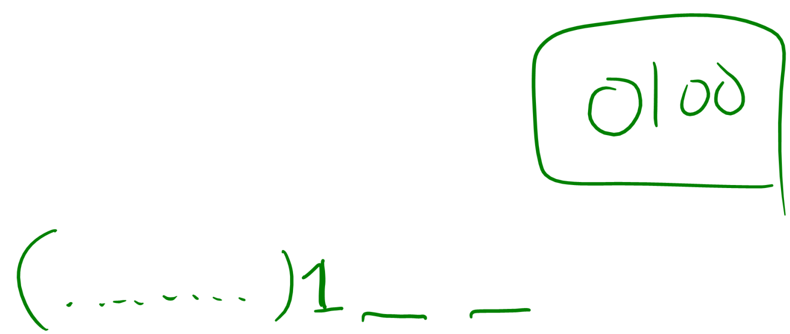
CSE 311

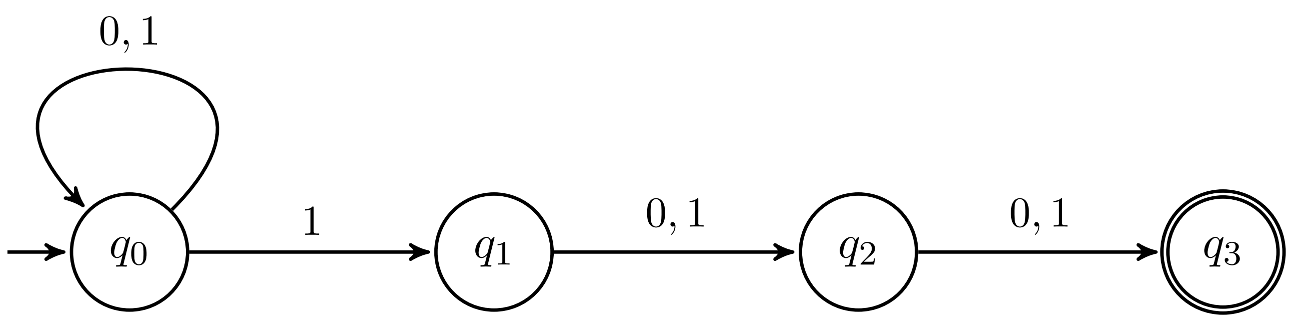
Foundations of Computing I

* All slides are a combined effort between previous instructors of the course

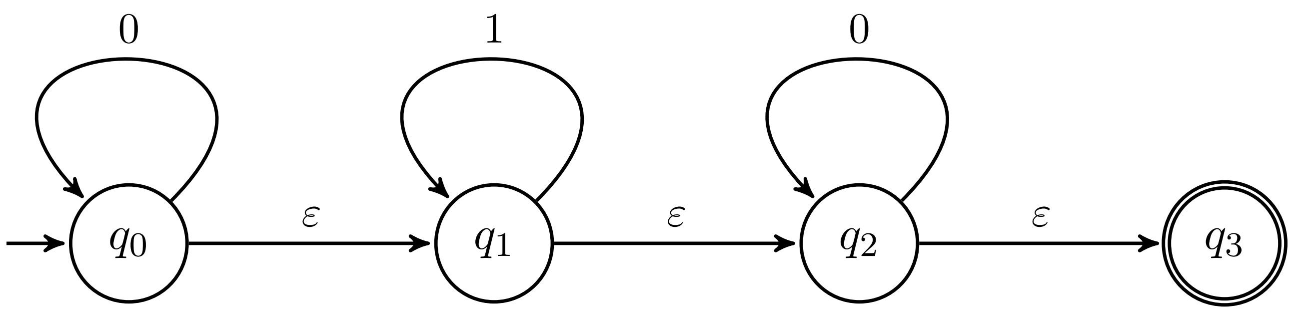
Construct an NFA for binary strings with a 1 three positions from the end



Construct an NFA for binary strings with a 1 three positions from the end



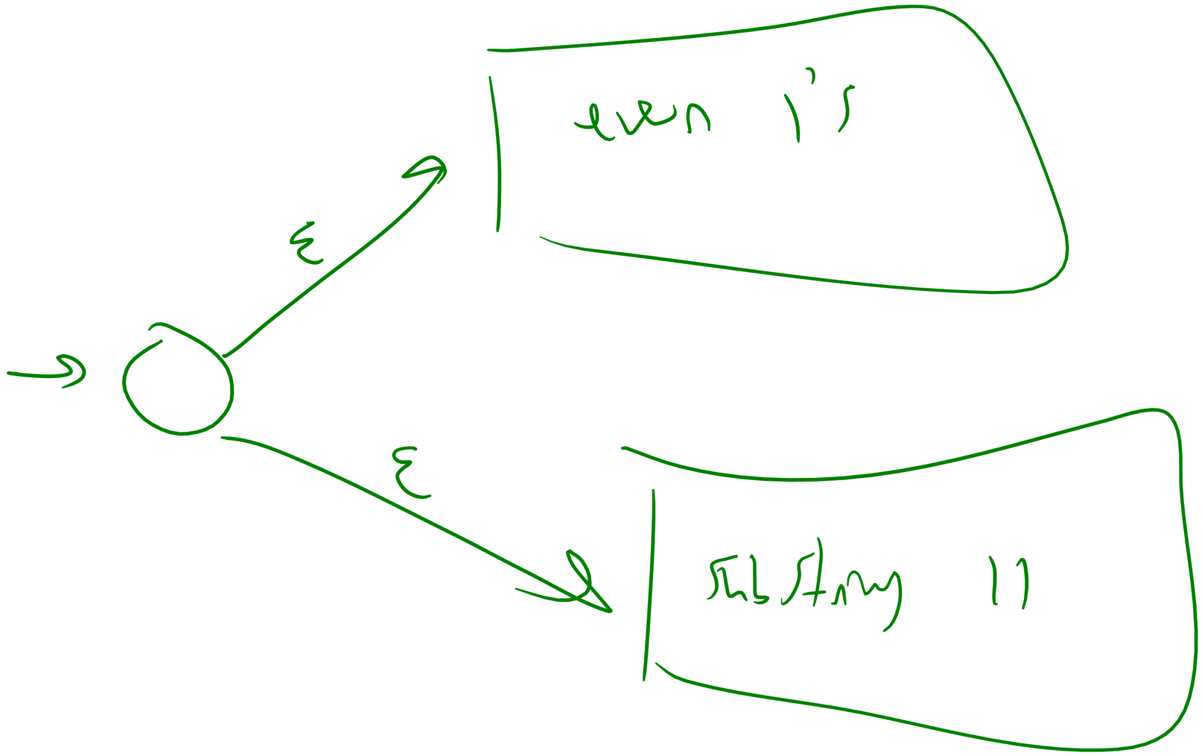
Epsilon Transitions



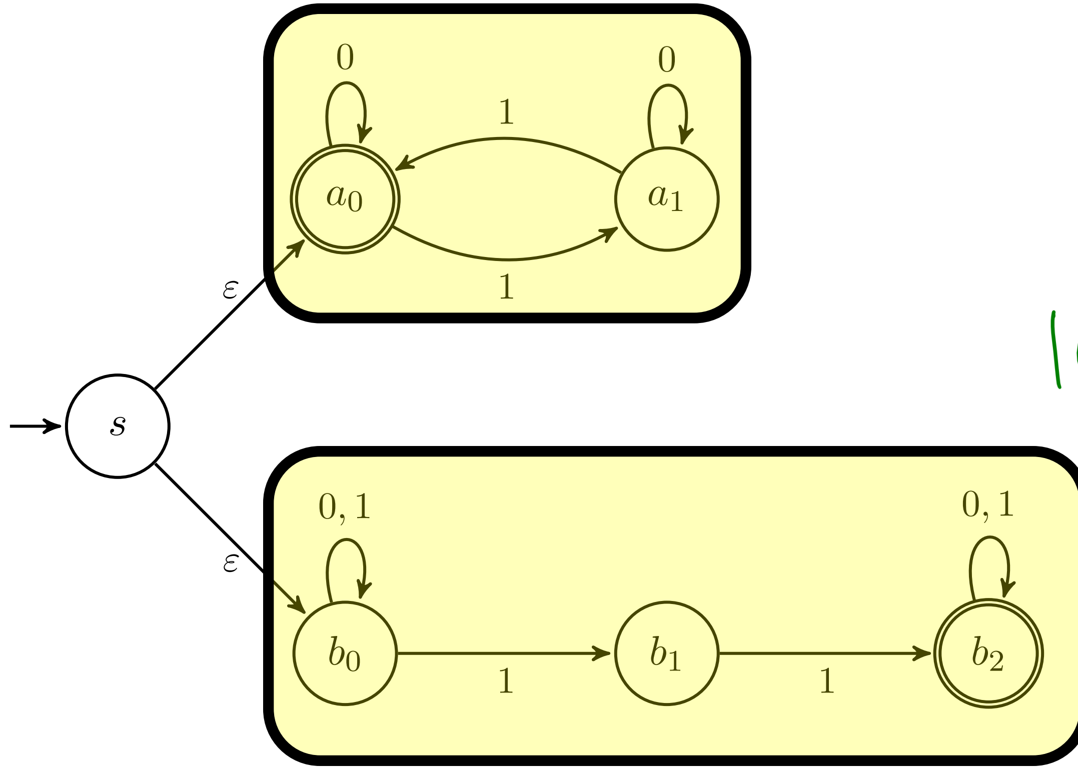
An "epsilon transition" is a transition in an NFA that **doesn't eat any of the string**. In other words, we may take it for free.

This NFA accepts the language `0*1*0*`.

Construct an NFA for binary strings with an even # of 1's or the substring 11



Construct an NFA for binary strings with an even # of 1's or the substring 11



101

The top machine accepts strings with an even number of 1's
The bottom machine accepts strings with the substring 11.

Since we have epsilon transitions to each, it's the union machine!

CSE 311: Foundations of Computing

Lecture 23: NFAs, Regular expressions, and NFA→DFA

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...

REG → ∅ | ε | a | REG* | REG ∪ REG |

REG REG

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:

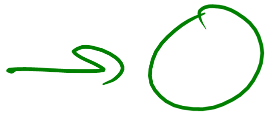
    (A $\cup$ B)

    (AB)

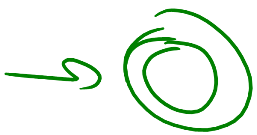
    A*

Base Case

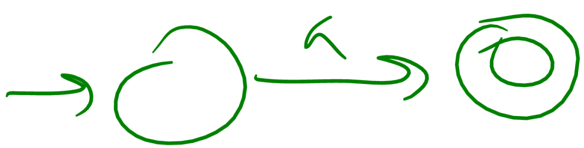
- Case ∅:



- Case ε:

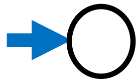


- Case *a*:

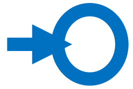


Base Case

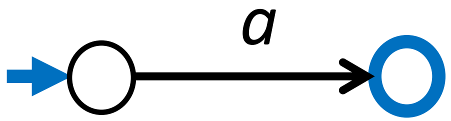
- Case ∅:



- Case ε:

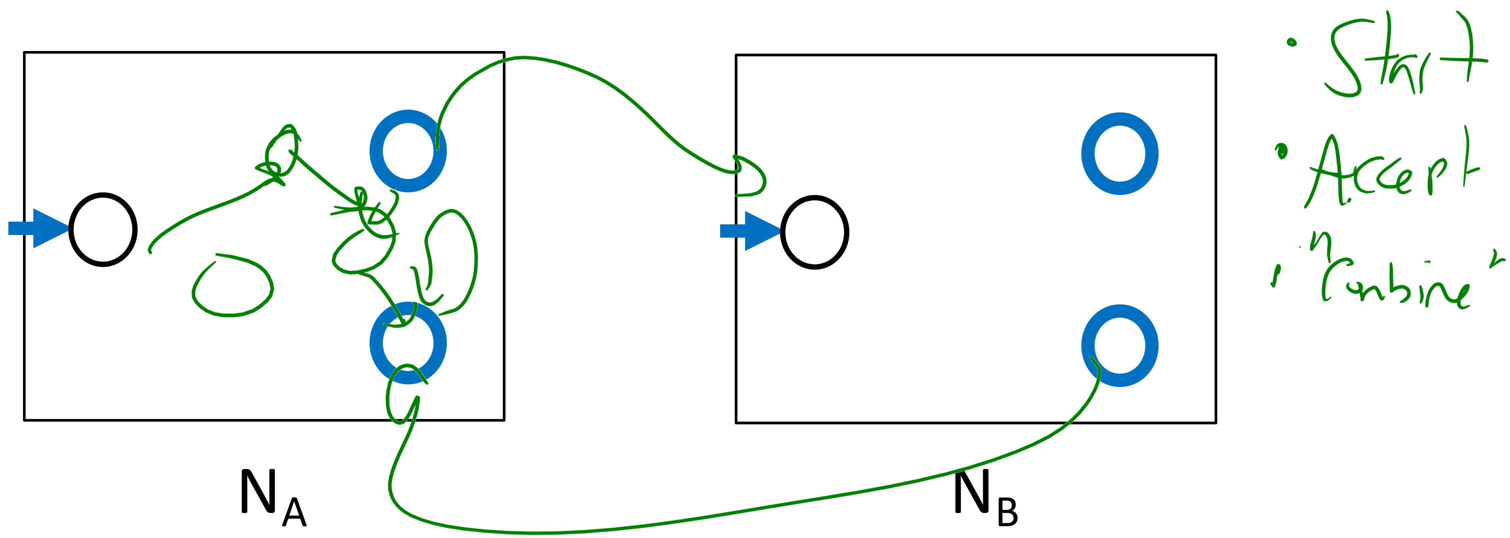


- Case *a*:



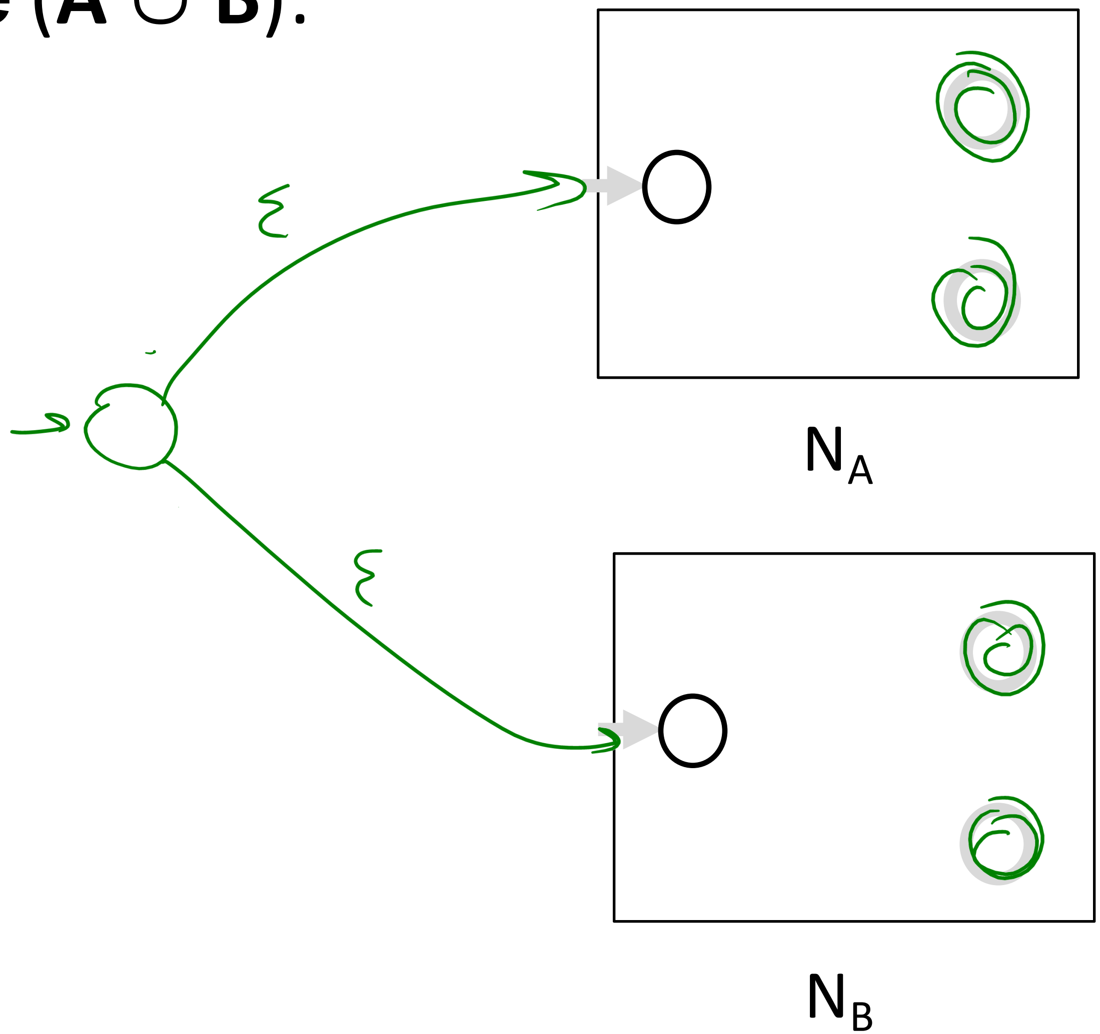
Inductive Hypothesis

- 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



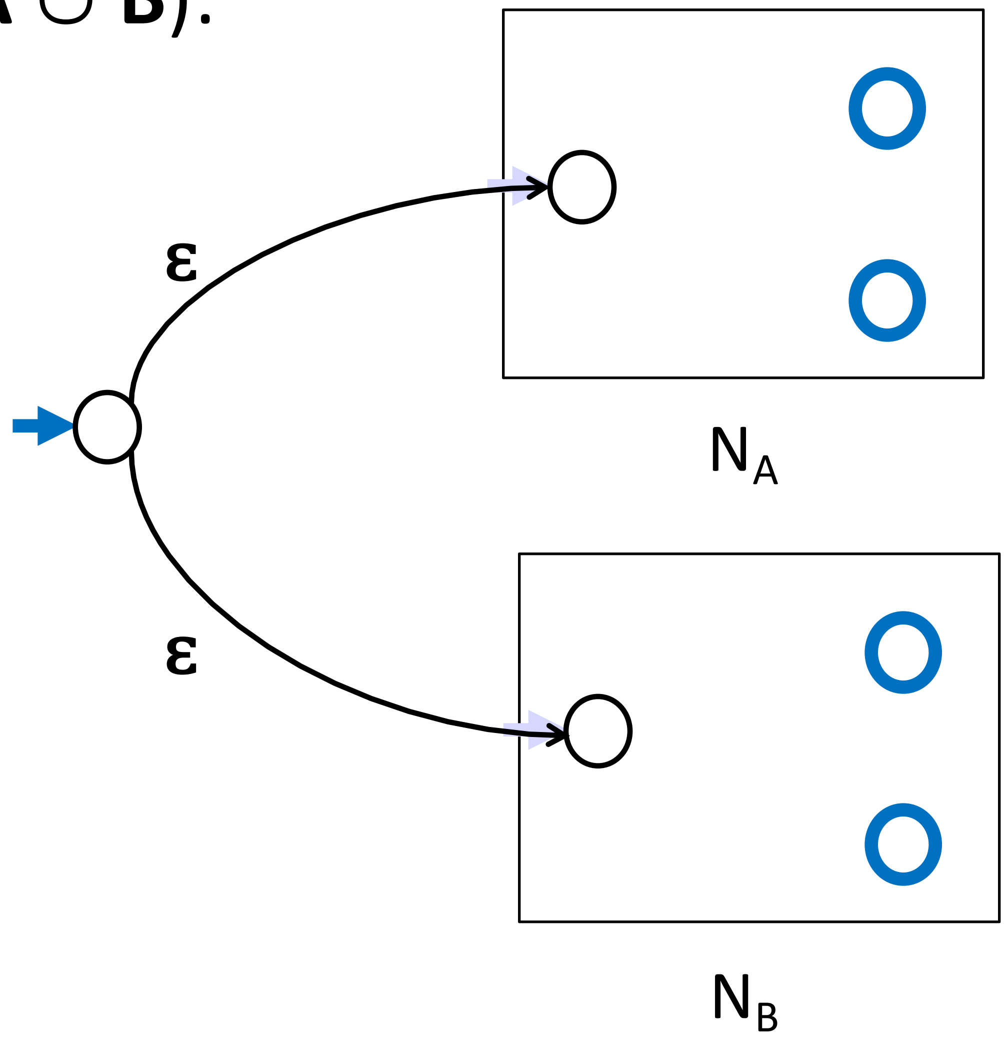
Inductive Step

Case (A ∪ B):



Inductive Step

Case (A ∪ B):

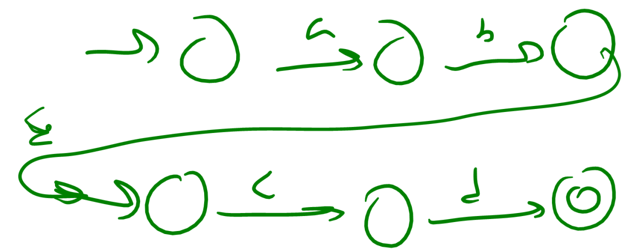


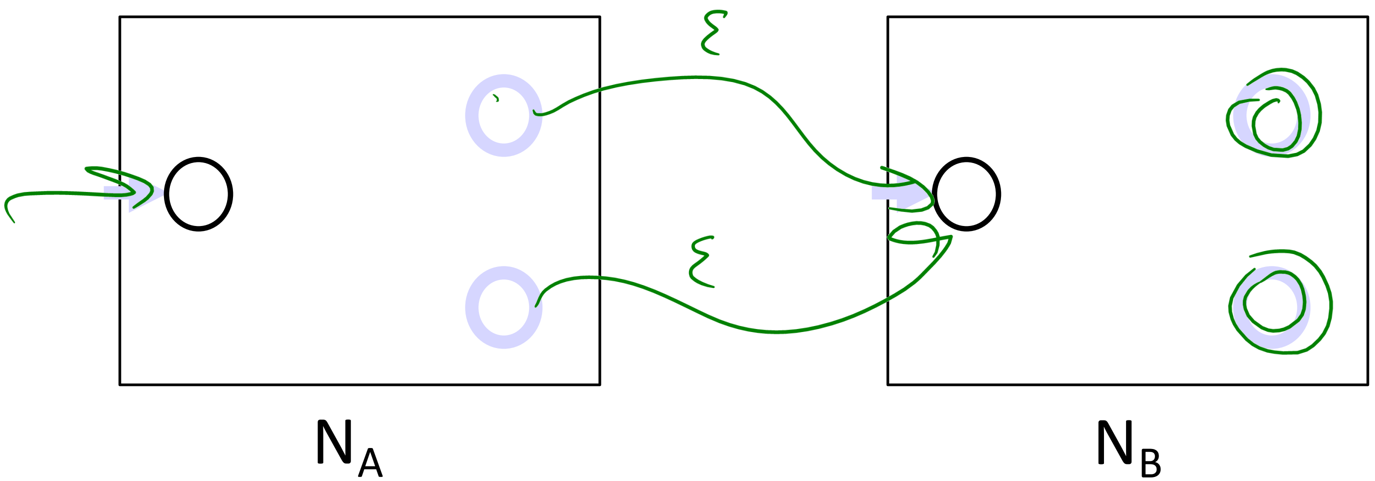
Inductive Step

Case (AB):

A = ab

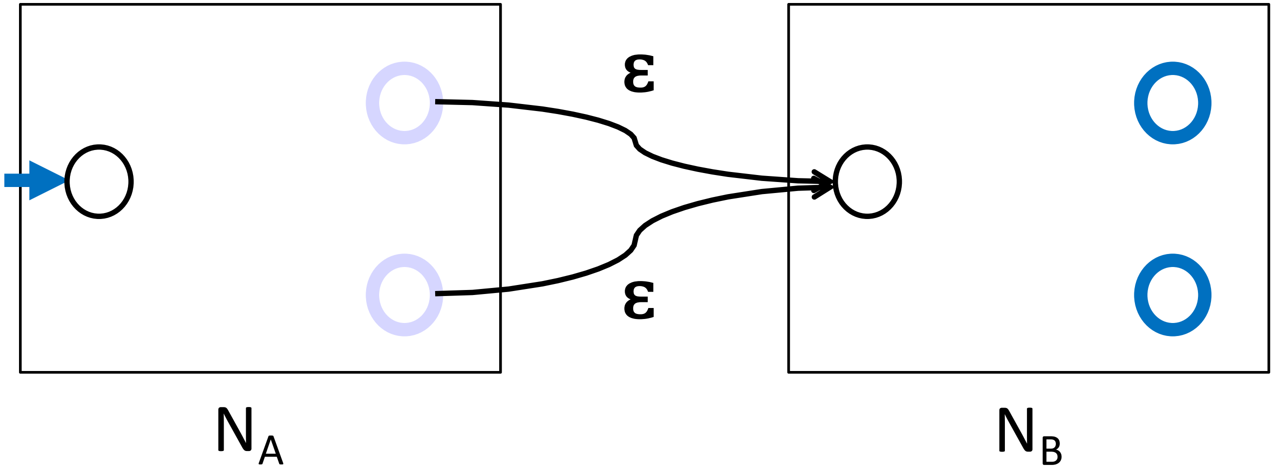
B = cd





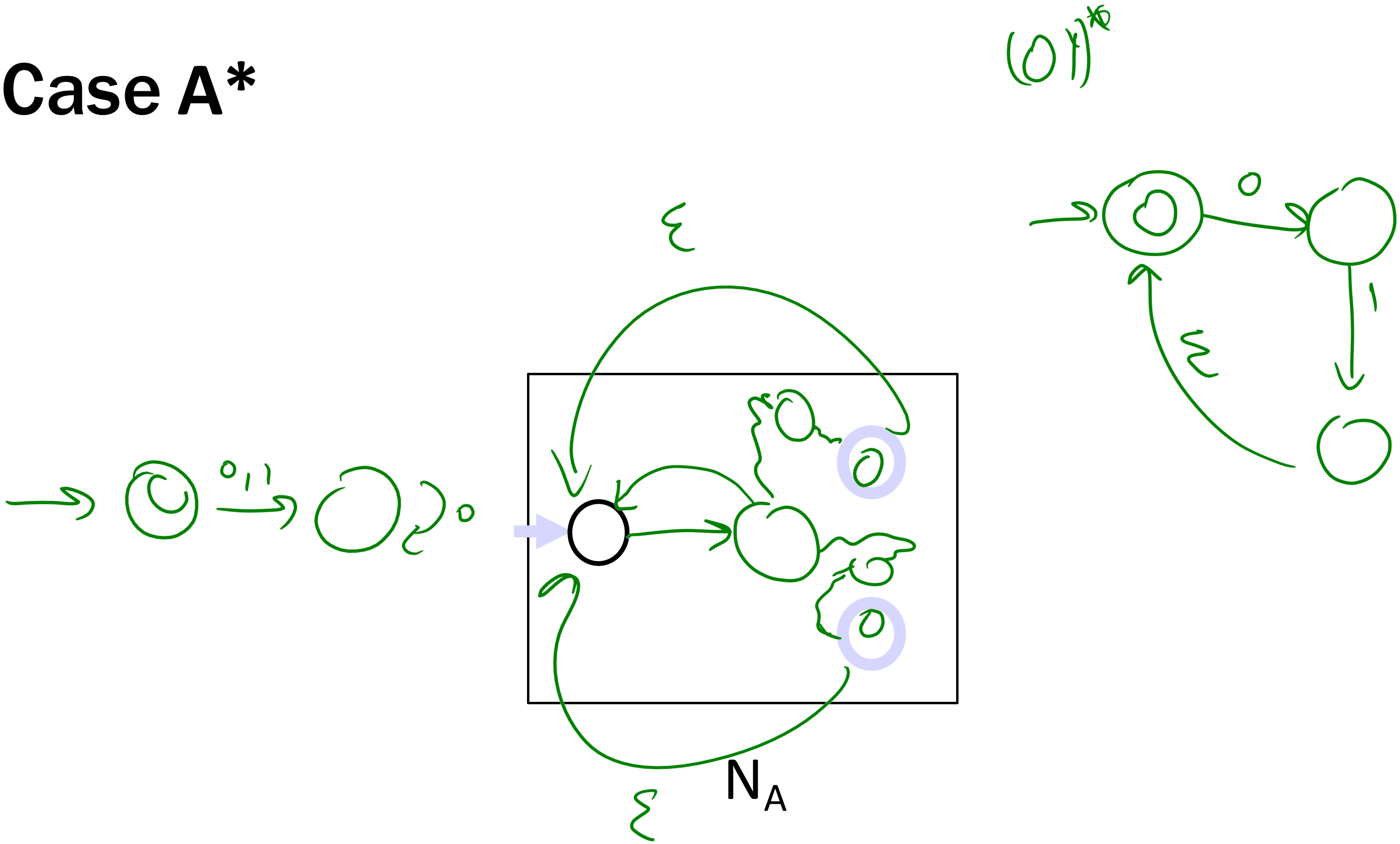
Inductive Step

Case (AB):



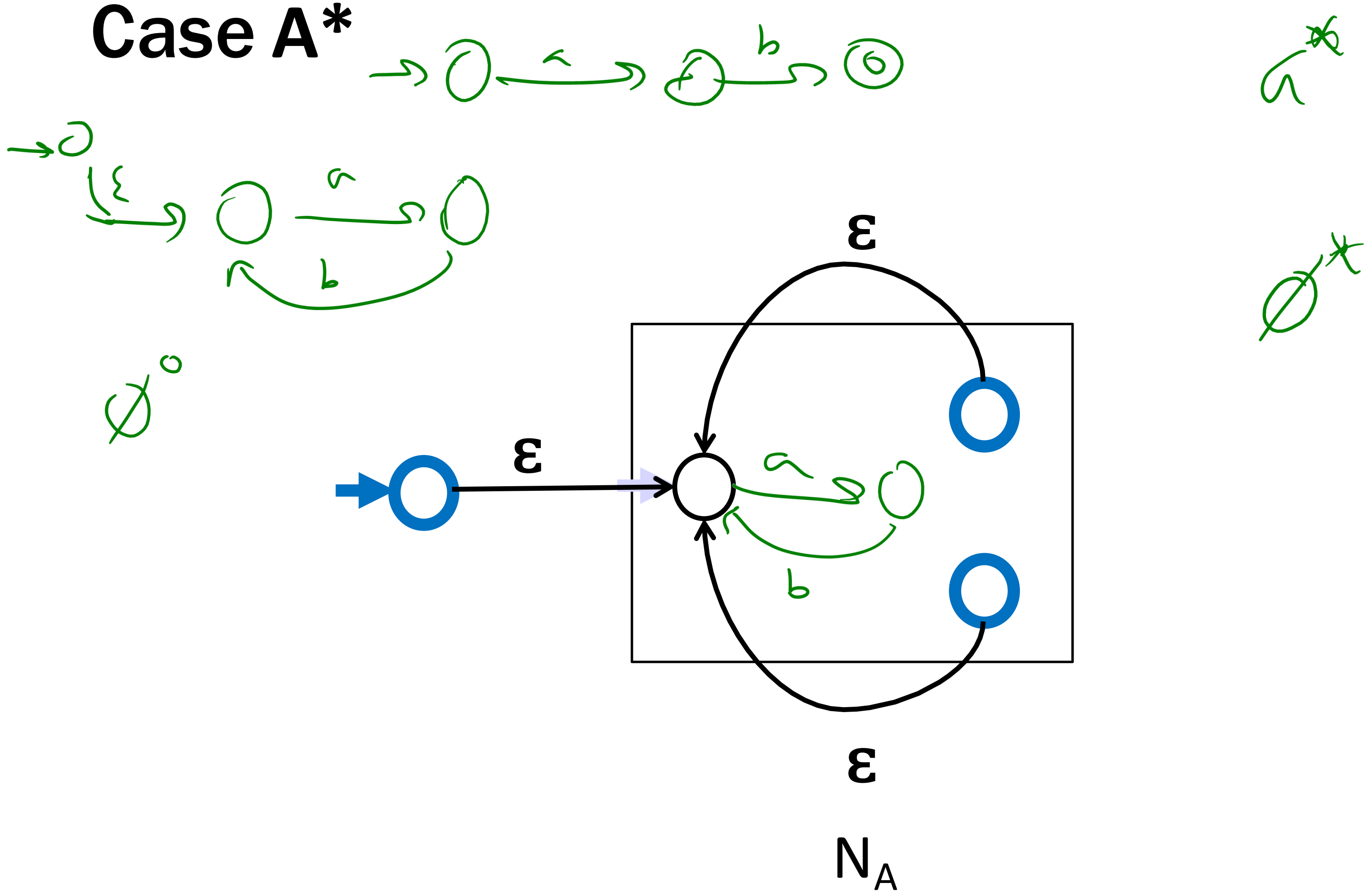
Inductive Step

Case A*



Inductive Step

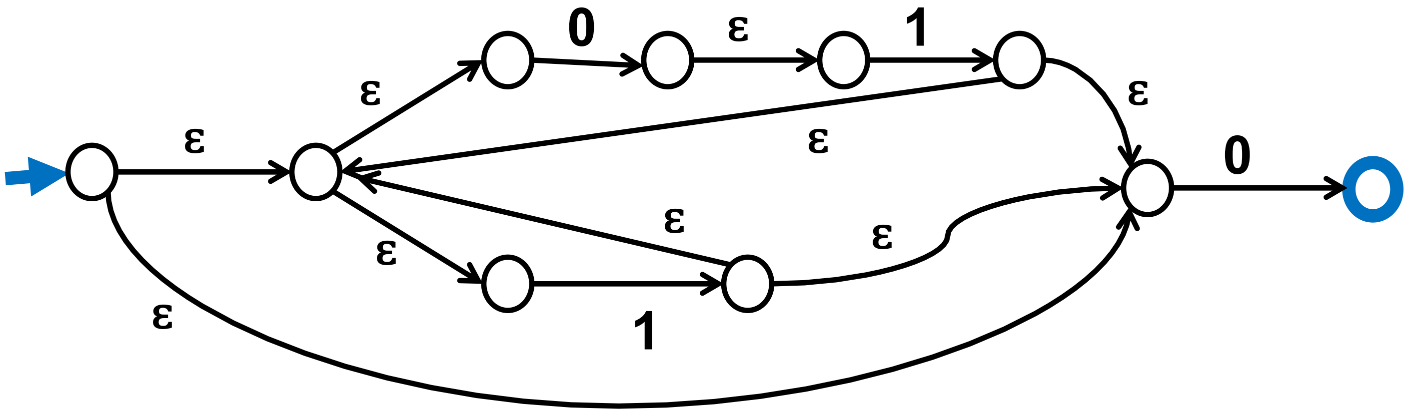
Case A*



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

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

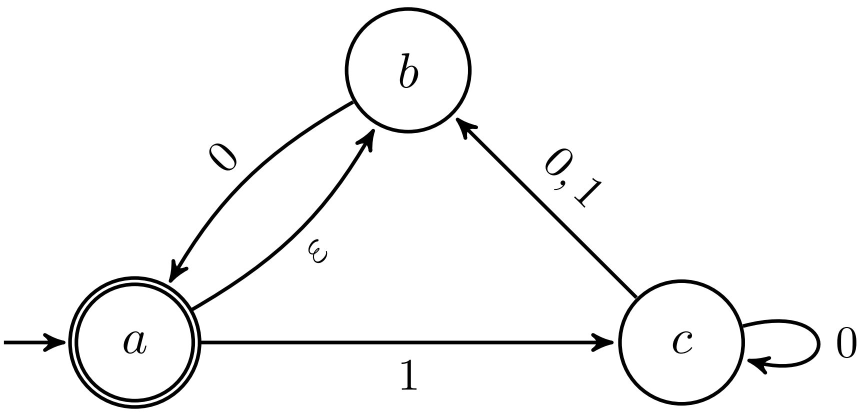
Every DFA is an NFA

- 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

Example: NFA to DFA



NFA

DFA

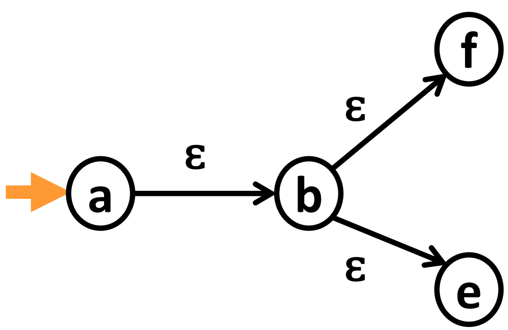
Conversion of NFAs to a DFAs

- 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

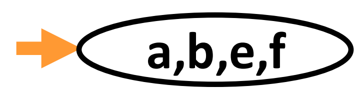
Conversion of NFAs to a DFAs

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

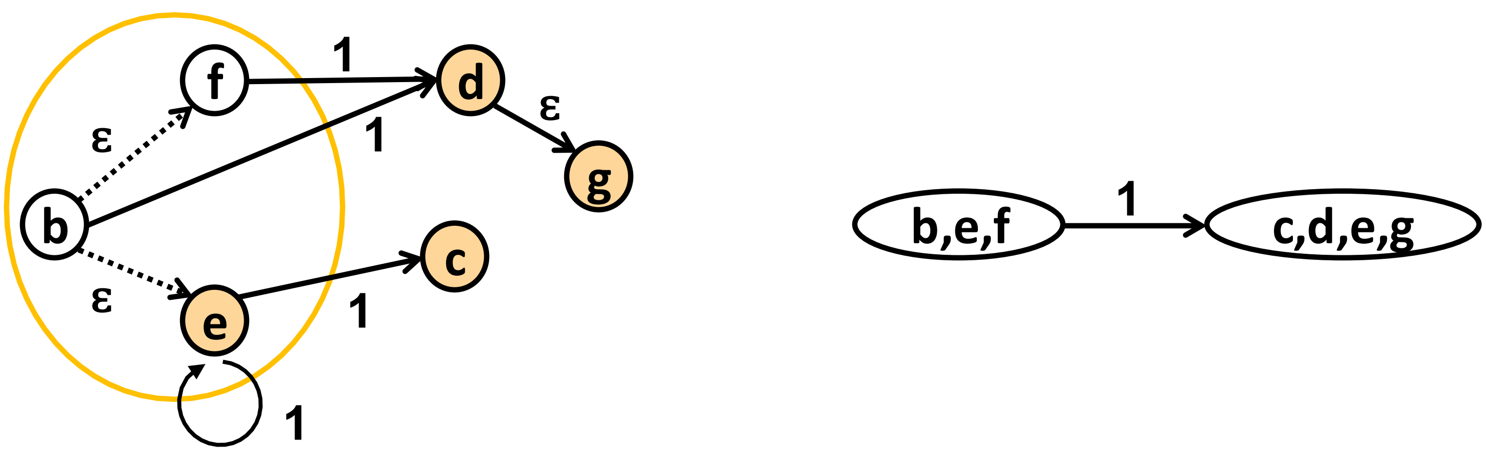
Conversion of NFAs to a DFAs

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**

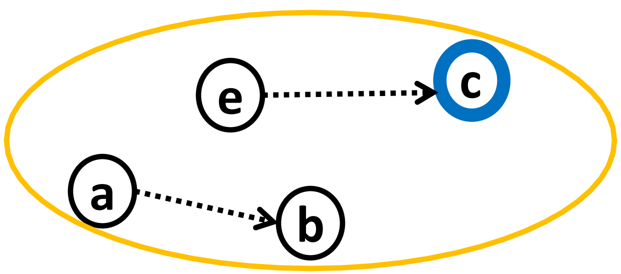
- T will be ∅ if no edges from S labeled **s** exist



Conversion of NFAs to a DFAs

Final states for the DFA

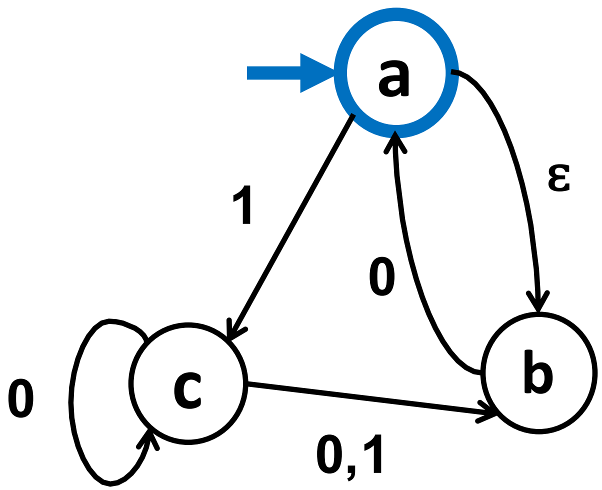
– All states whose set contain some final state of the NFA



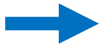
NFA

DFA

Example: NFA to DFA

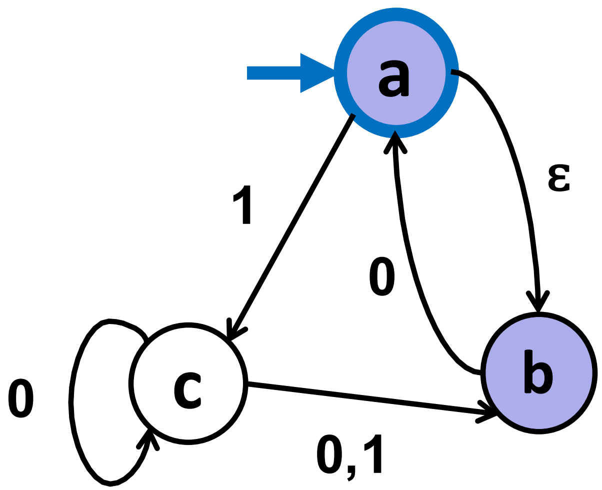


NFA

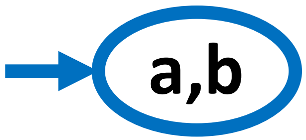


DFA

Example: NFA to DFA

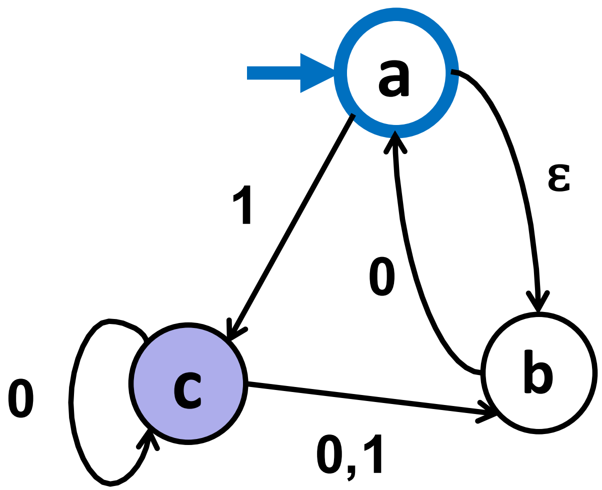


NFA

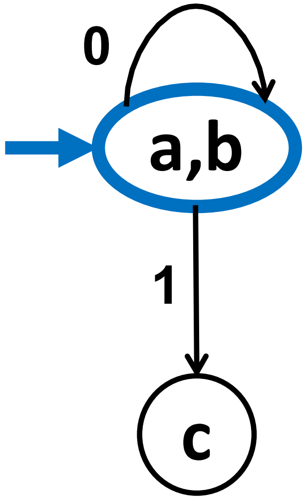


DFA

Example: NFA to DFA

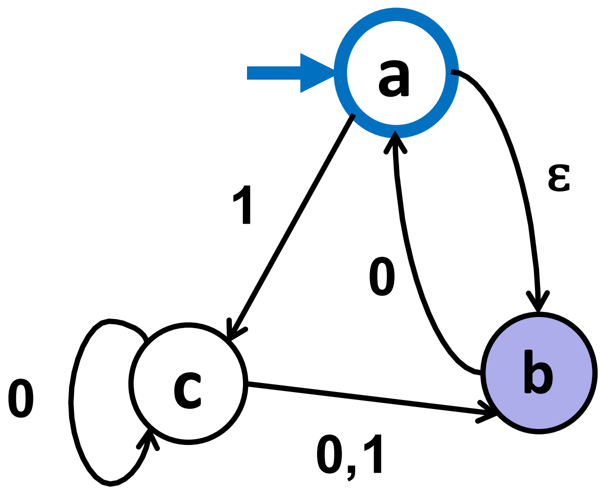


NFA

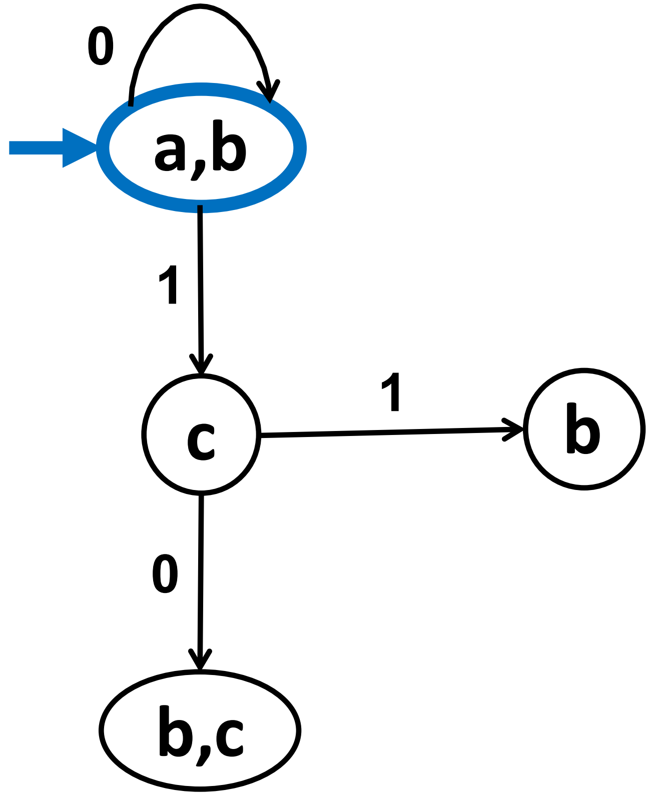


DFA

Example: NFA to DFA

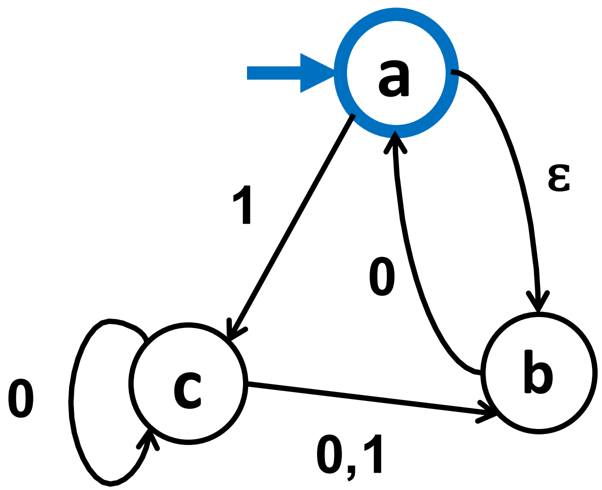


NFA

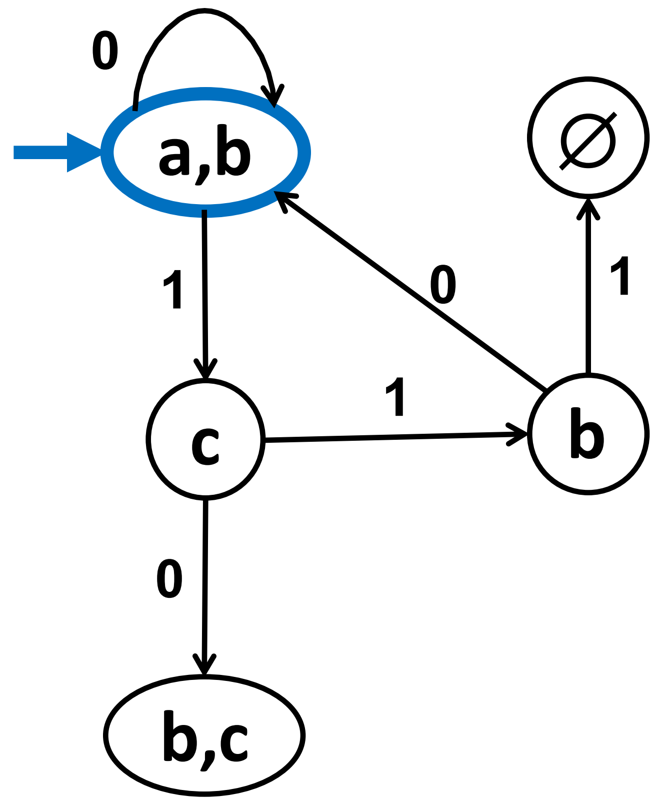


DFA

Example: NFA to DFA

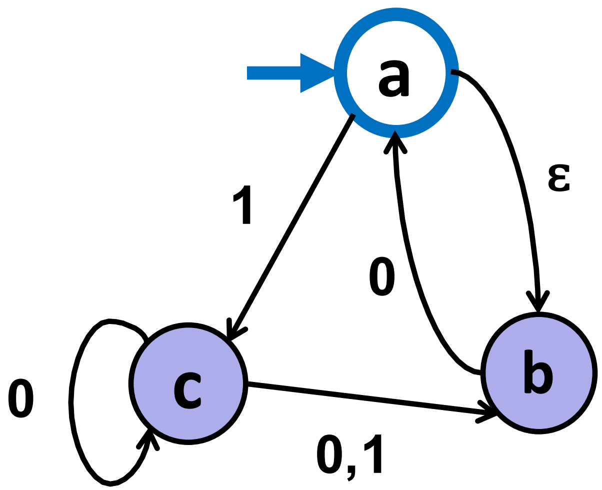


NFA

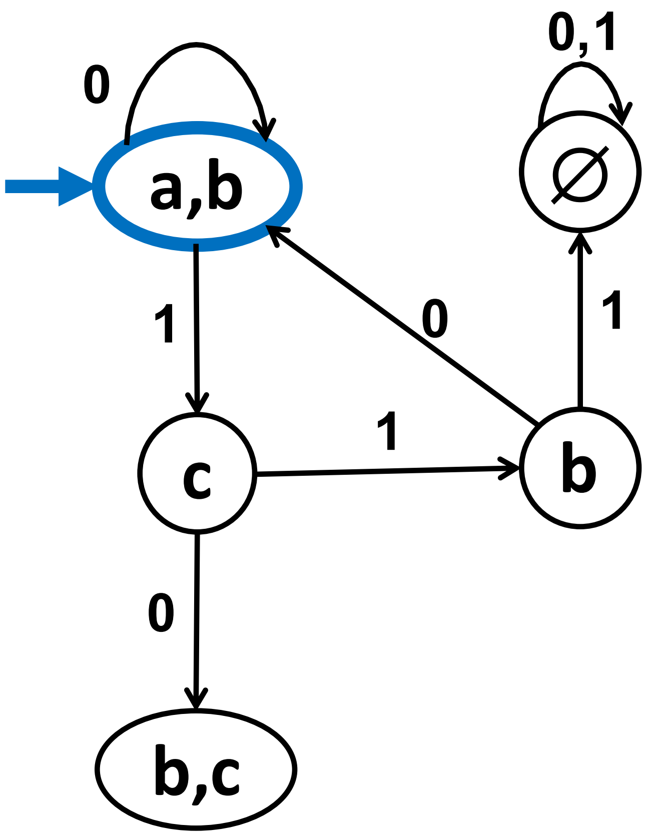


DFA

Example: NFA to DFA

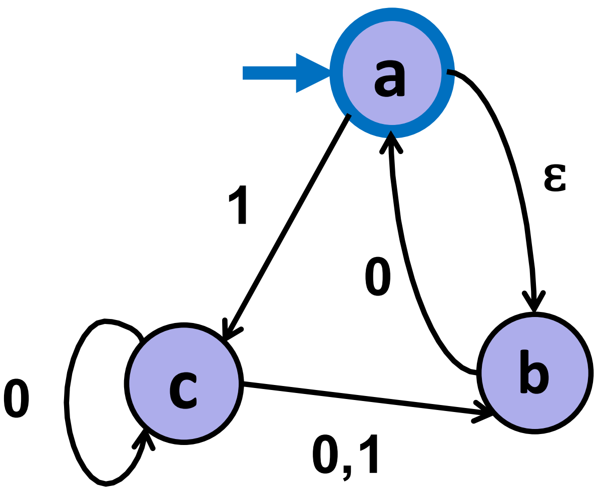


NFA

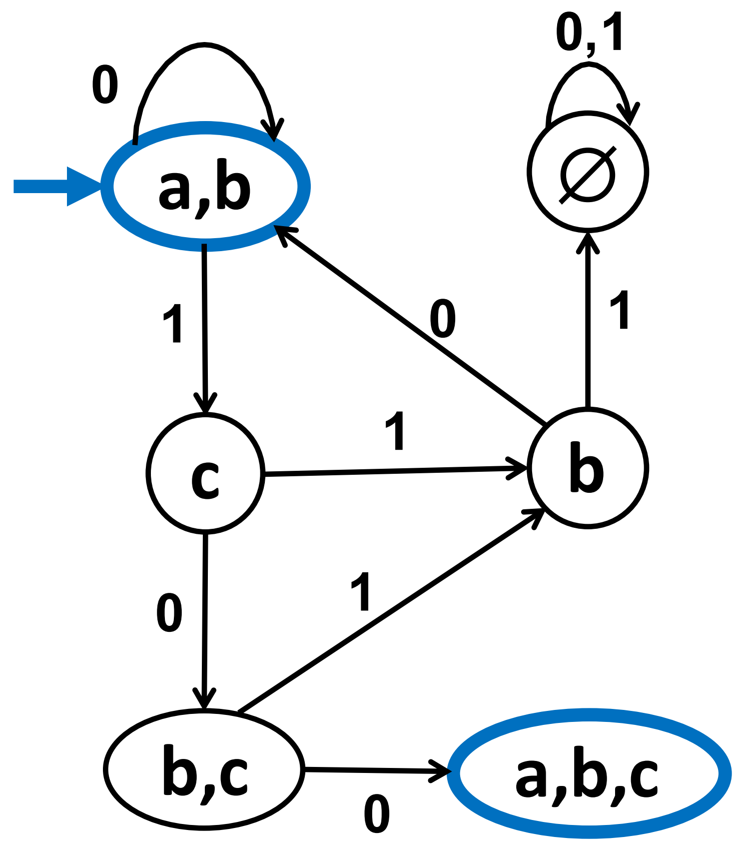


DFA

Example: NFA to DFA

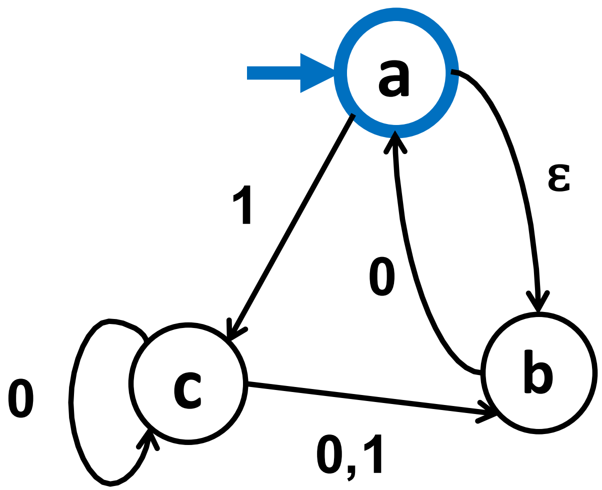


NFA

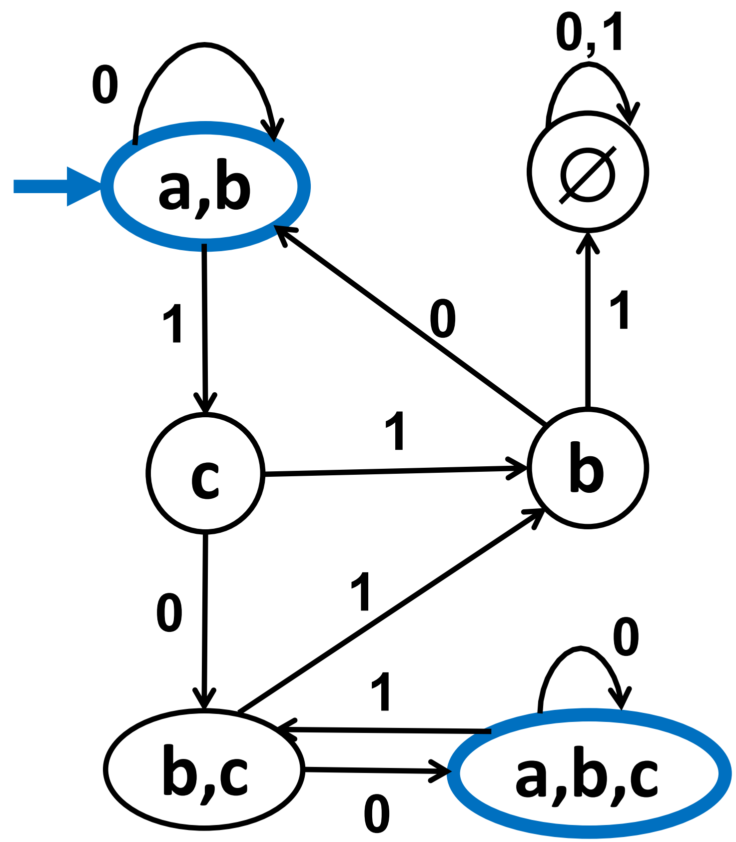


DFA

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
  - n-state NFA yields DFA with at most 2^n states
  - We saw an example where roughly 2^n is necessary

    Is the n^{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