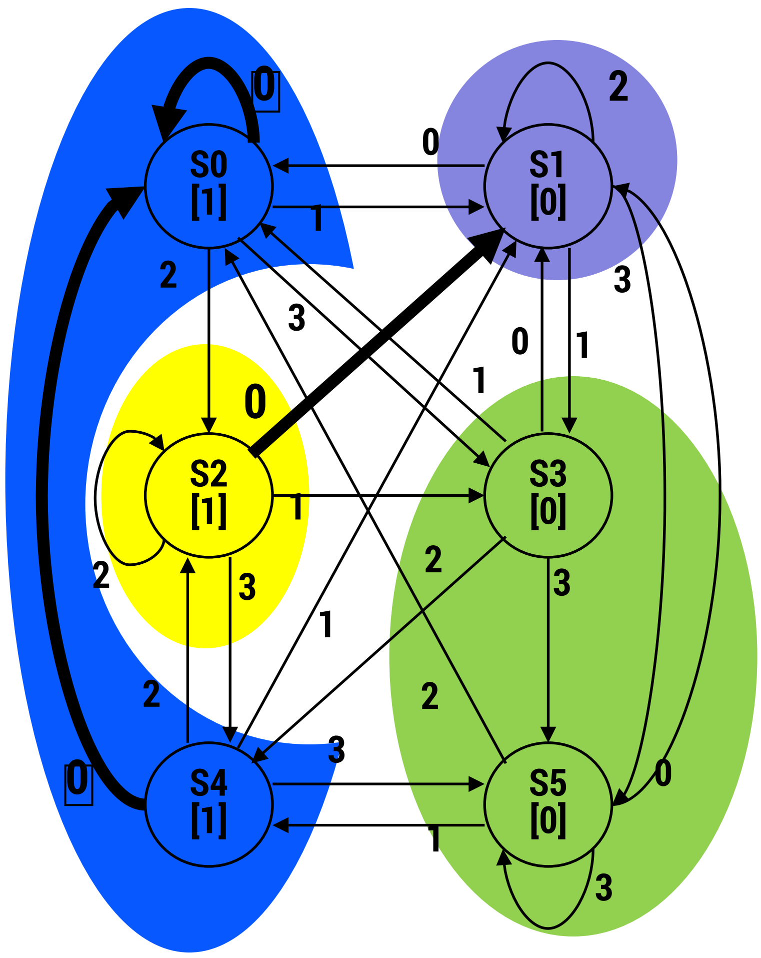
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

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



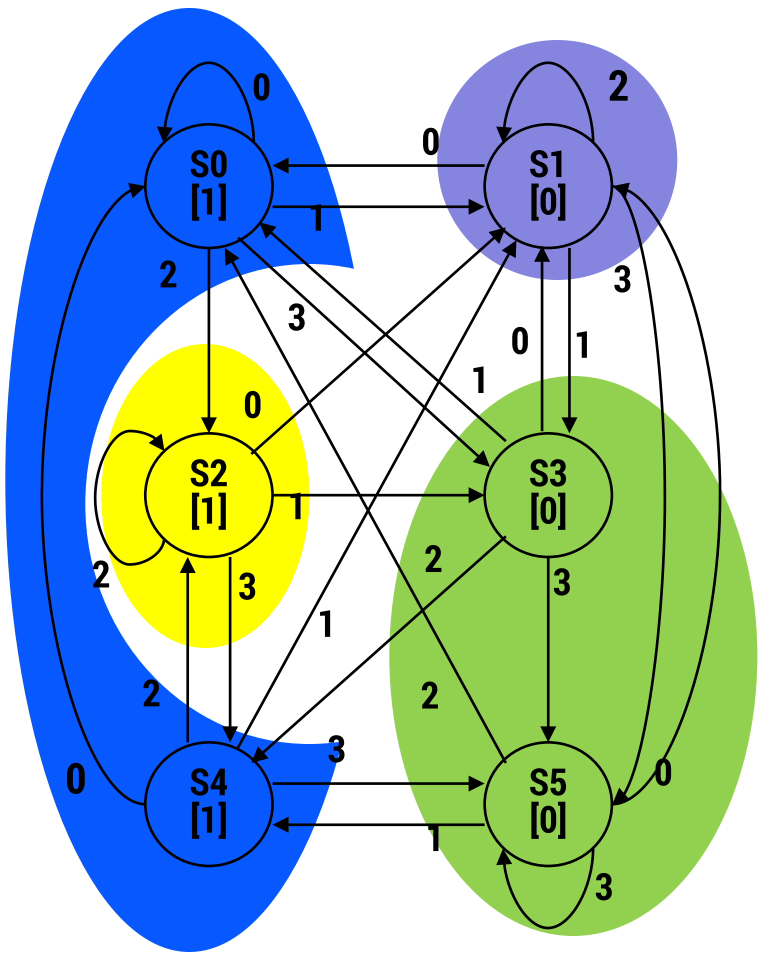
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

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



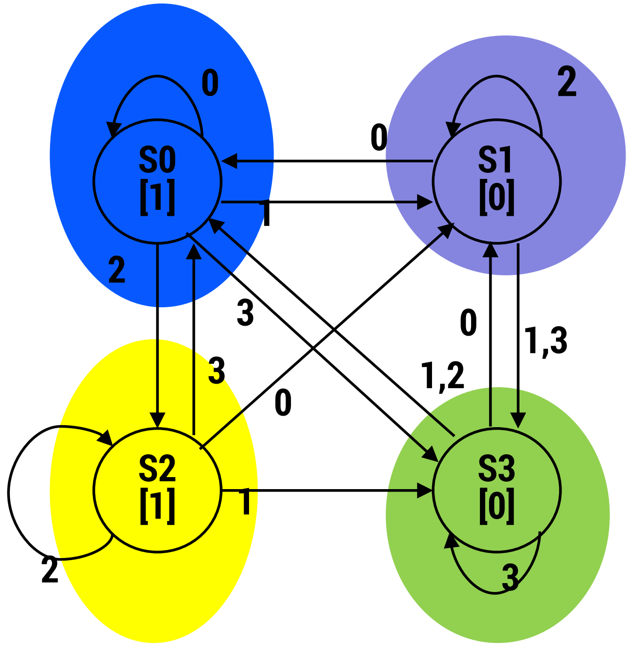
present state	next state 0	next state 1	next state 2	next state 3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S5	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

state
transition table

Can combine states S0-S4 and S3-S5.

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

minimized machine



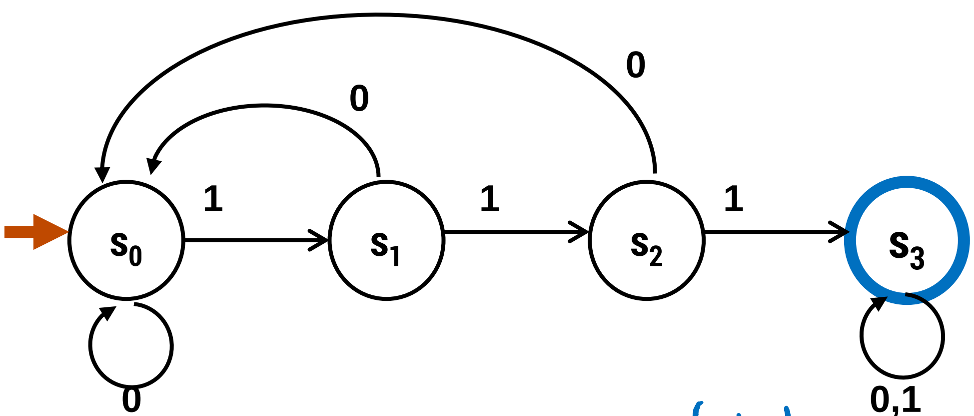
present state	next state 0	1	2	3	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S3	0
S2	S1	S3	S2	S0	1
S3	S1	S0	S0	S3	0

state
transition table

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

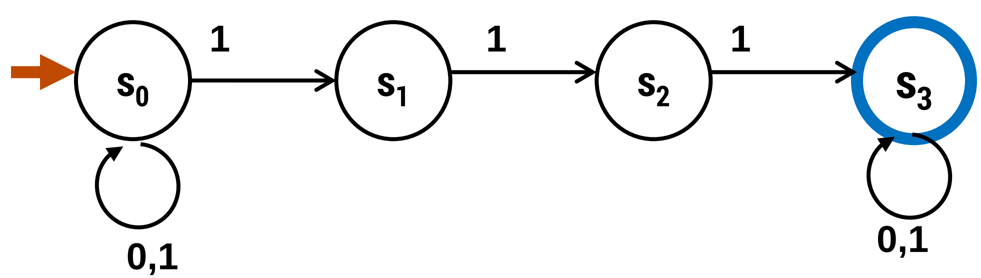


0111

nondeterministic finite automaton (NFA)

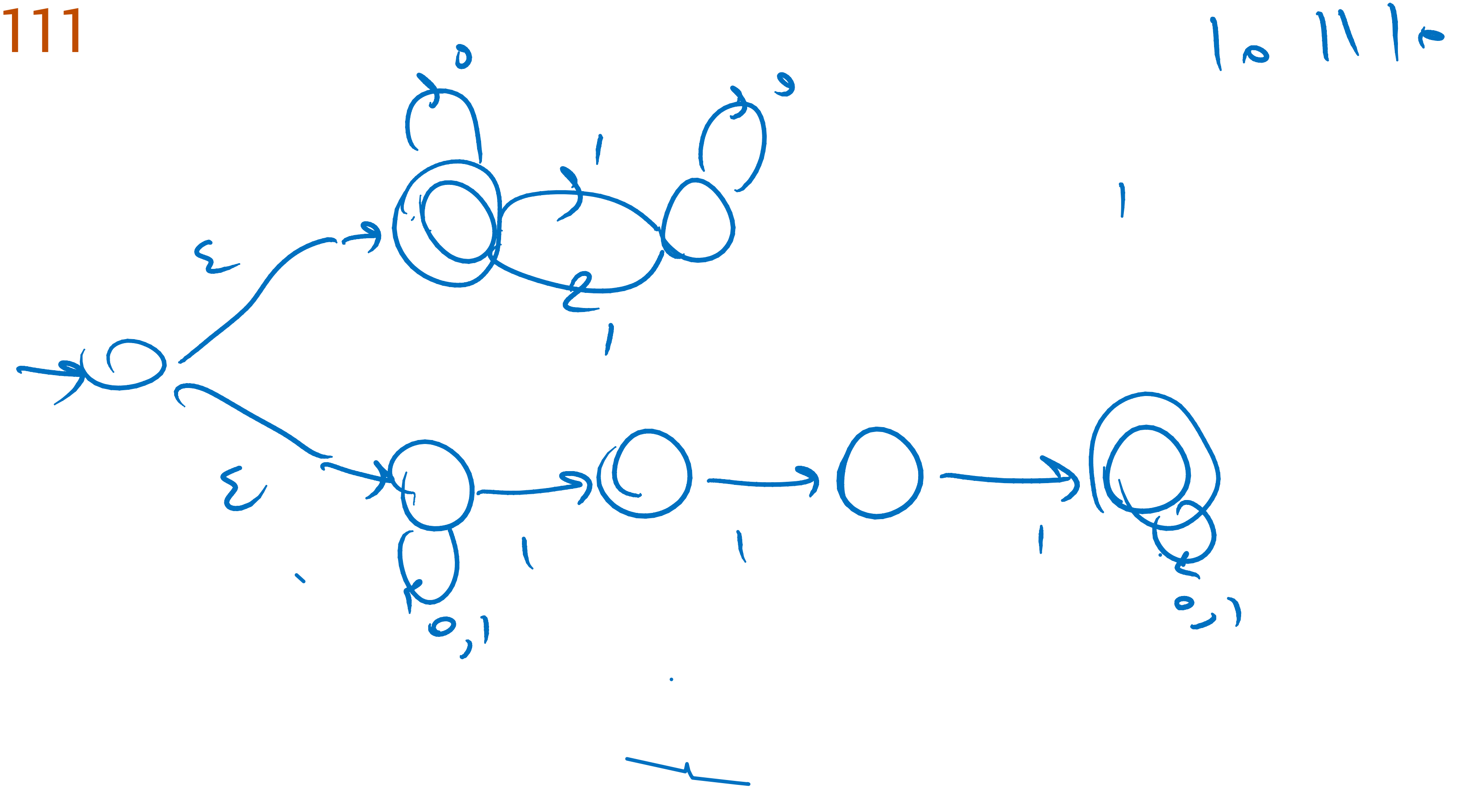
- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol--- can have 0 or >1
  - Also can have edges labeled by empty string ε
- **Definition:** x is in the language recognized by an NFA if and only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

01110 101110



goal: NFA to recognize...

binary strings that have even # of 1's or contain the substring 111



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

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:
  - $\varnothing$, $\boldsymbol{\varepsilon}$ 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:

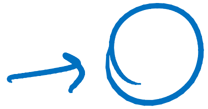
    (A $\cup$ B)

    (AB)

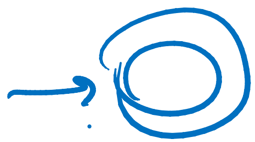
    A*

base case

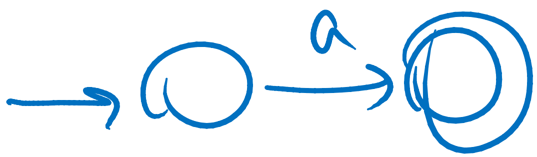
- Case ∅:



- Case **ε**:

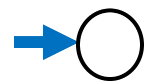


- Case ***a***:

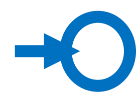


base case

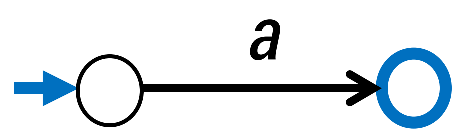
- Case ∅:



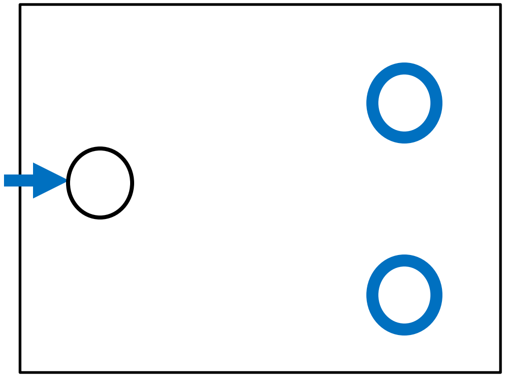
- Case **ε**:



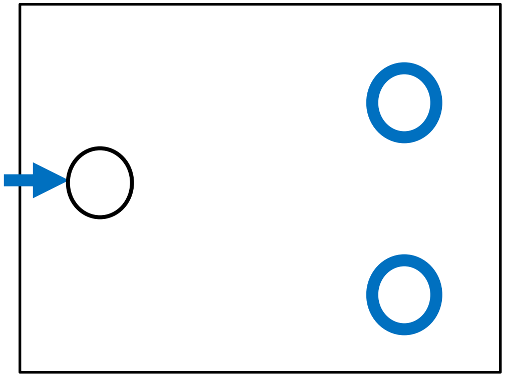
- Case ***a***:



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

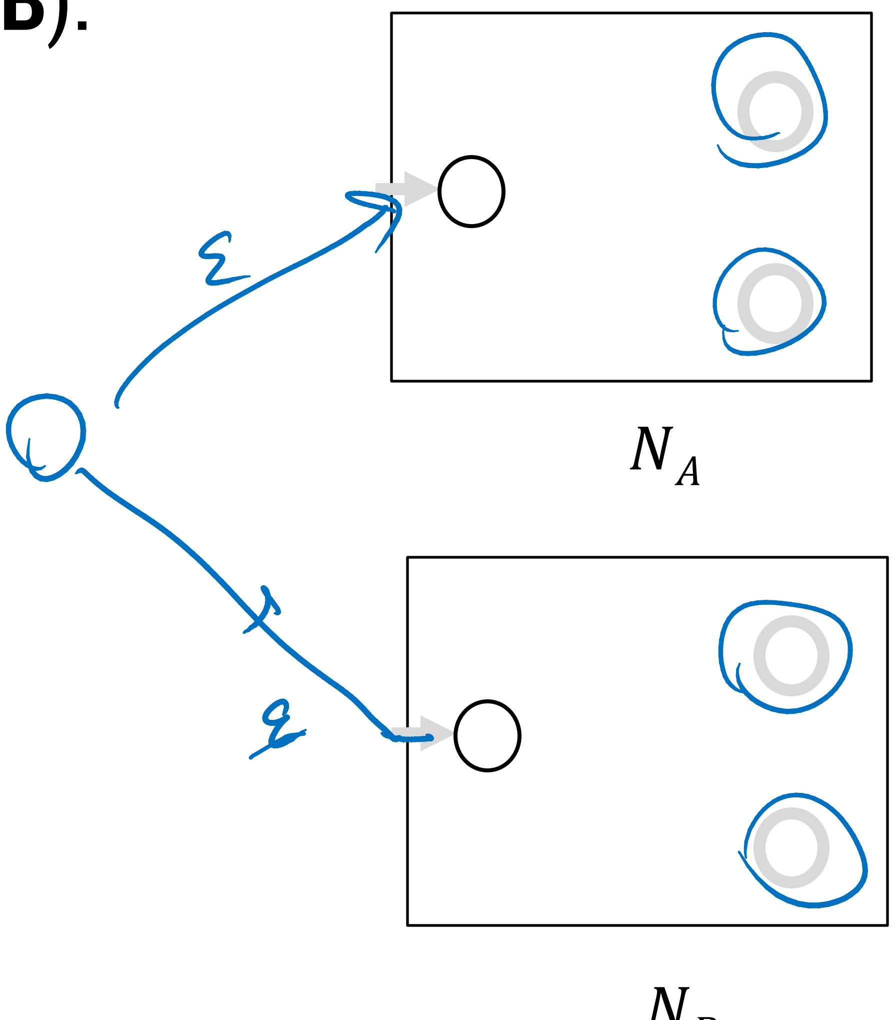


N_A

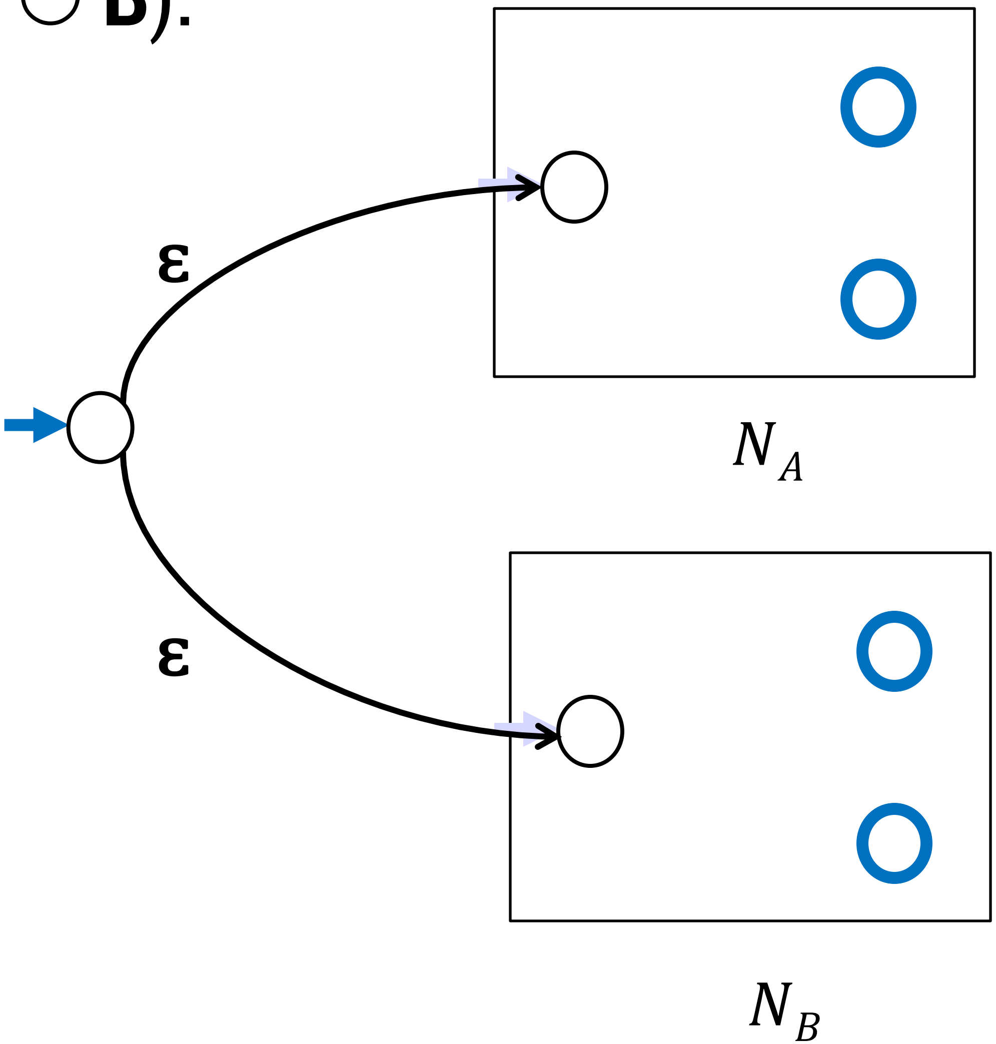


N_B

Case (A ∪ B):

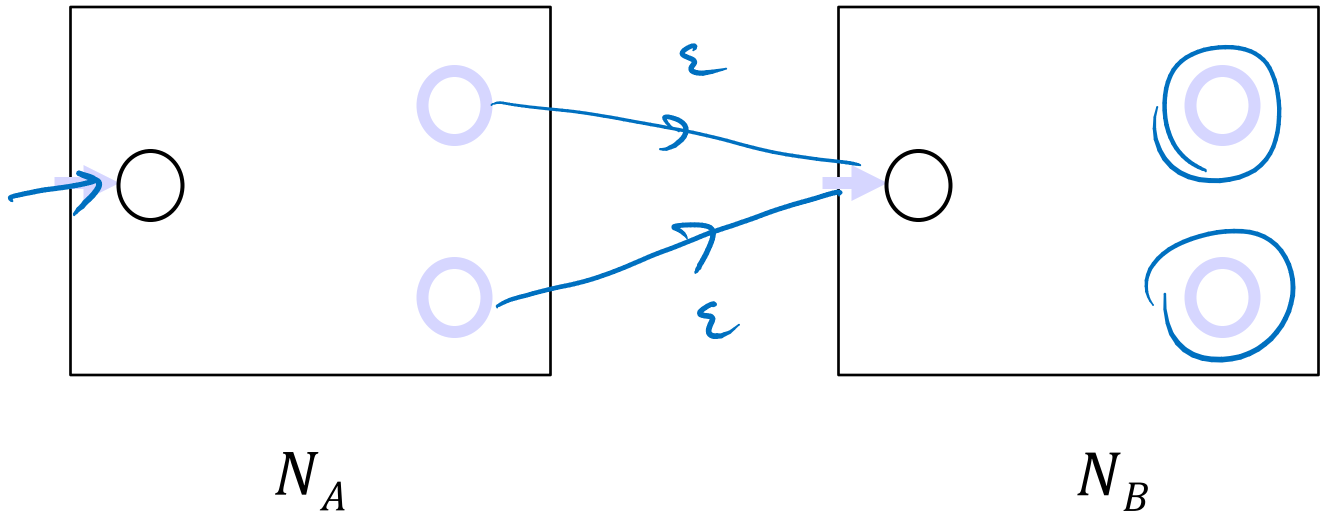


Case ($A \cup B$):

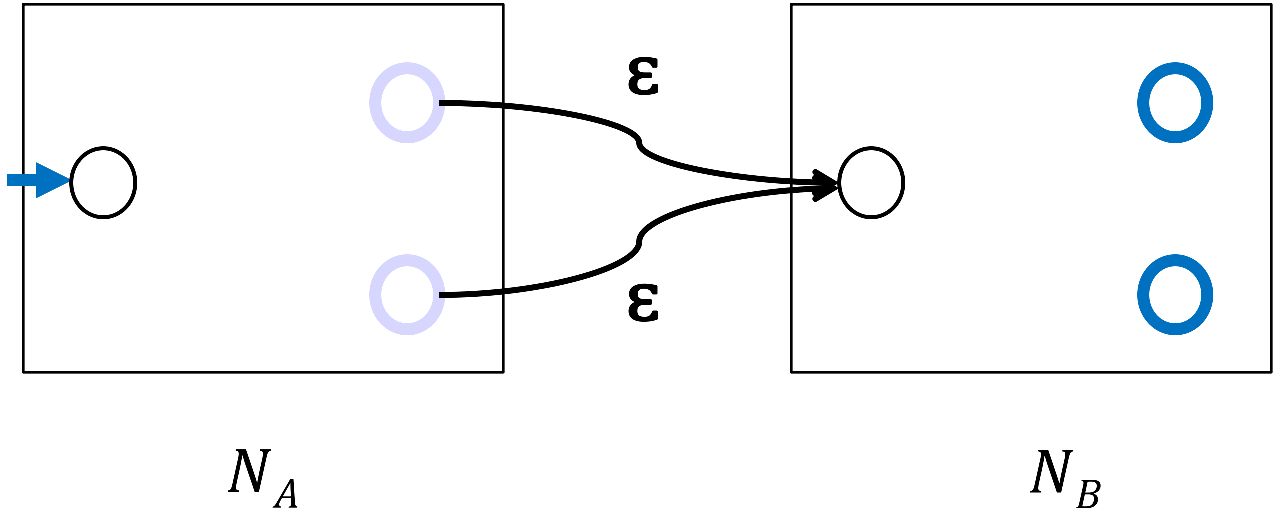


inductive step

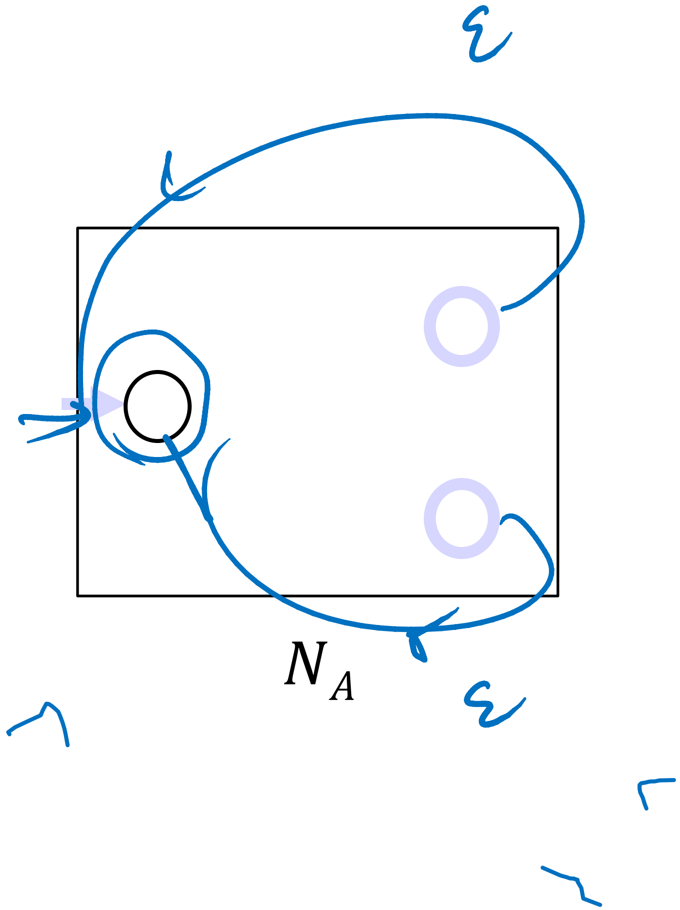
Case (AB):



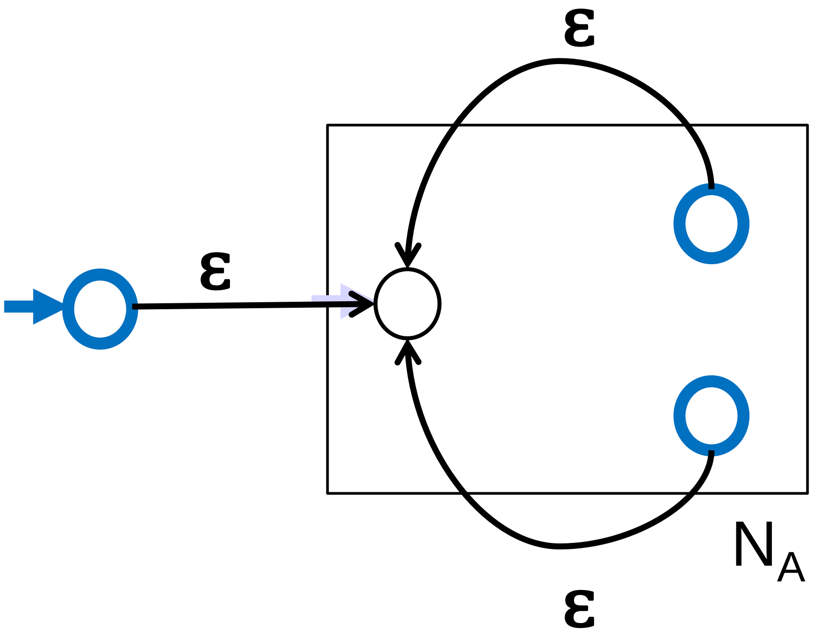
Case (AB):



Case A*



Case A*



ε

A

AA