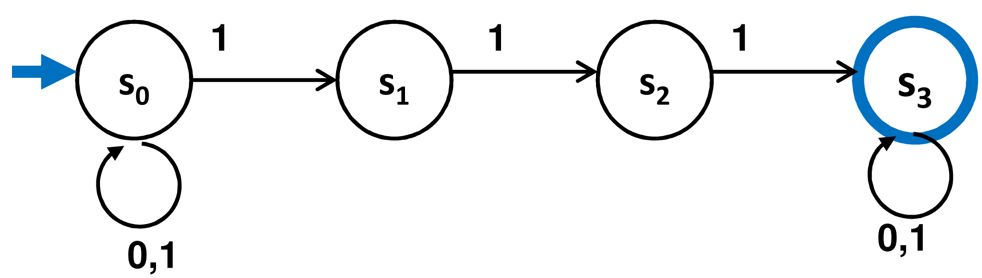
# CSE 311: Foundations of Computing

## Lecture 24: NFAs, Regular expressions, and NFA→DFA

# 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
- **Defn:** 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



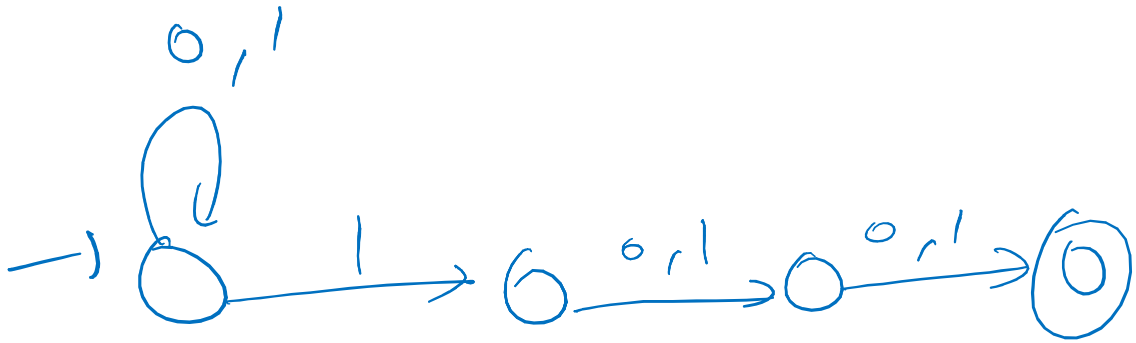
# Three ways of thinking about NFAs

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

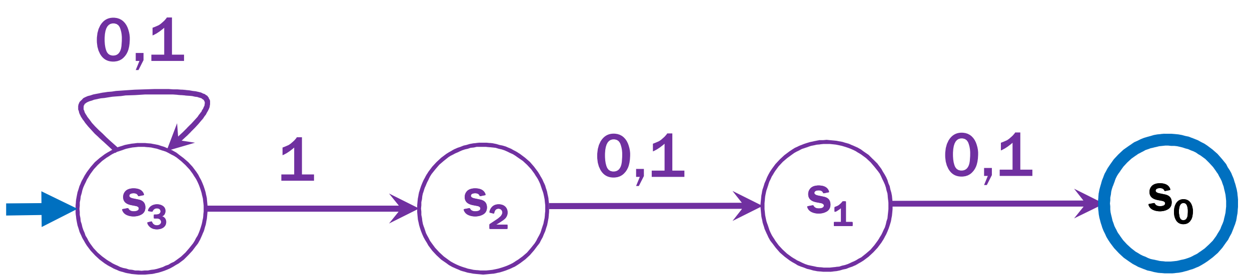
- **Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)**

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

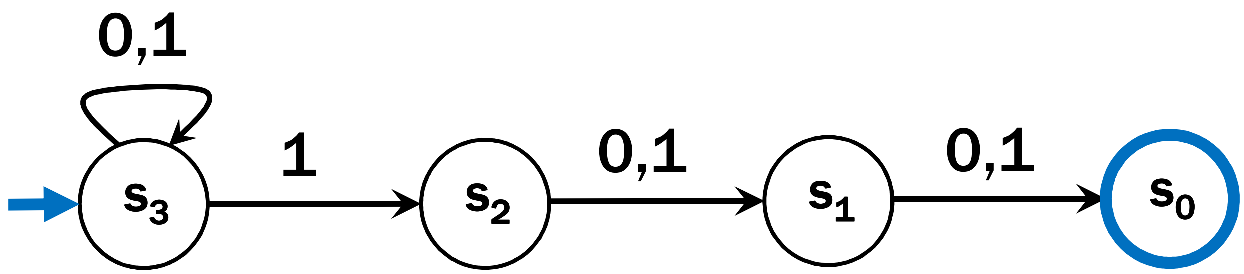
# NFA for set of binary strings with a 1 in the 3rd position from the end



# NFA for set of binary strings with a 1 in the 3$^{rd}$ position from the end

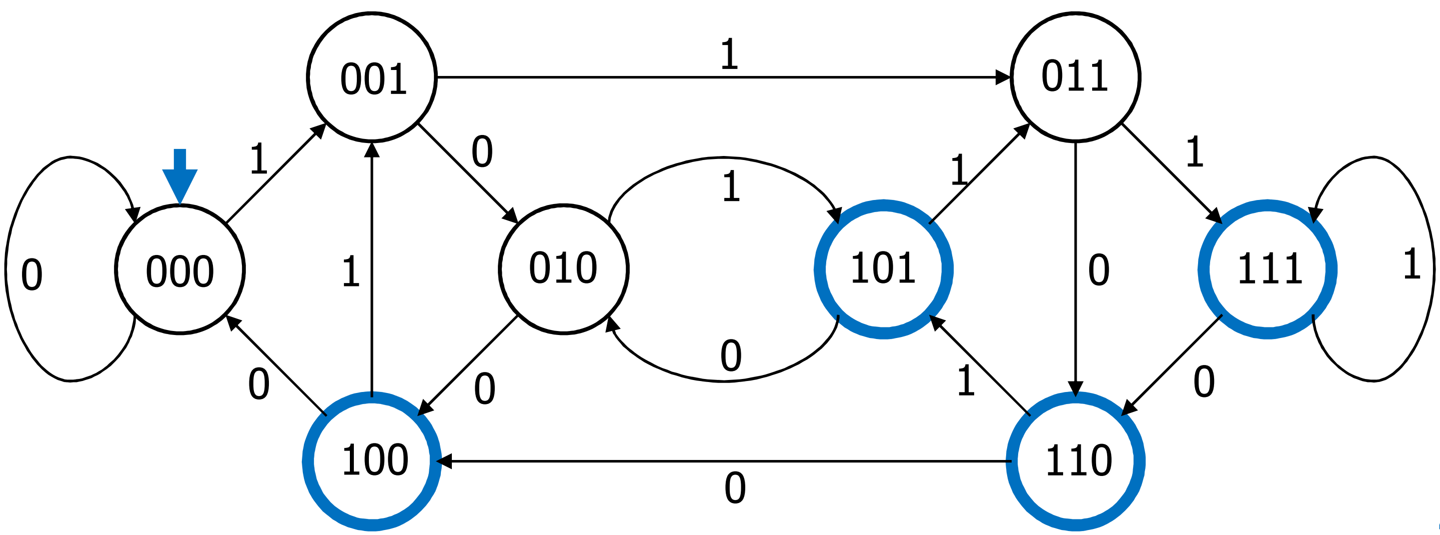


# Compare with the smallest DFA



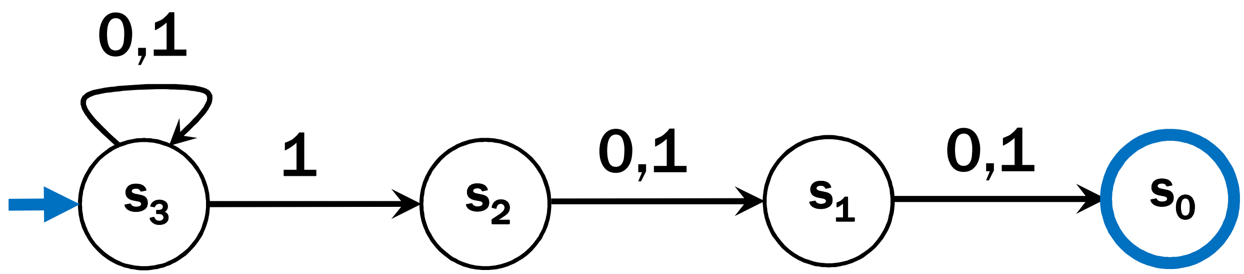

$n^{th}$ from end

$n+1$

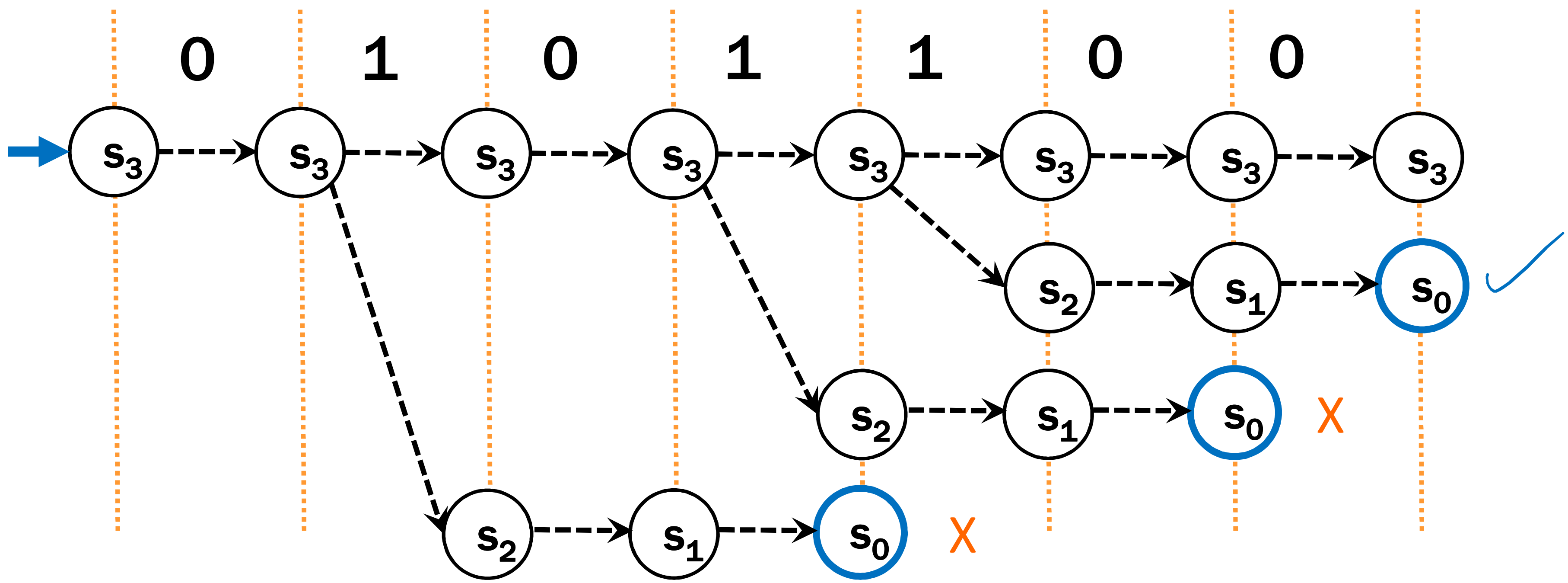



$2^n$

# 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...

$P(A)$ be "There is an NFA $N_A$ that recognizes the language represented by reg exp. $A$"

$\forall$ regular expression $A$ $P(A)$ is true

# Regular Expressions over $\Sigma$

- **Basis:**
  - $\varnothing$, $\varepsilon$ 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:

    **$(A \cup B)$**

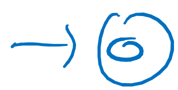
    **$(AB)$**

    **$A^*$**

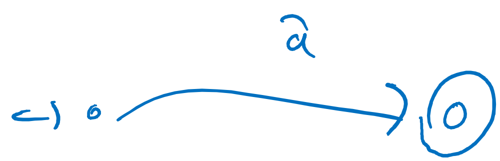
# Base Case

- **Case ∅:**

- **Case ε:**

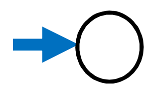


- **Case *a*:**

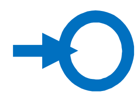


# Base Case

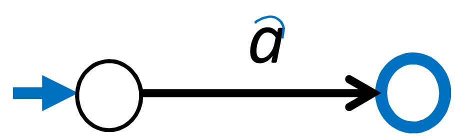
- **Case ∅:**



- **Case ε:**

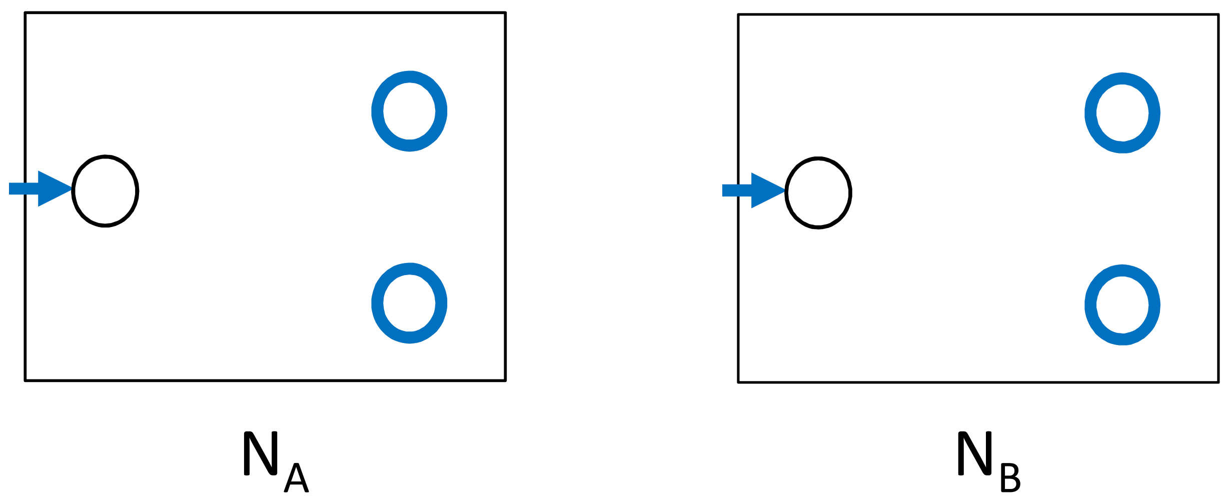


- **Case *a*:**



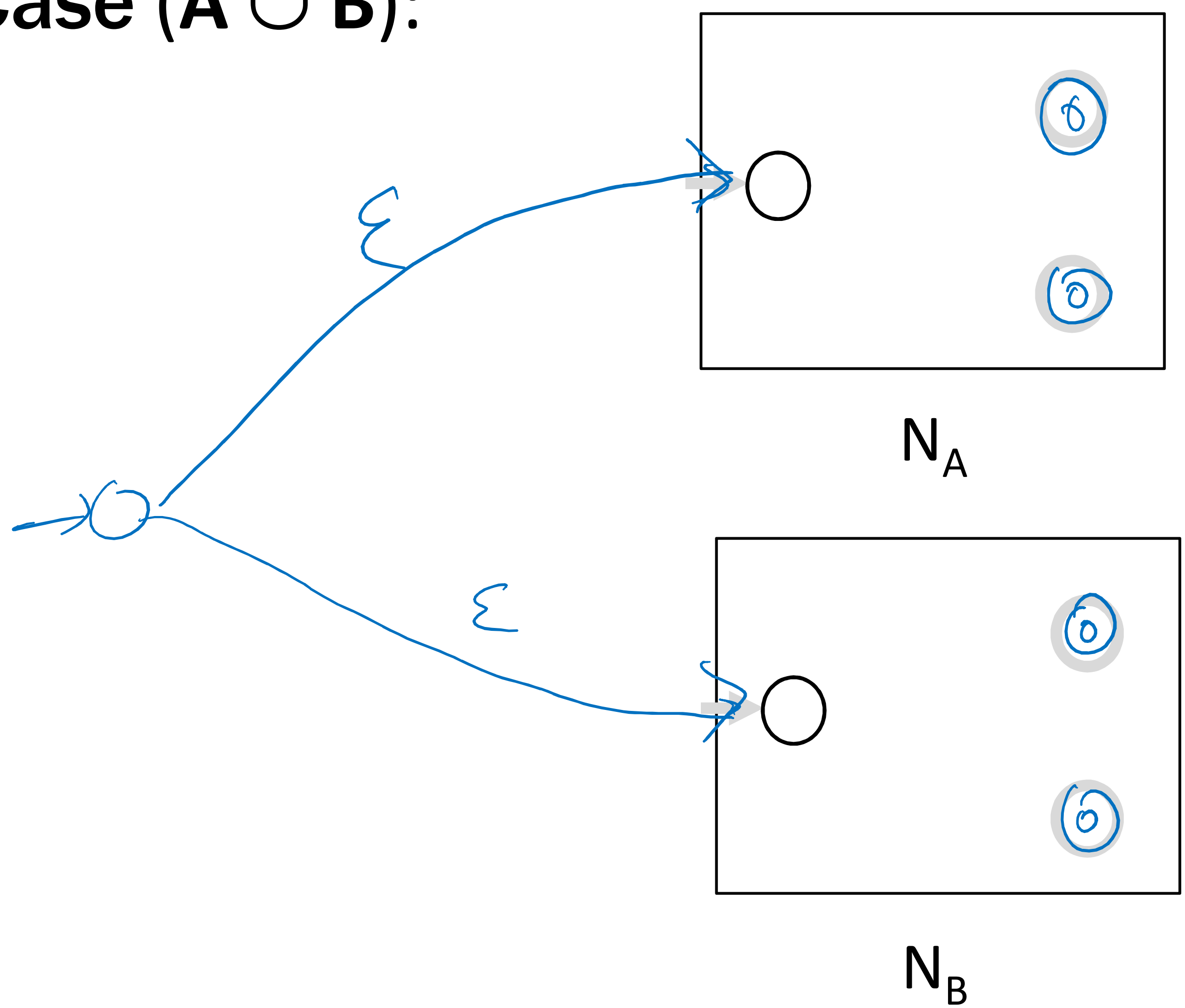
## Inductive Hypothesis

- 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



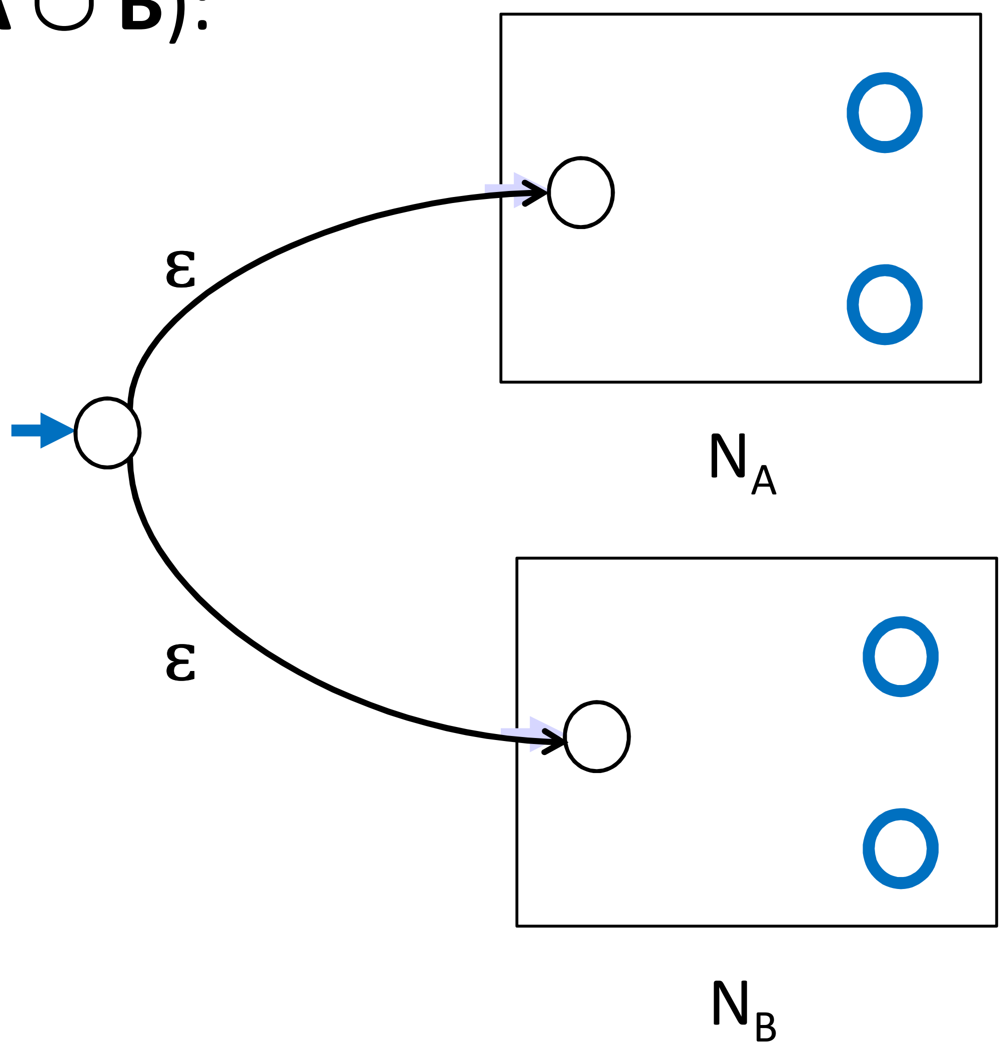
# Inductive Step

**Case ($A \cup B$):**



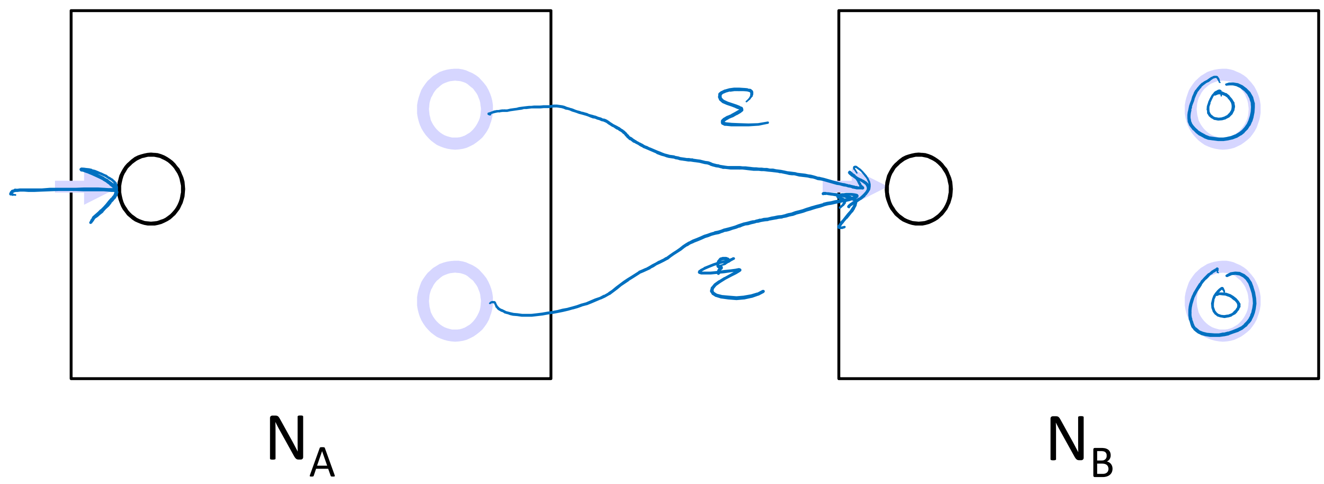
# Inductive Step

**Case ($A \cup B$):**



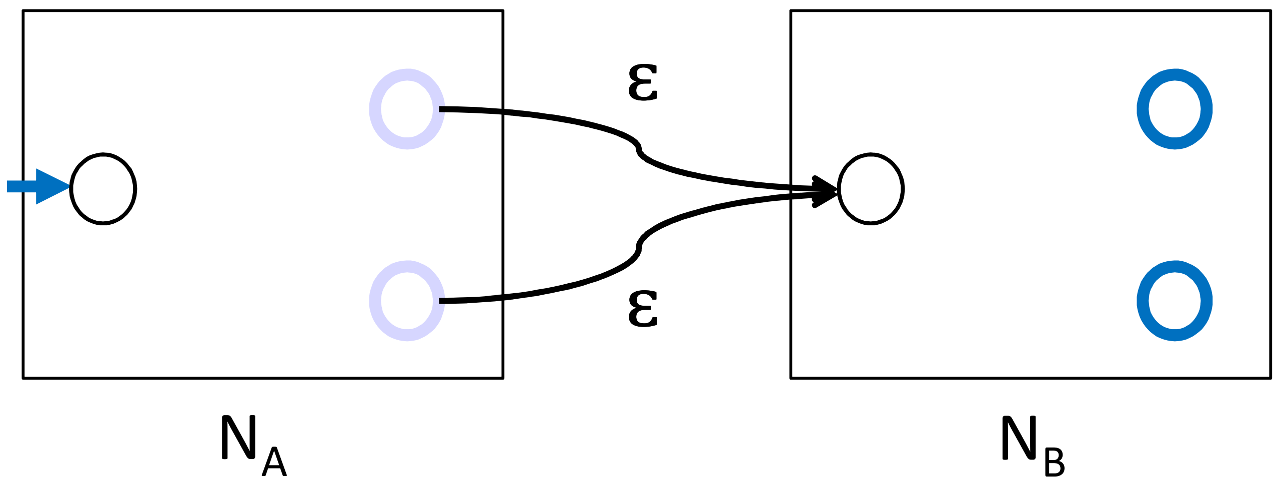
# Inductive Step

## Case (AB):



# Inductive Step

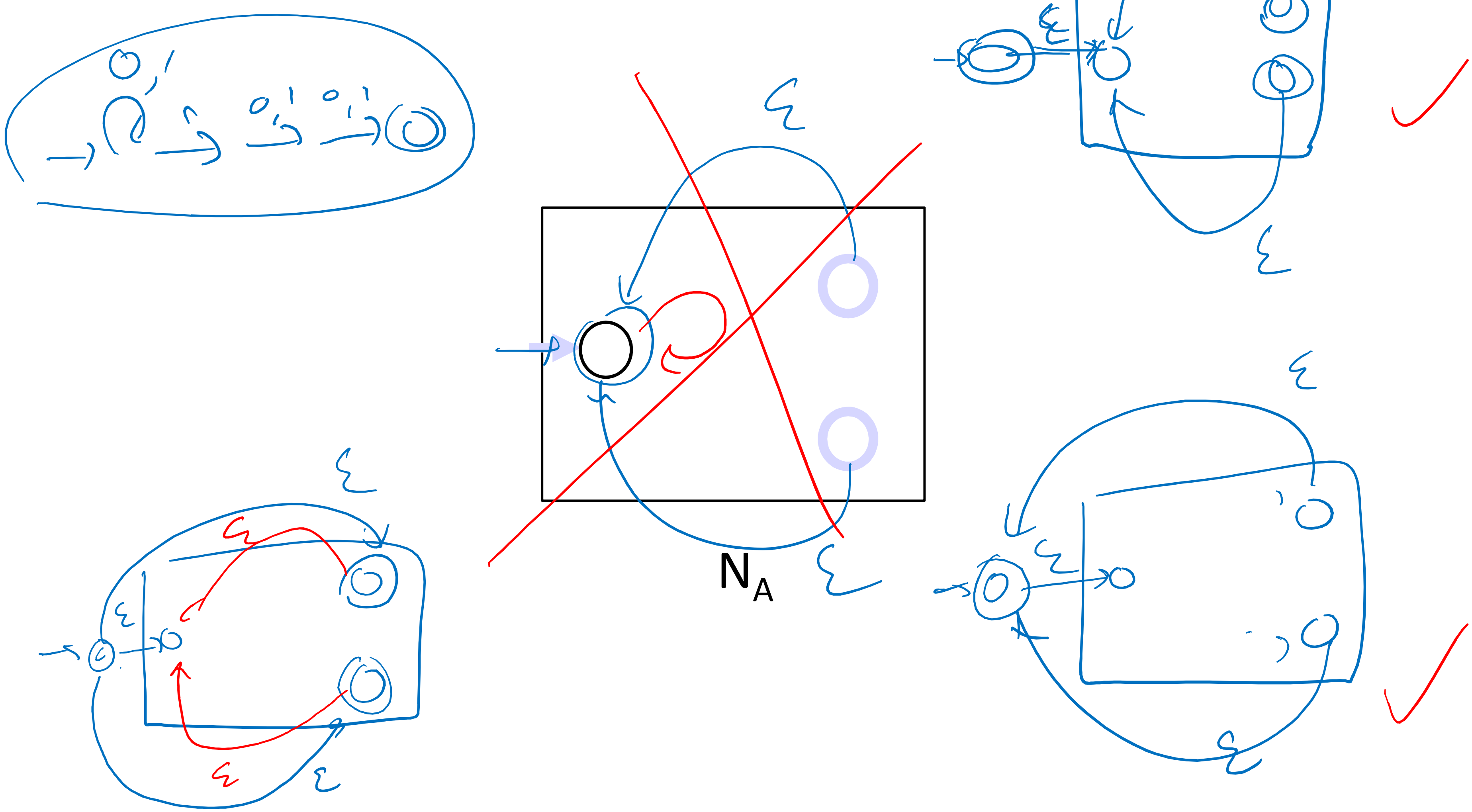
## Case (AB):



# Inductive Step

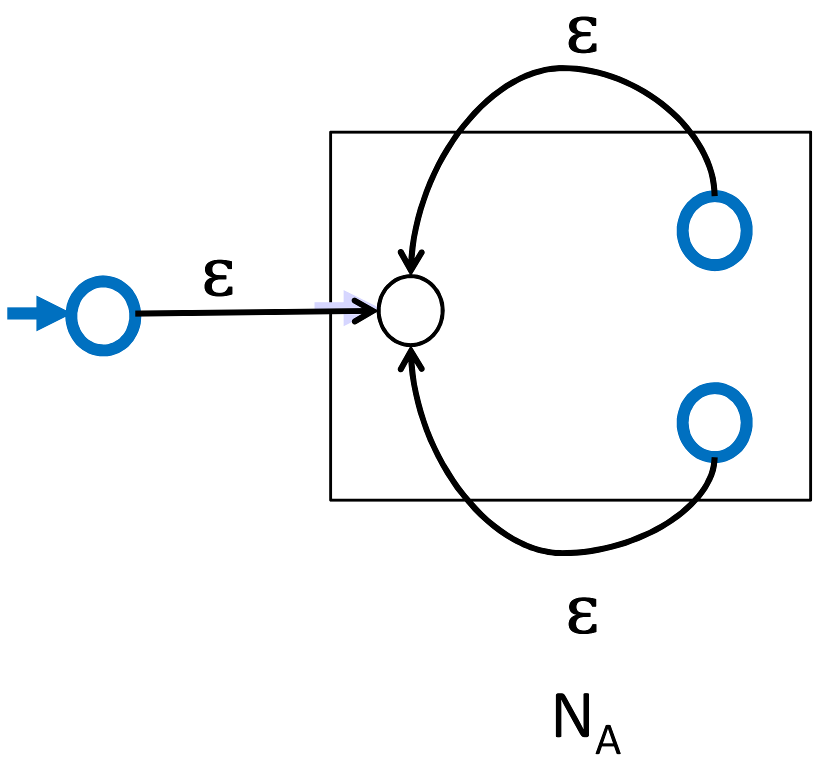
## Case A*

$\phi^* = \varepsilon$

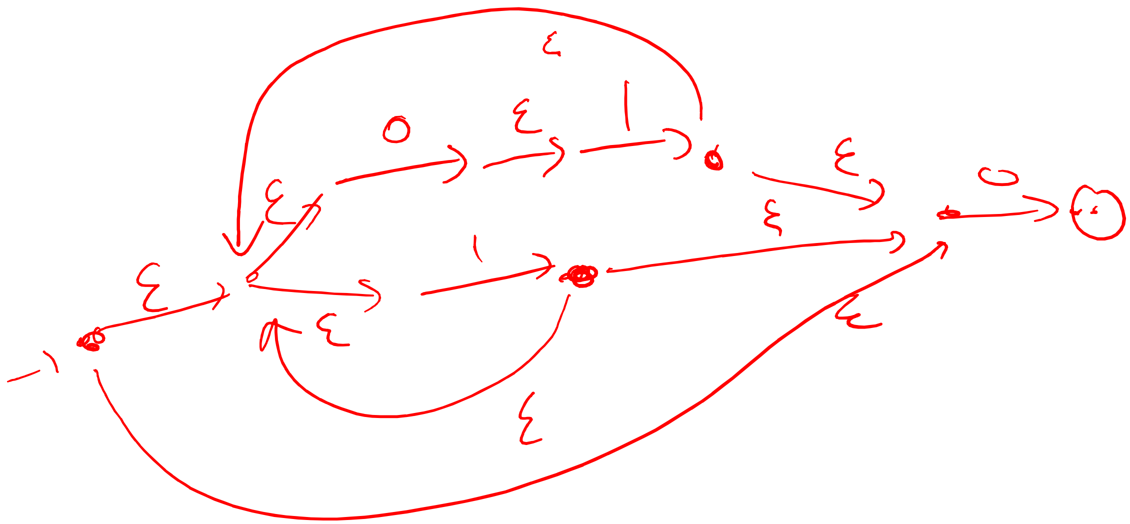


# Inductive Step

## Case A*

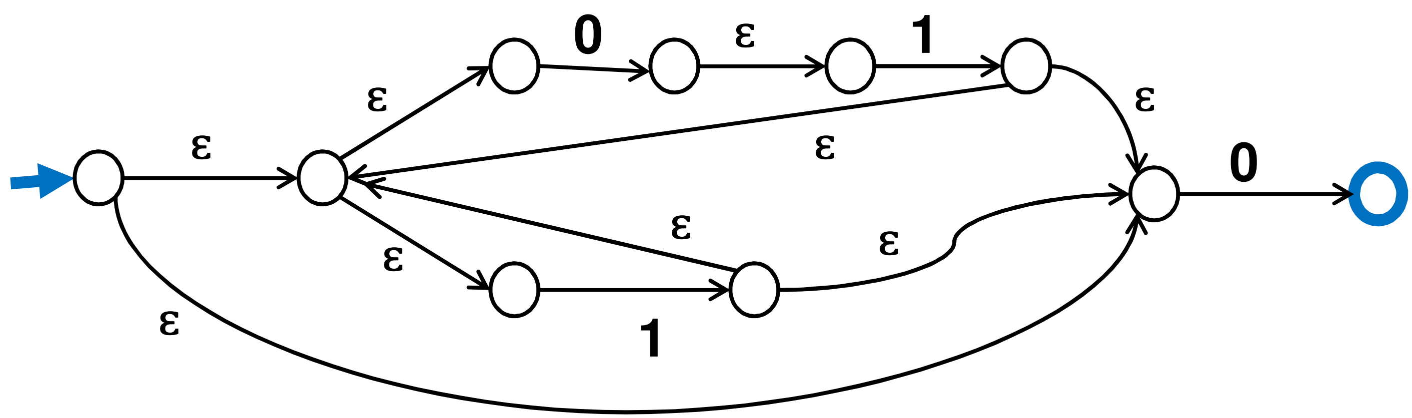


# Build an NFA for (01 ∪1)*0




# Solution

**(01 ∪1)*0**



# NFAs and DFAs

Every DFA is an NFA

– DFAs have requirements that NFAs don't have

Can NFAs recognize more languages?

# NFAs and DFAs

Every DFA is an NFA

– DFAs have requirements that NFAs don't have

Can NFAs recognize more languages? No!

**Theorem: For every NFA there is a DFA that recognizes exactly the same language**

# Three ways of thinking about NFAs

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

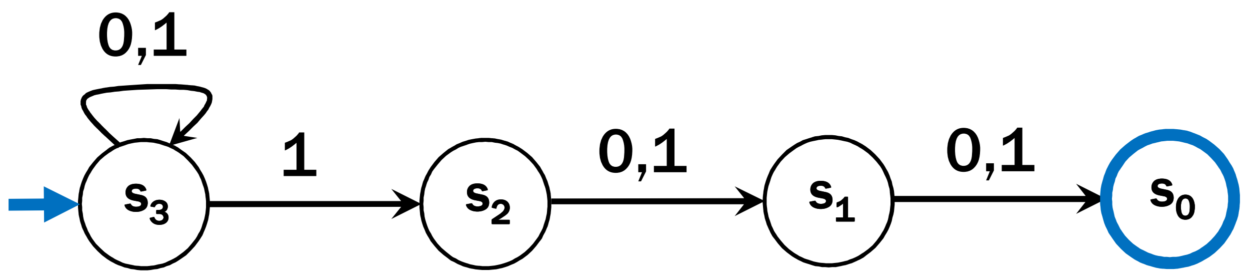
- **Perfect guesser: The NFA has input** x **and whenever there is a choice of what to do it magically guesses a good one (if one exists)**

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

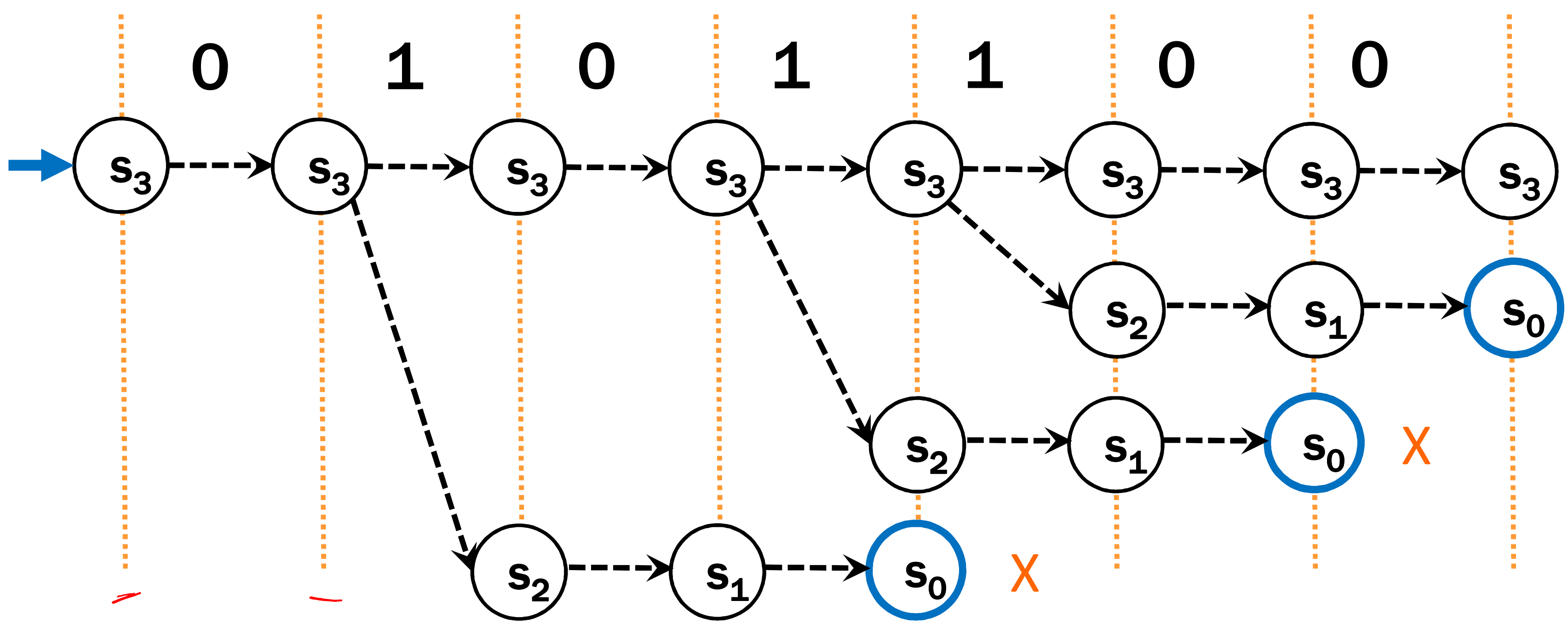
## Conversion of NFAs to a DFAs

- Proof Idea:
  - The DFA keeps track of ALL the states that the part of the input string read so far can reach in the NFA
  - There will be one state in the DFA for each *subset* of states of the NFA that can be reached by some string

# Parallel Exploration view of an NFA



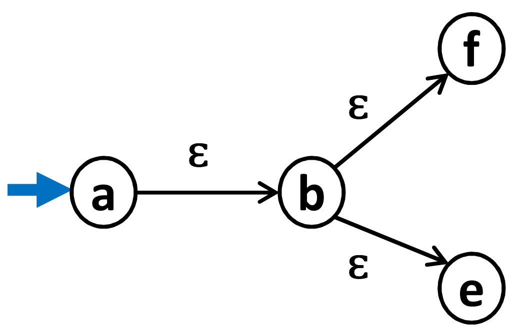

Input string 0101100



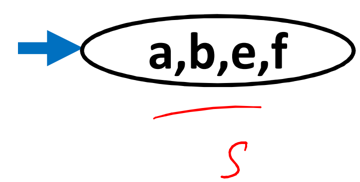
# Conversion of NFAs to a DFAs

## New start state for DFA

– The set of all states reachable from the start state of the NFA using only edges labeled ε




NFA

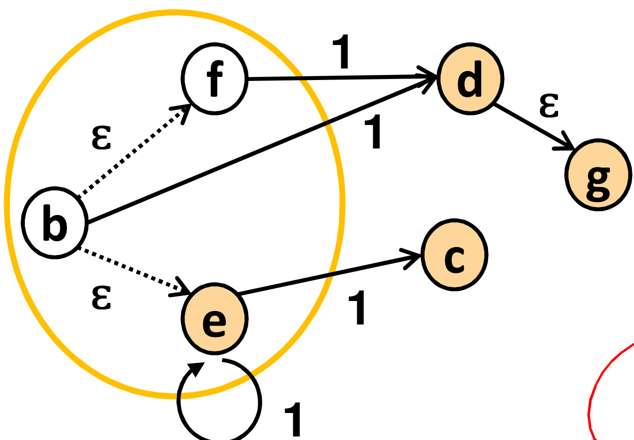


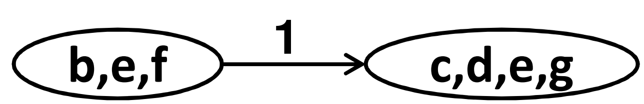

DFA

# Conversion of NFAs to a DFAs

For each state of the DFA corresponding to a set S of states of the NFA and each symbol s

- Add an edge labeled s to state corresponding to T, the set of states of the NFA reached by
  - starting from some state in S, then
  - following one edge labeled by s, and
    then following some number of edges labeled by ε
- T will be ∅ if no edges from S labeled s exist



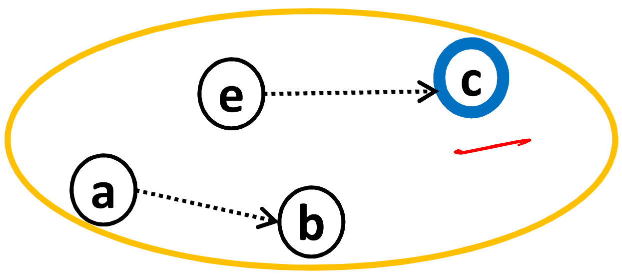




$6 \; 2^{0,1}$

# Conversion of NFAs to a DFAs

## Final states for the DFA

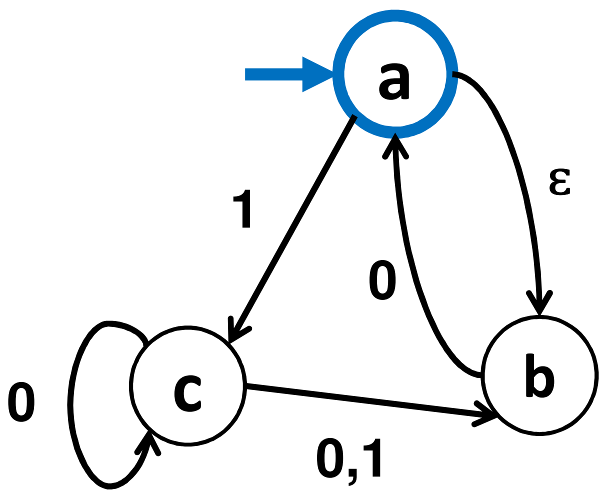
– All states whose set contain some final state of the NFA



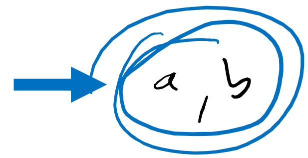

NFA


DFA

# Example: NFA to DFA

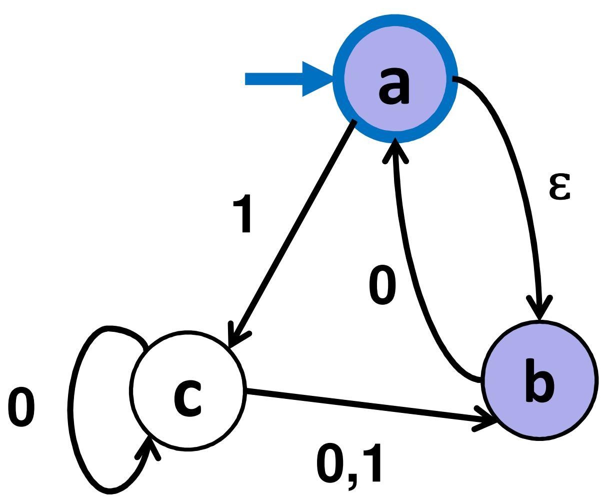



NFA

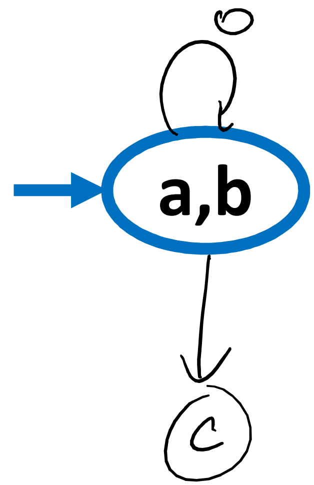



DFA

# Example: NFA to DFA

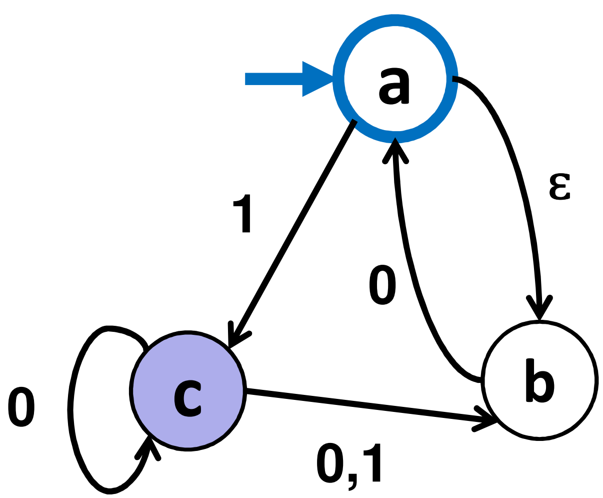



NFA

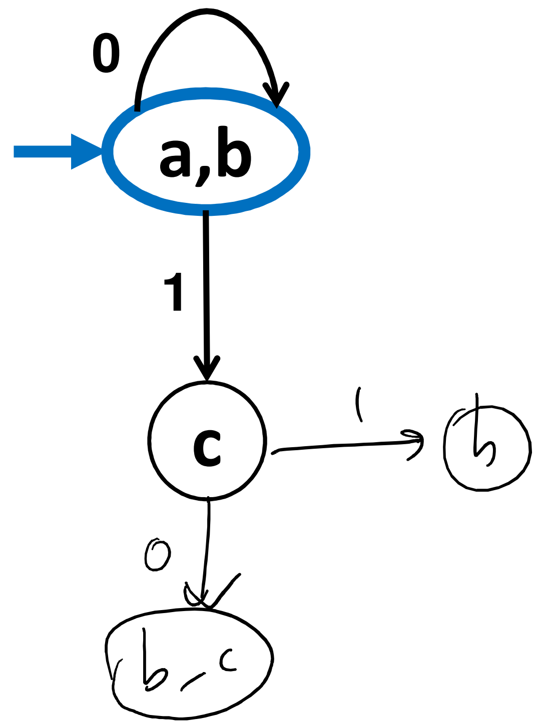



DFA

# Example: NFA to DFA

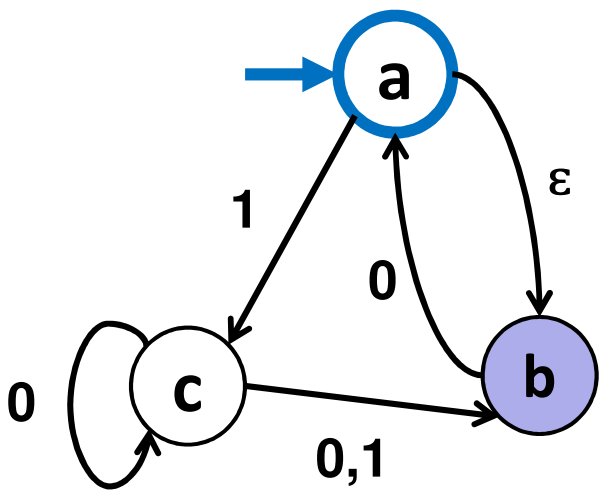



NFA

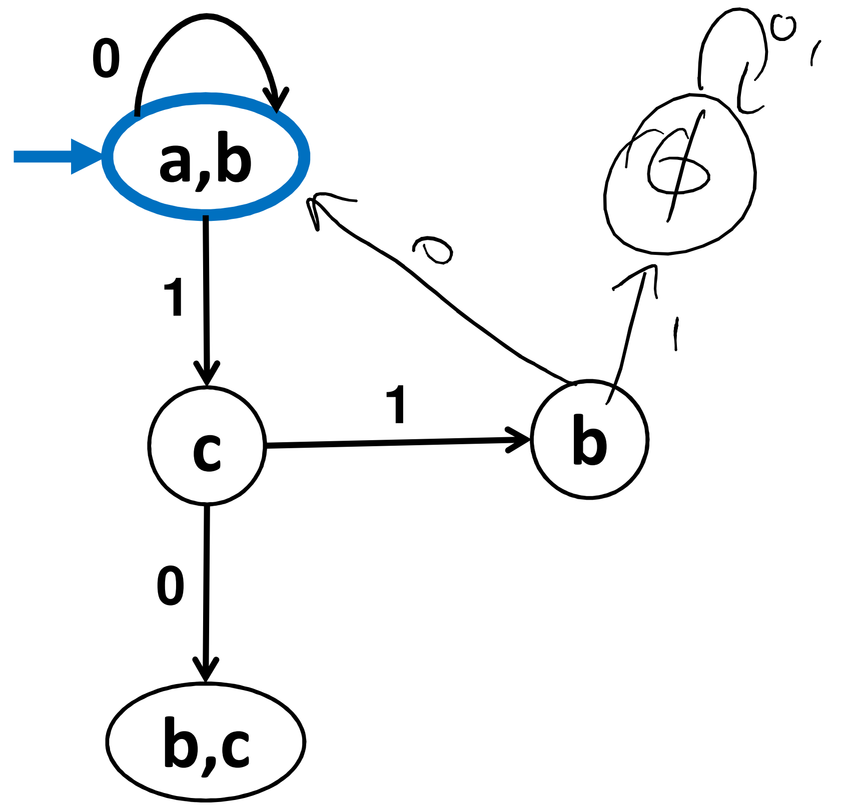



DFA

# Example: NFA to DFA

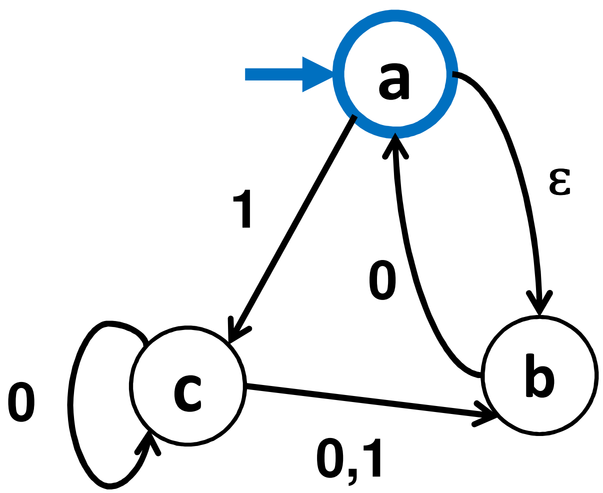



NFA

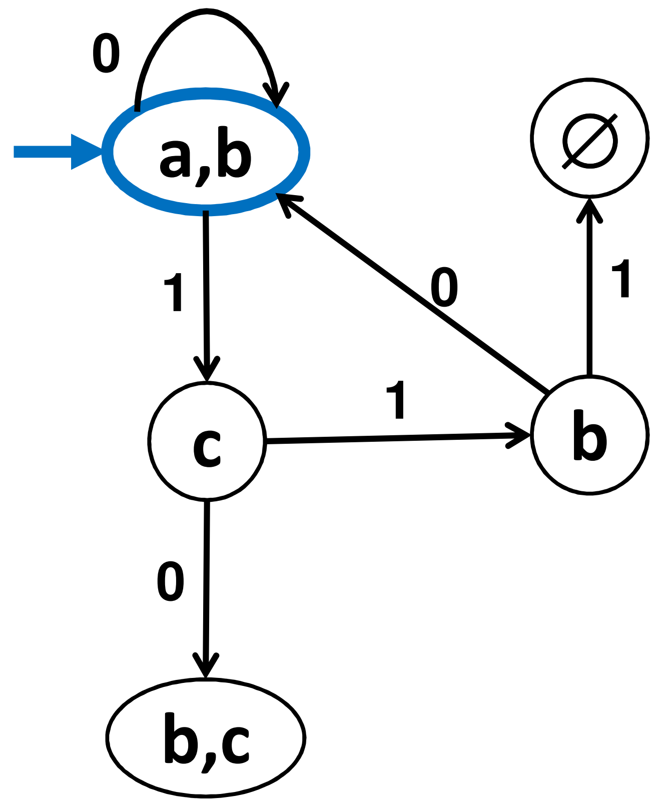



DFA

# Example: NFA to DFA

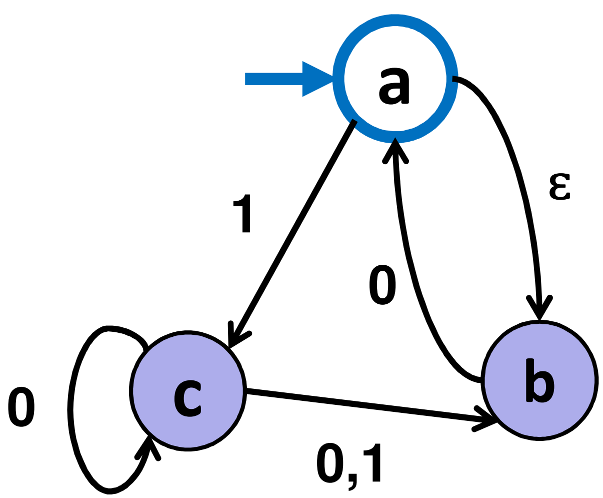



NFA

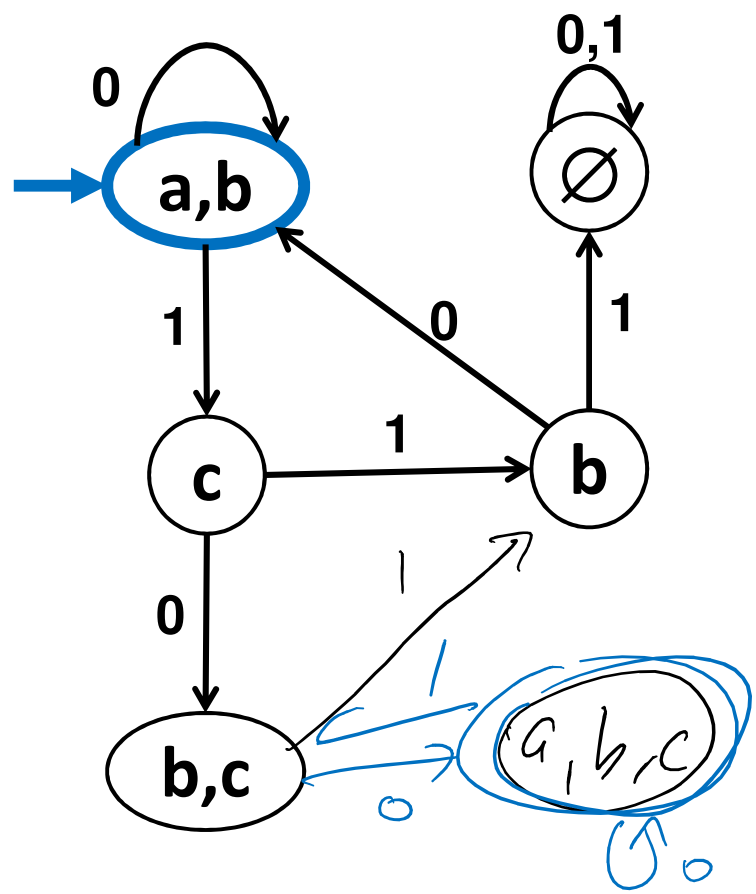



DFA

# Example: NFA to DFA

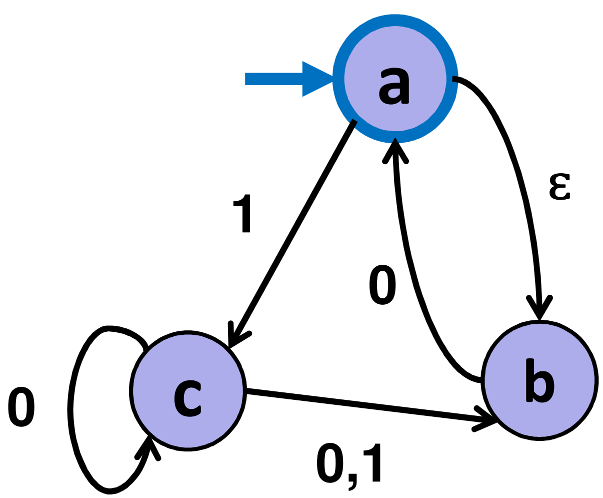



NFA

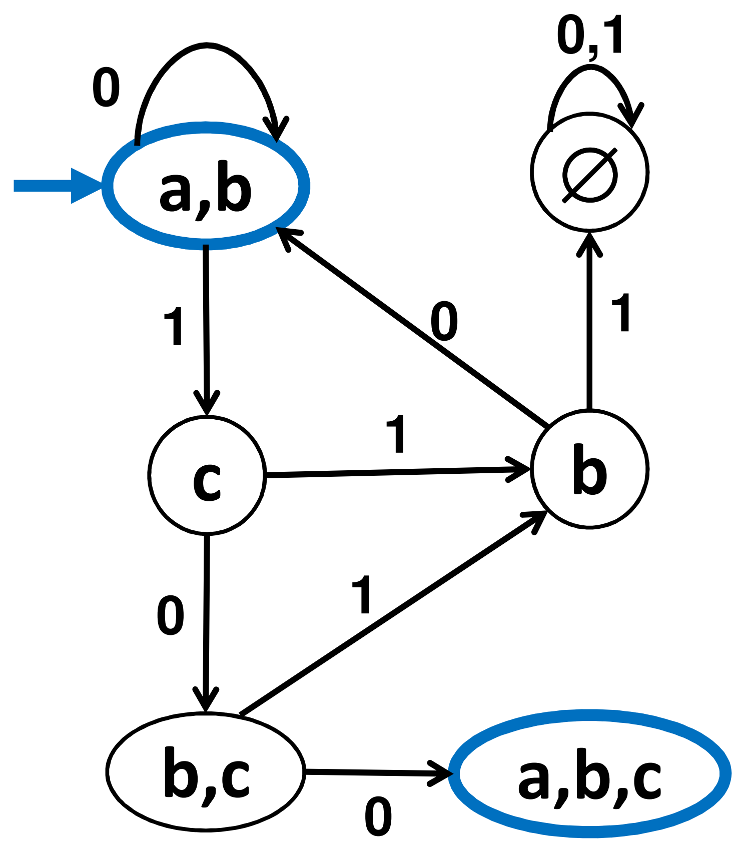



DFA

# Example: NFA to DFA

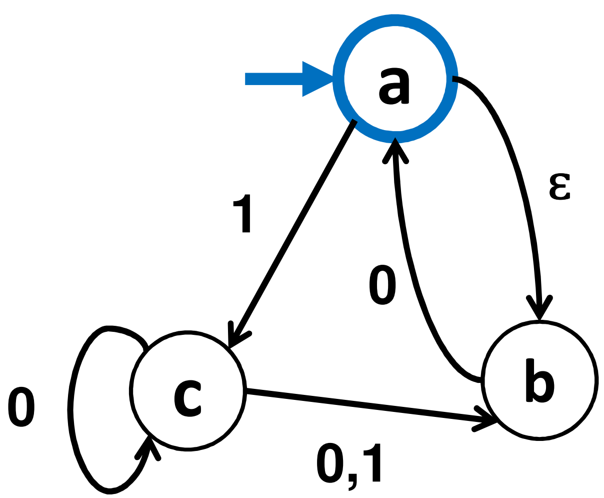



NFA

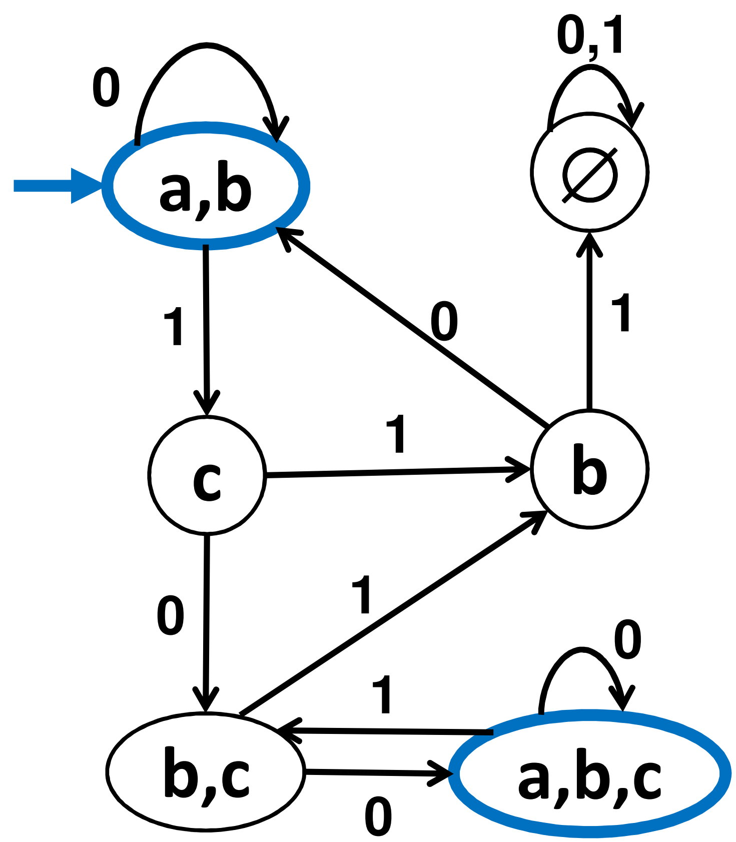



DFA

# Example: NFA to DFA




NFA




DFA

# Exponential Blow-up in Simulating Nondeterminism

- **In general the DFA might need a state for every subset of states of the NFA**
  - Power set of the set of states of the NFA
  - $\boldsymbol{n}$-state NFA yields DFA with at most $\mathbf{2}^{\boldsymbol{n}}$ states
  - We saw an example where roughly $\mathbf{2}^{\boldsymbol{n}}$ is necessary

    "Is the $\boldsymbol{n}^{\text{th}}$ char from the end a 1?"

- **The famous "P=NP?" question asks whether a similar blow-up is always necessary to get rid of nondeterminism for polynomial-time algorithms**