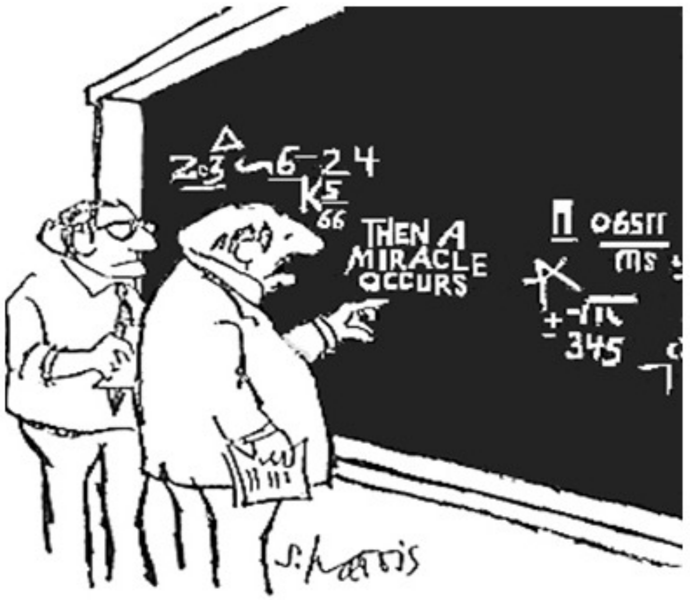
CSE 311: Foundations of Computing

Lecture 25: NFAs and their relation to REs & DFAs

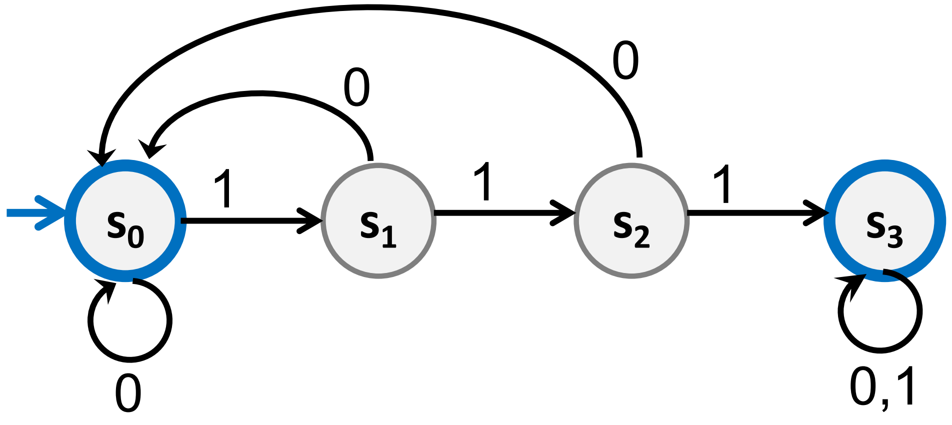


"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

Recall: DFAs

- States
- Transitions on input symbols
- Start state and final states
- The "language recognized" by the machine is the set of strings that reach a final state from the start

Old State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3

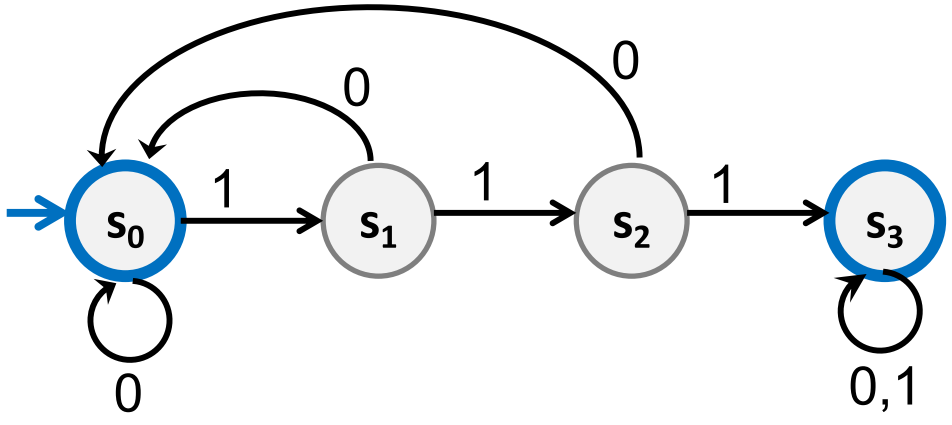


Recall: DFAs

- Each machine designed for strings over some fixed alphabet Σ.

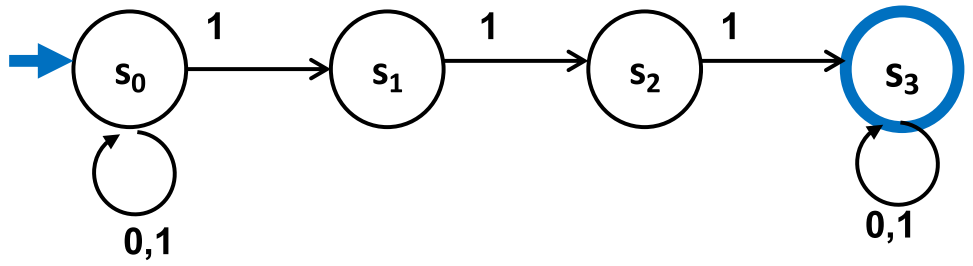
- Must have a transition defined from each state for ***every*** symbol in Σ.

Old State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



Last Time: 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 <u>some</u> valid execution of the machine gets to an accept state



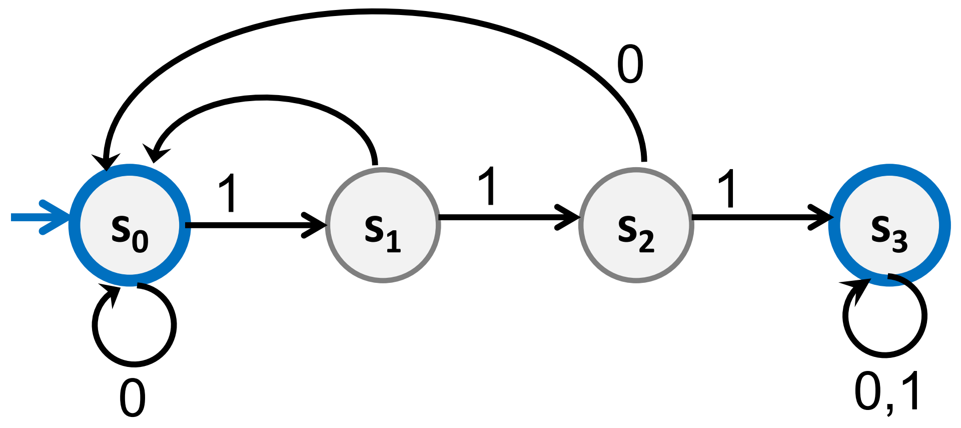
Three ways of thinking about NFAs

- 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)
- Outside observer: Is there a path labeled by x from the start state to some accepting state?
- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

Path Labels

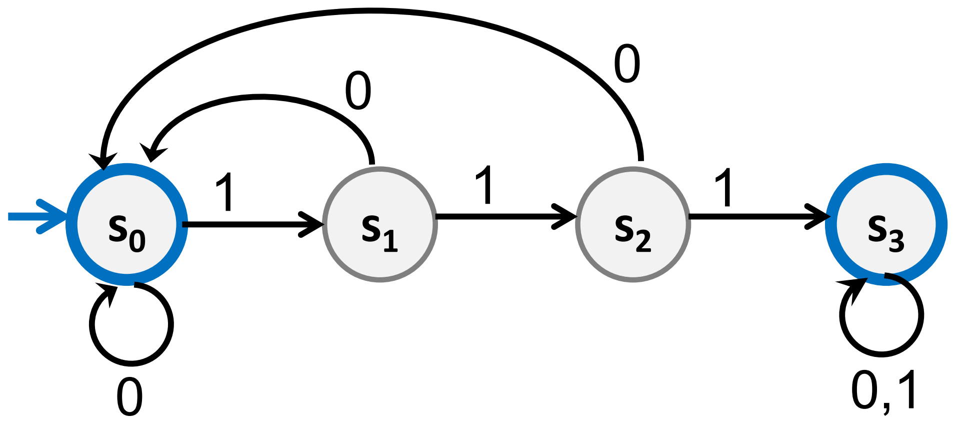
Def: The label of path $v_0, v_1, \ldots, v_n$ is the
<u>concatenation</u> of the labels of the edges
$(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)$

Example: The label of path s_0, s_1, s_2, s_0, s_0 is 1100



Deterministic Finite Automata (DFA)

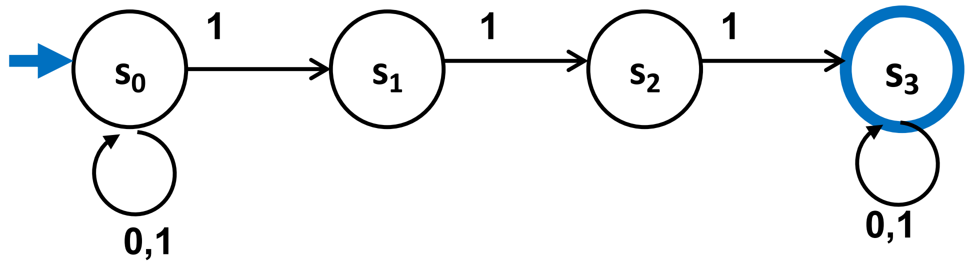
- **Def:** x is in the language recognized by an DFA if and only if x labels a path from the start state to some final state



- Path $v_0, v_1, \ldots, v_n$ with $v_0 = s_0$ and label x describes a correct simulation of the DFA on input x
  - i-th step must match the i-th character of x

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 <u>some</u> **path** from the start state to an accepting state



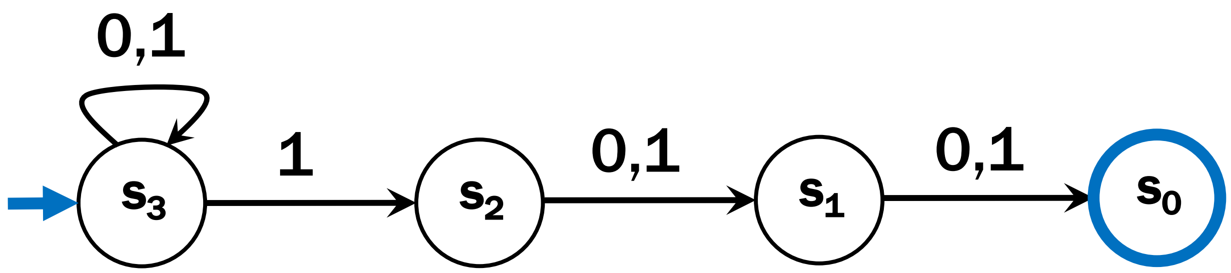
Three ways of thinking about NFAs

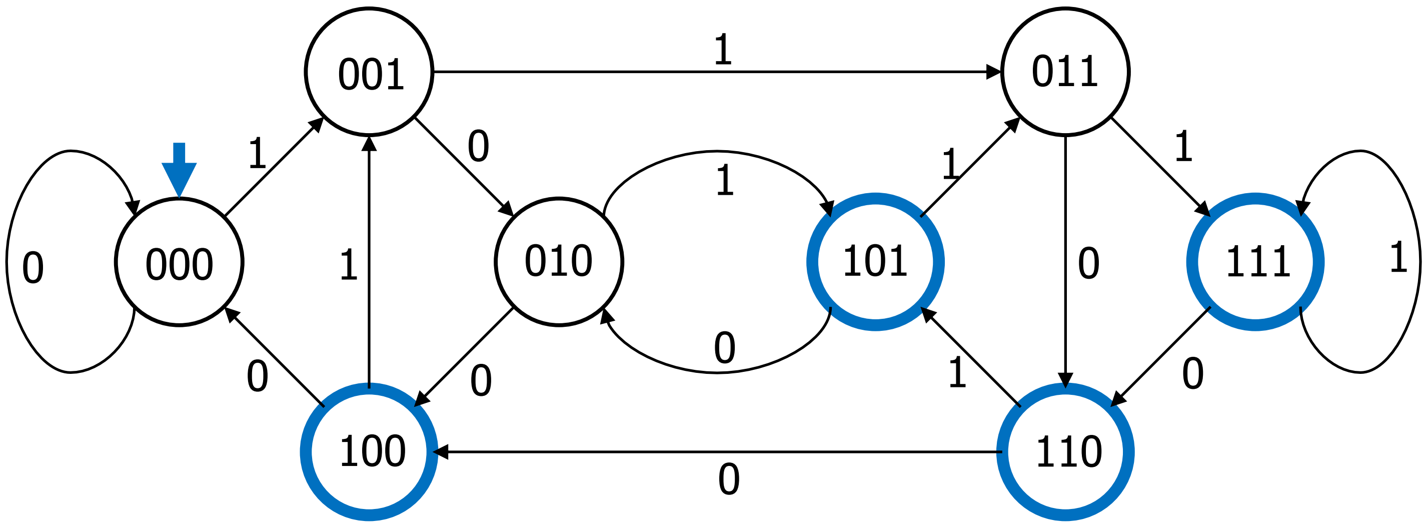
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)

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

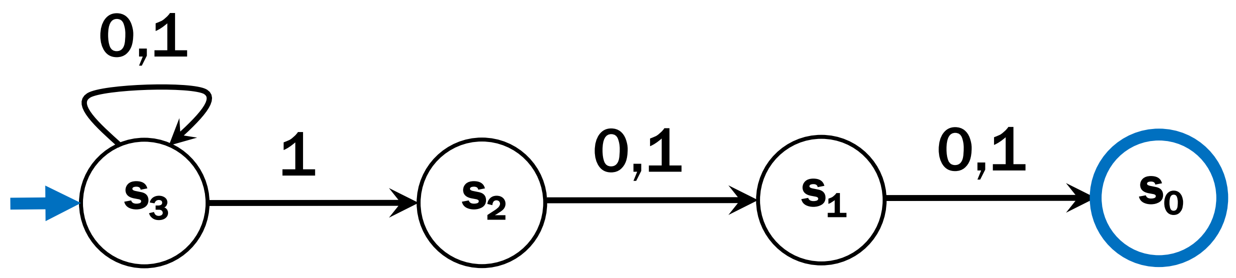
- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

Compare with the smallest DFA

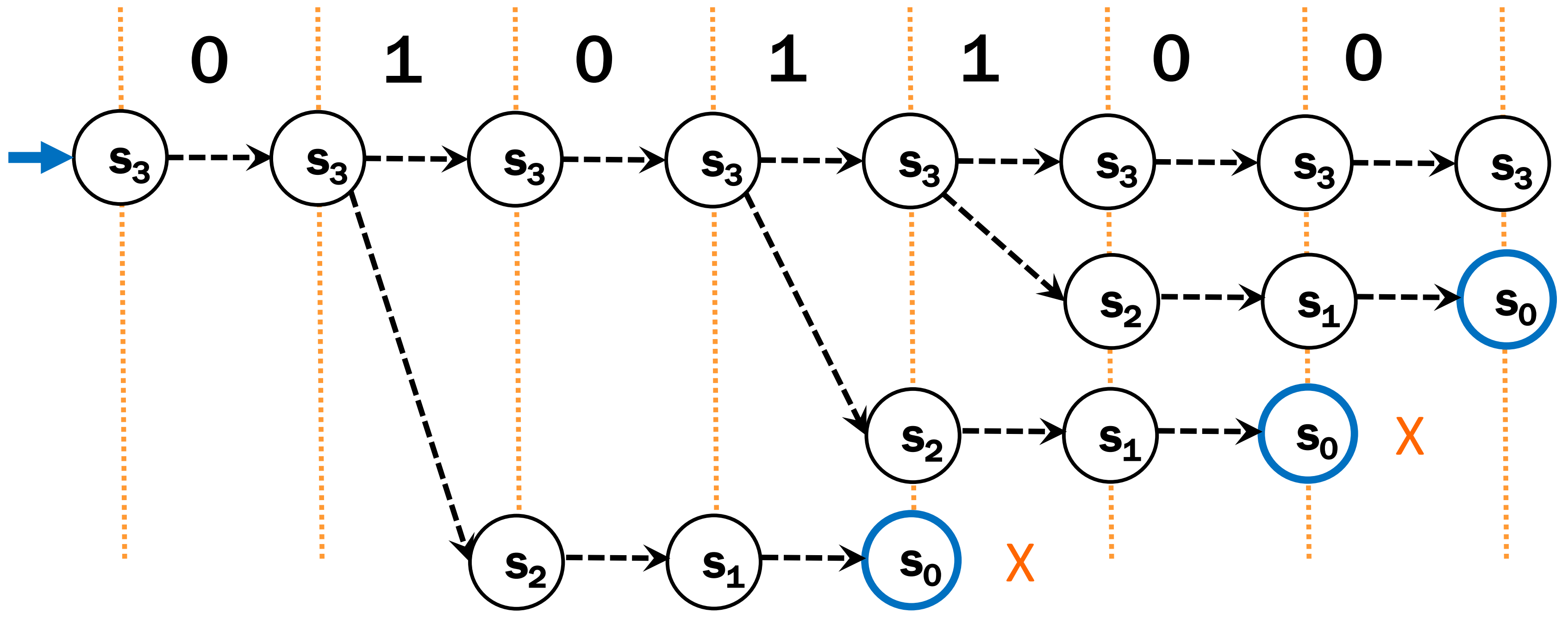




Parallel Exploration view of an NFA



Input string 0101100



Summary of NFAs

- Generalization of DFAs
  - drop two restrictions of DFAs
  - every DFA <u>is</u> an NFA
- *Seem* to be more powerful
  - designing is easier than with DFAs
- *Seem* related to regular expressions

The story so far...

$$\boxed{\text{REs}} \subseteq \boxed{\text{CFGs}}$$

$$\boxed{\text{DFAs}} \subseteq \boxed{\text{NFAs}}$$

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:**
  - ε is a regular expression
  - ***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*

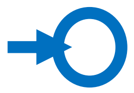
Base Case

- Case ε:

- Case *a*:

Base Case

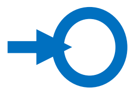
- Case ε:



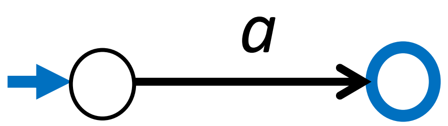
- Case ***a***:

Base Case

- Case ε:

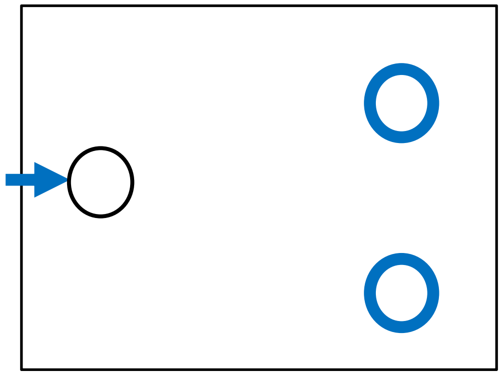


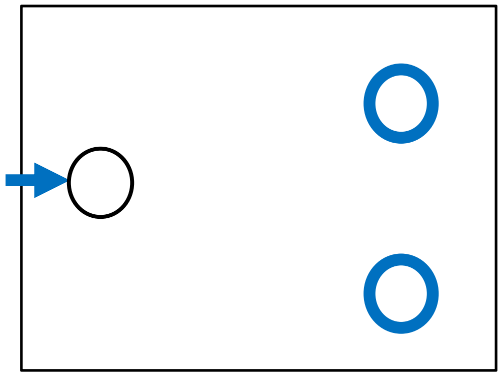
- Case ***a***:



Inductive Hypothesis

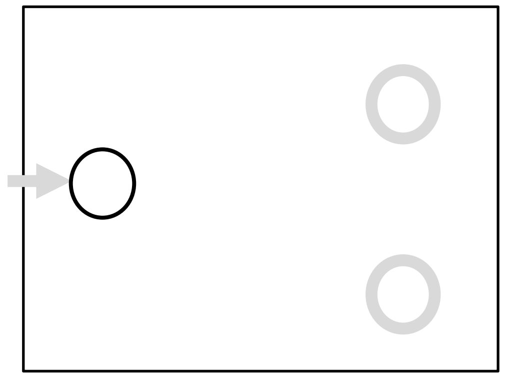
- Suppose that for some regular expressions A and B there exist NFAs N_A and N_B such that N_A recognizes the language given by A and N_B recognizes the language given by B


N_A

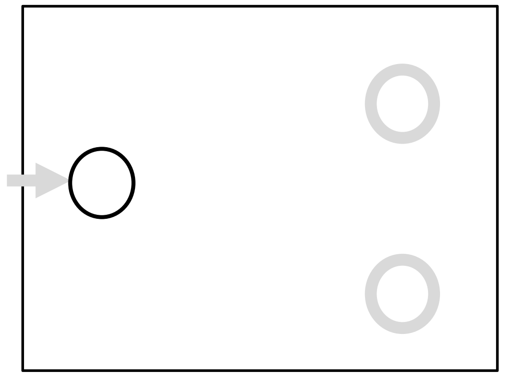

N_B

Inductive Step

Case A ∪ B:



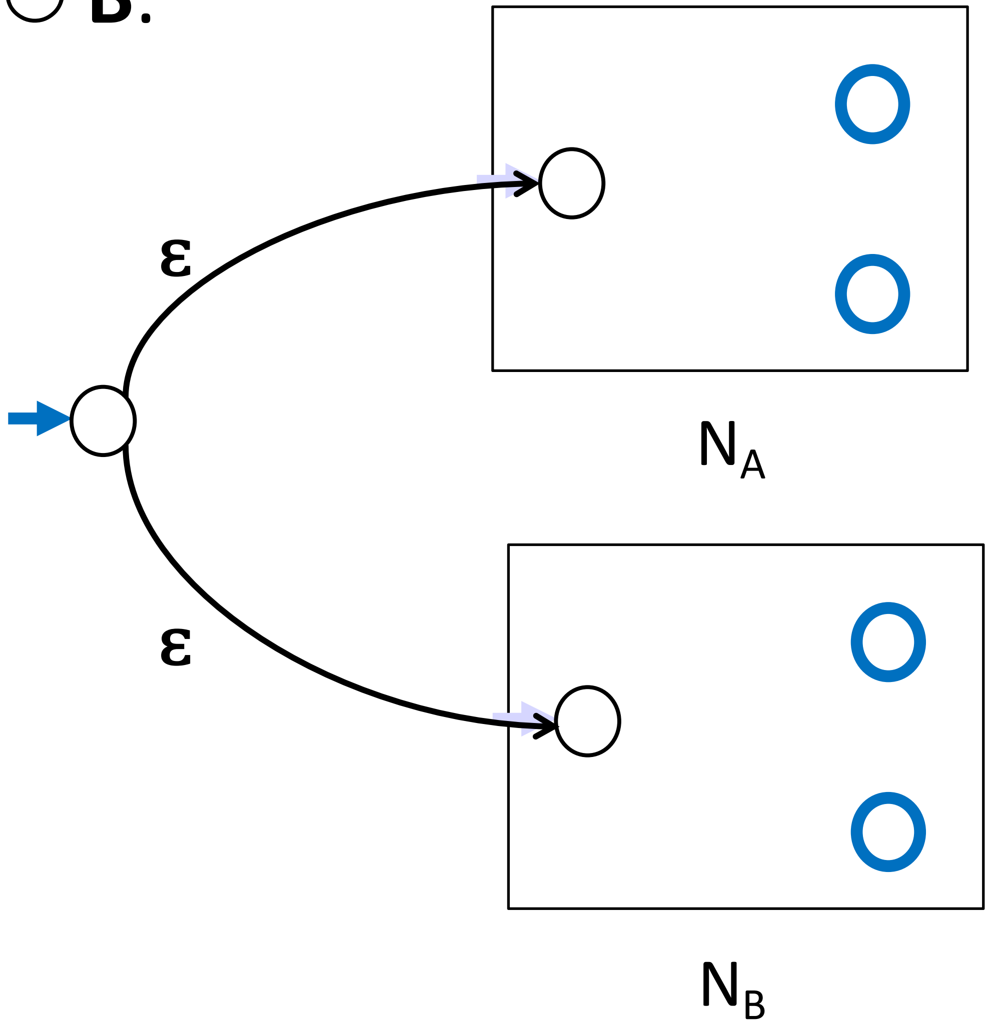
N_A



N_B

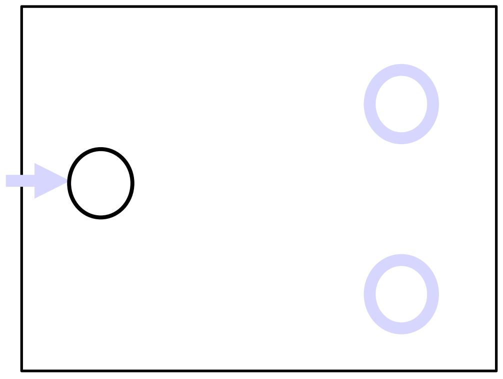
Inductive Step

Case A ∪ B:

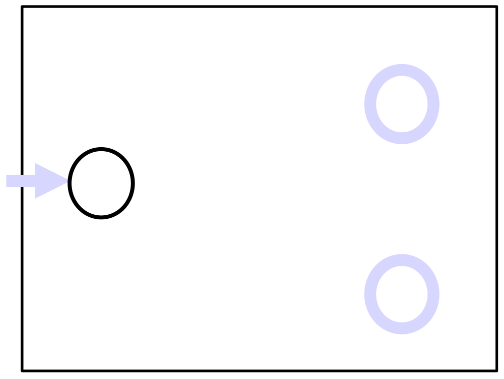


Inductive Step

Case AB:



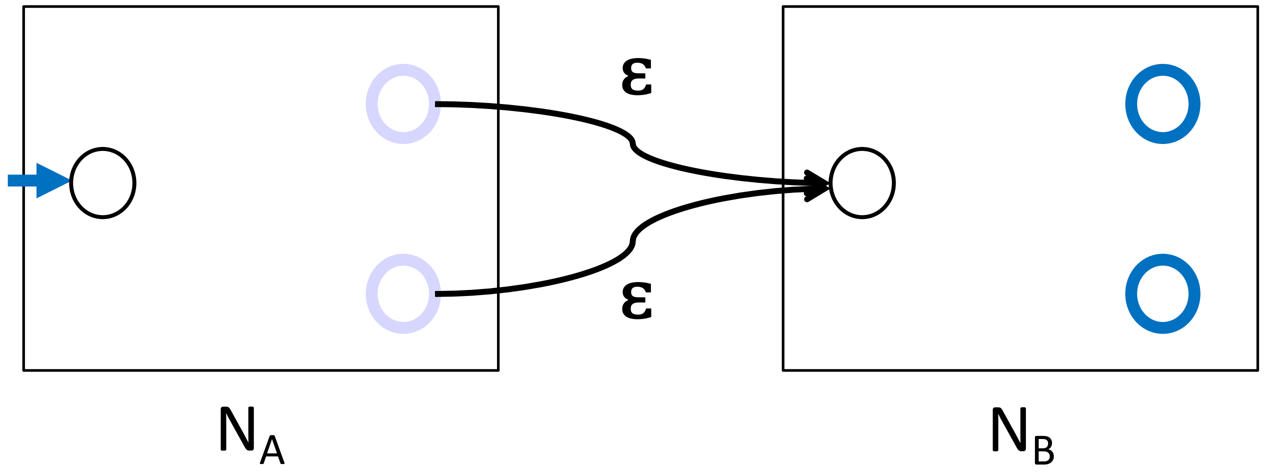
N_A



N_B

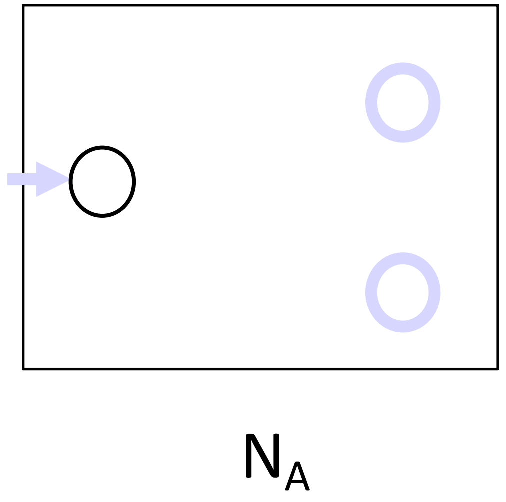
Inductive Step

Case AB:



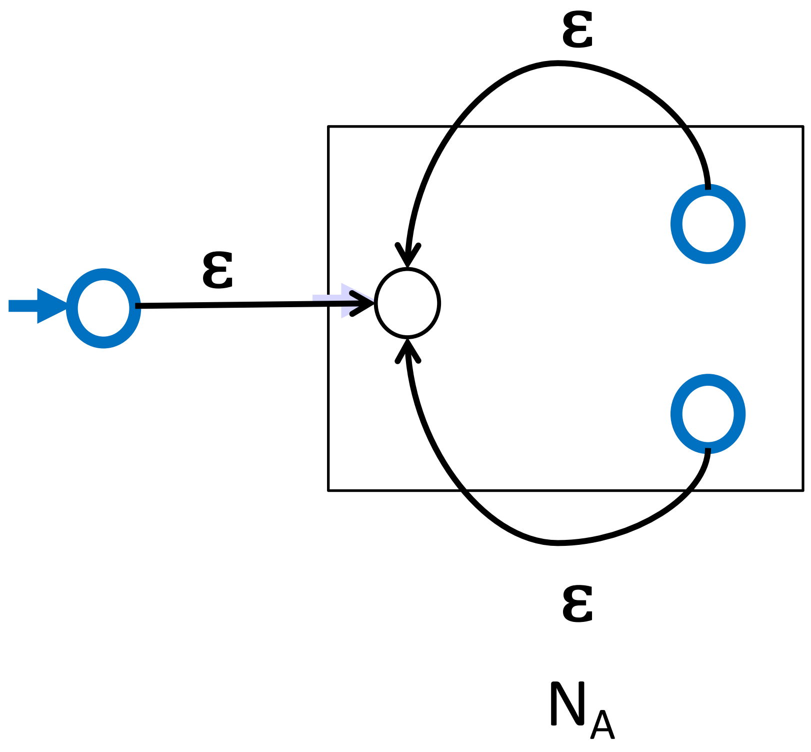
Inductive Step

Case A*



Inductive Step

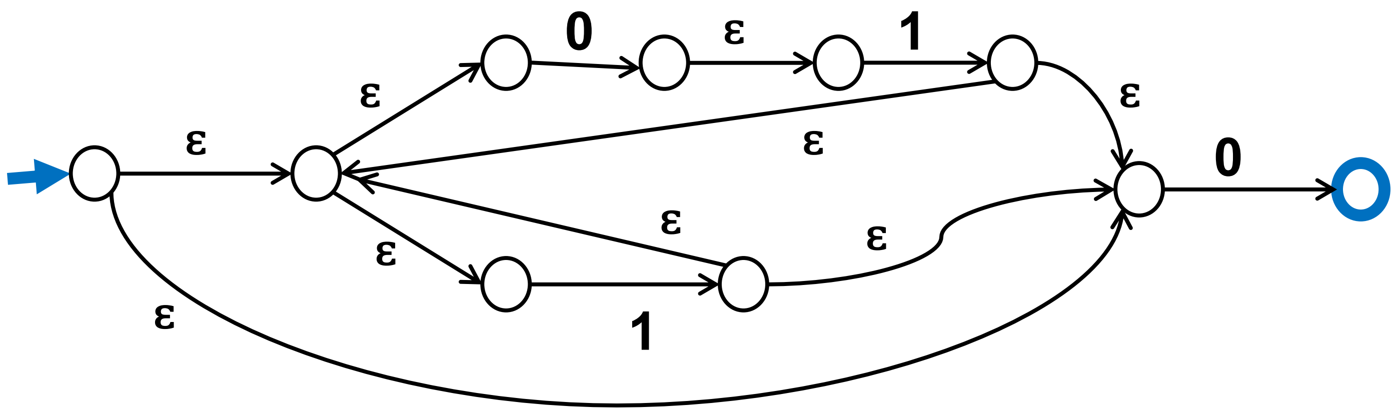
Case A*



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

Solution

(01 ∪1)*0



The story so far...

$$\begin{tikzcd}
\boxed{\text{REs}} \arrow[r, phantom, "\subseteq"] & \boxed{\text{CFGs}} \\
\boxed{\text{DFAs}} \arrow[r, phantom, "\subseteq"] & \boxed{\text{NFAs}} \arrow[ul, phantom, sloped, "\subseteq"]
\end{tikzcd}$$

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

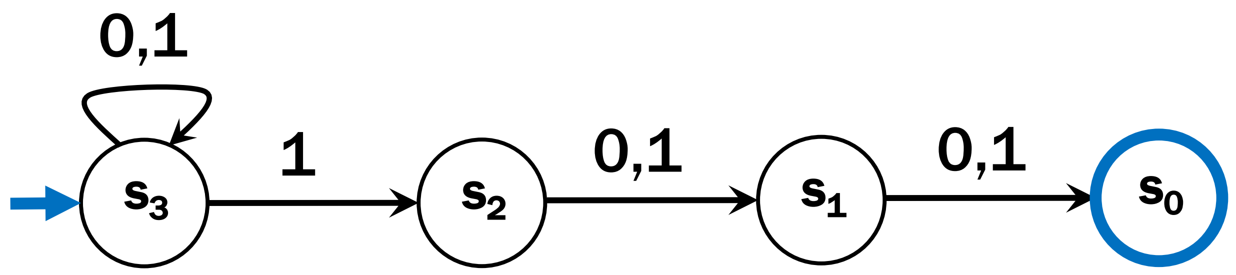
Can NFAs recognize more languages? No!

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

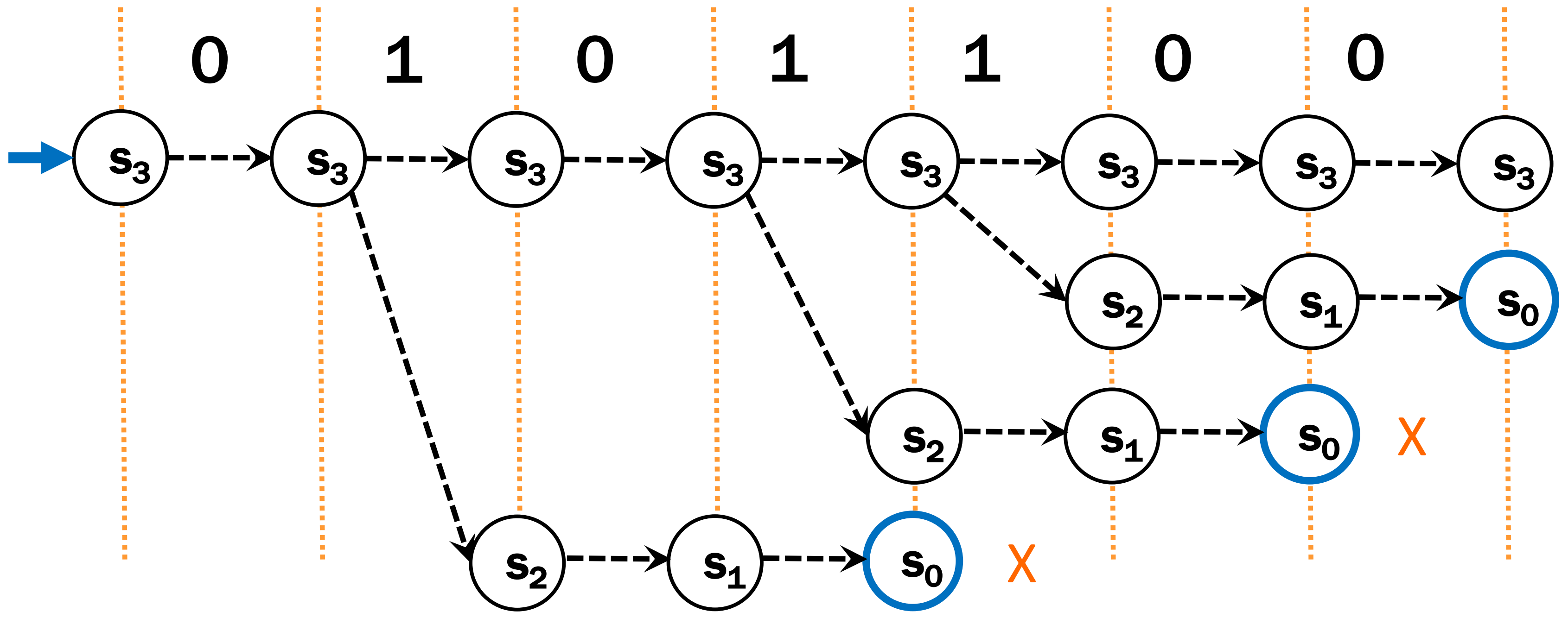
Three ways of thinking about NFAs

- 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

Parallel Exploration view of an NFA



Input string 0101100



Conversion of NFAs to a DFAs

- Construction Idea:
  - The DFA keeps track of ALL states reachable in the NFA along a path labeled by the input so far

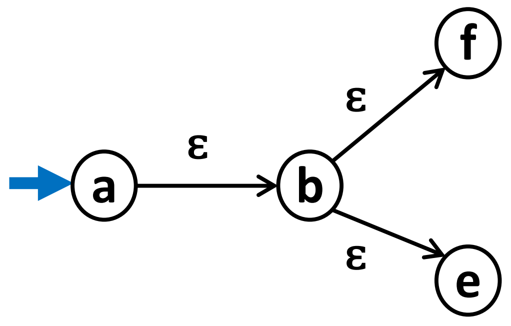
    (Note: not all *paths*; all *last states* on those paths.)

  - There will be one state in the DFA for each *subset* of states of the NFA that can be reached by some string

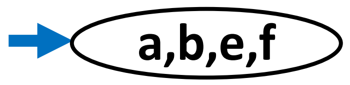
Conversion of NFAs to a DFAs

New start state for DFA

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



NFA

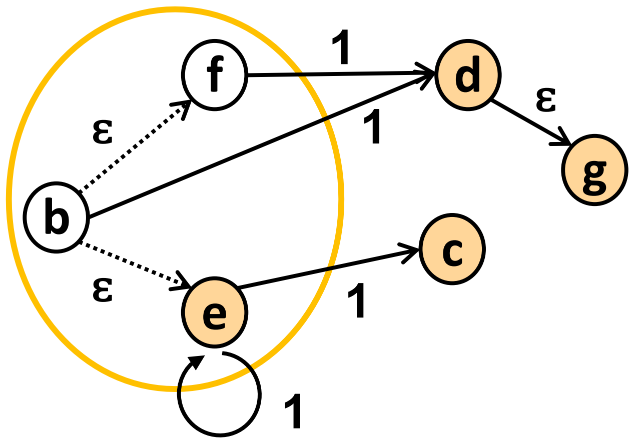


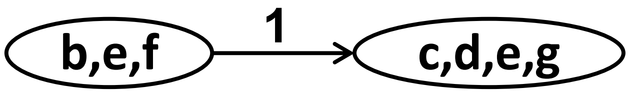
DFA

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

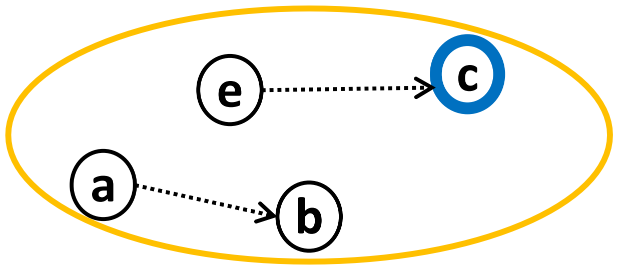




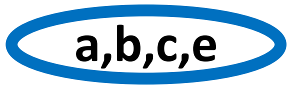
Conversion of NFAs to a DFAs

Final states for the DFA

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

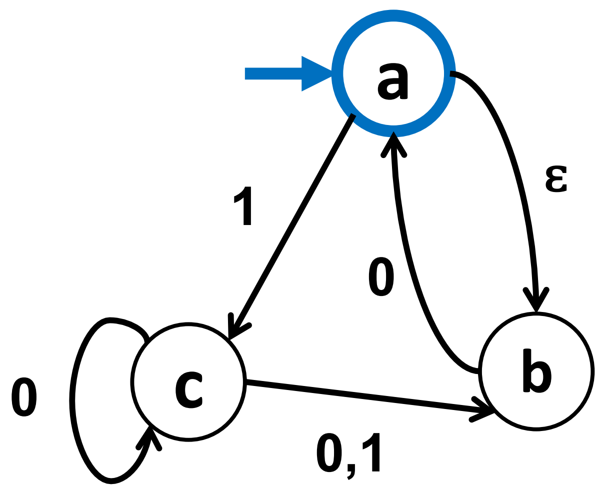


NFA

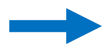


DFA

Example: NFA to DFA

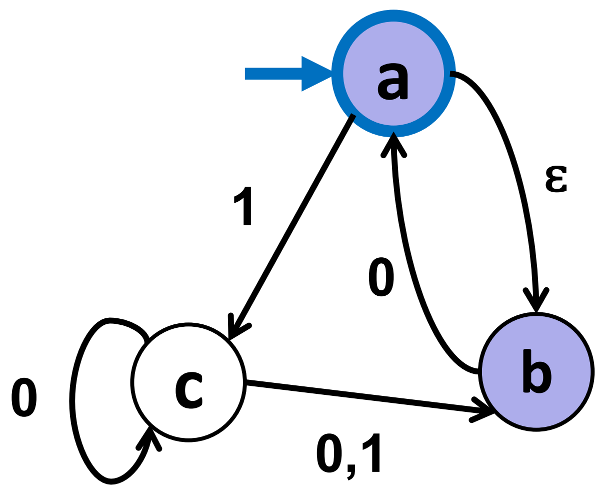


NFA

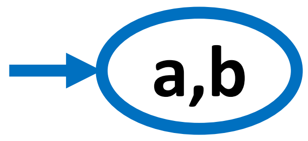


DFA

Example: NFA to DFA

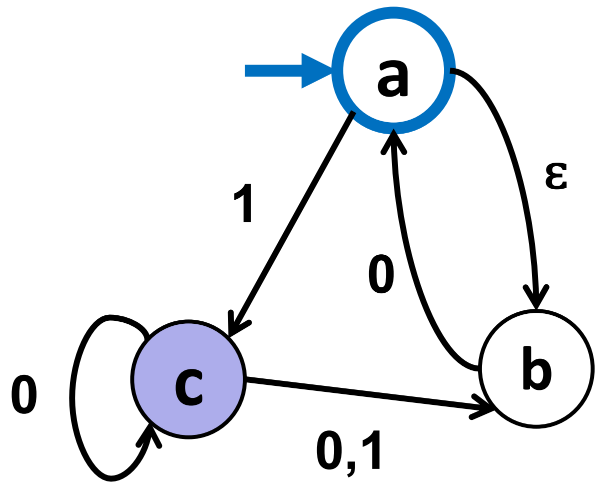


NFA

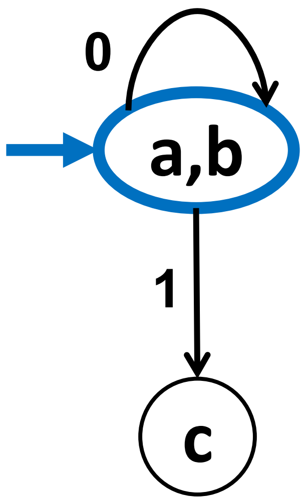


DFA

Example: NFA to DFA

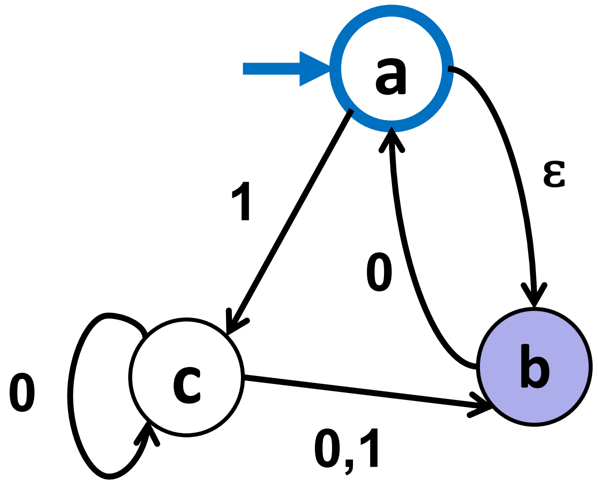


NFA

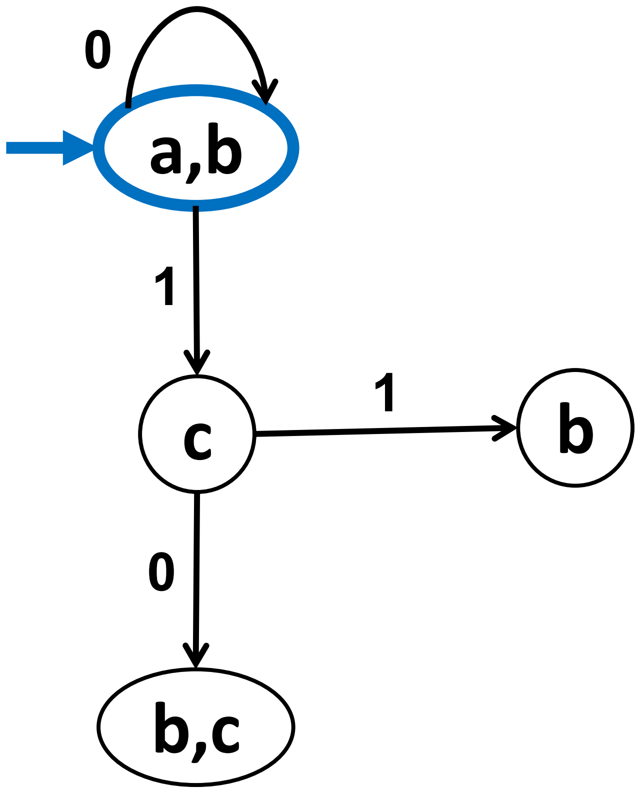


DFA

Example: NFA to DFA

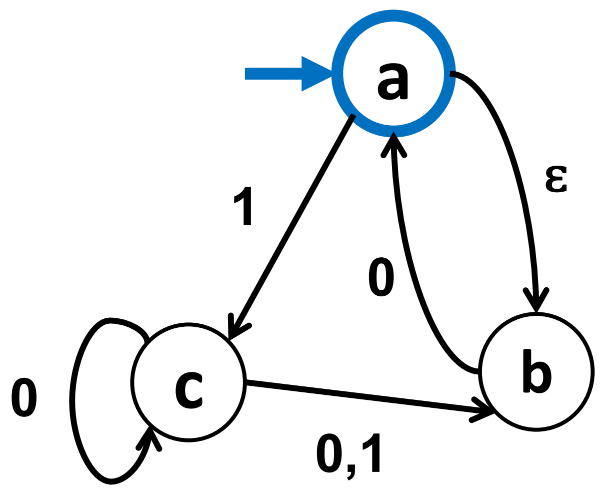


NFA

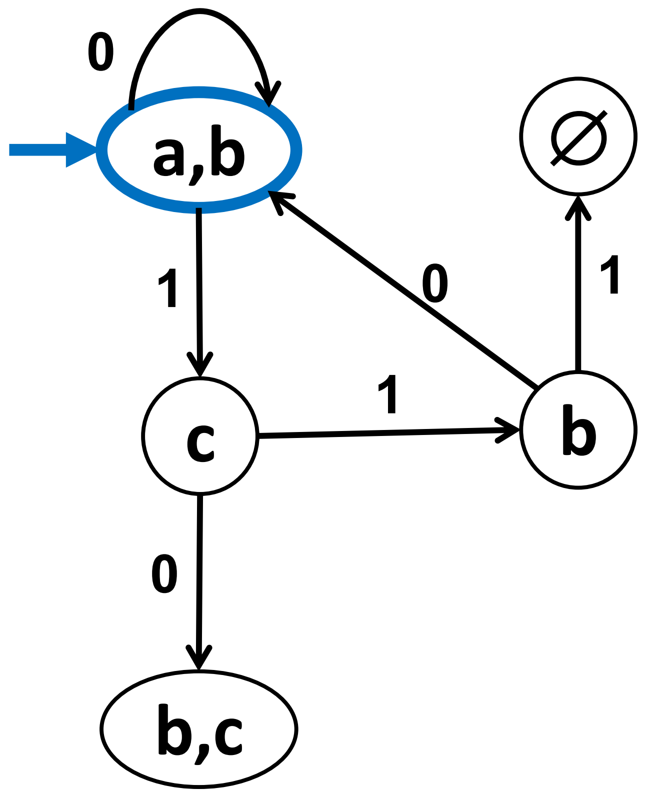


DFA

Example: NFA to DFA

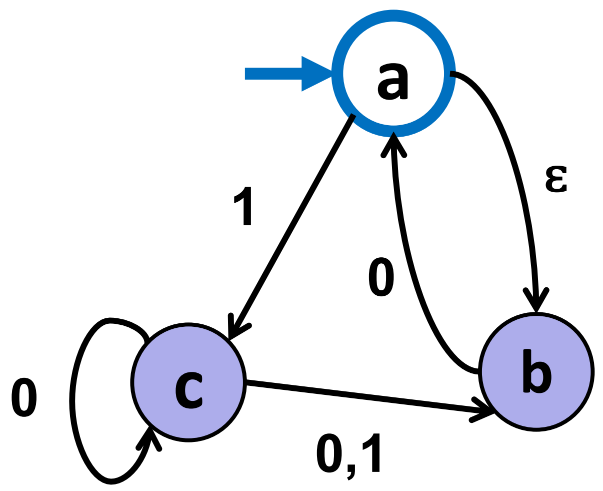


NFA

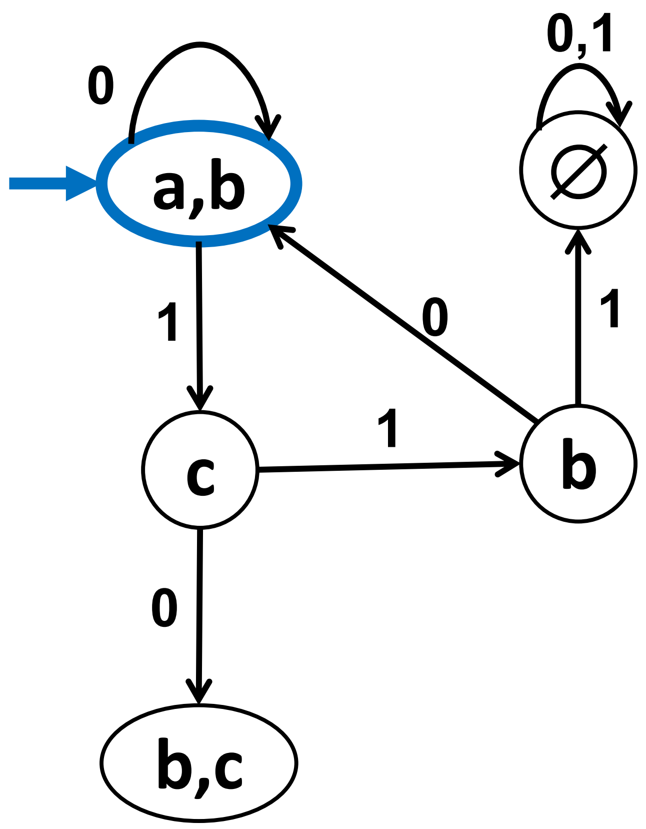


DFA

Example: NFA to DFA

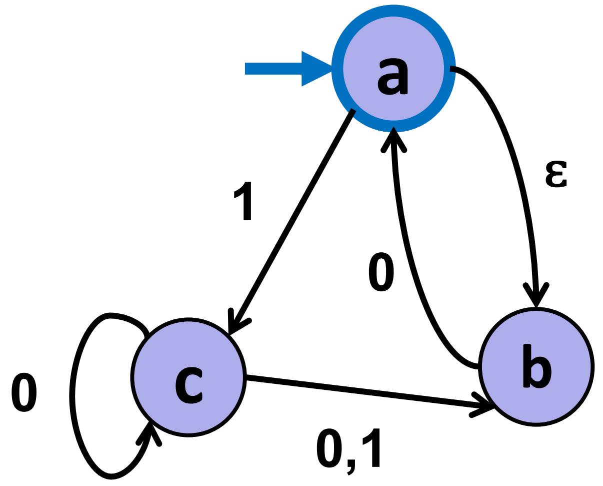


NFA

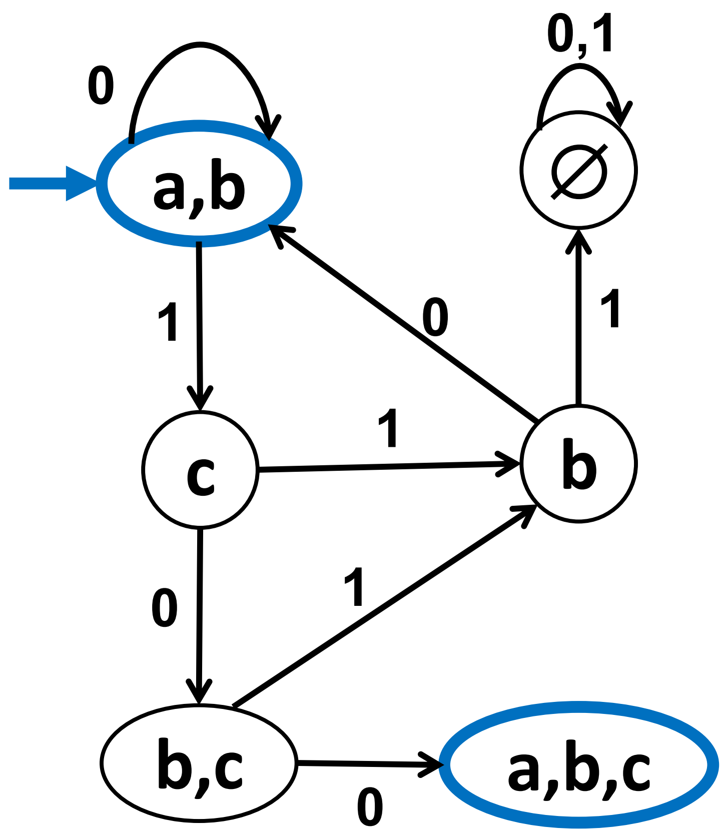


DFA

Example: NFA to DFA

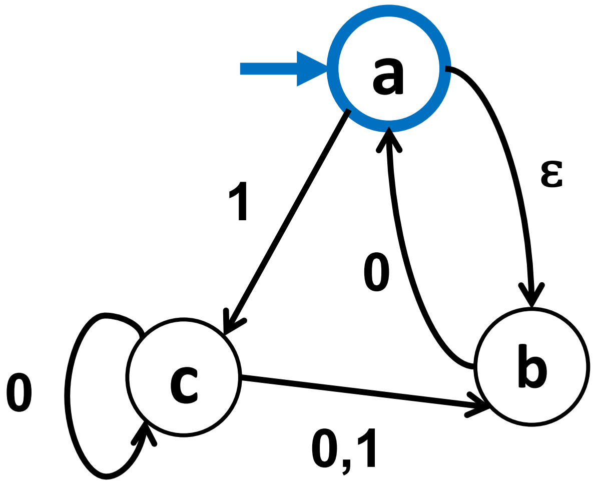


NFA

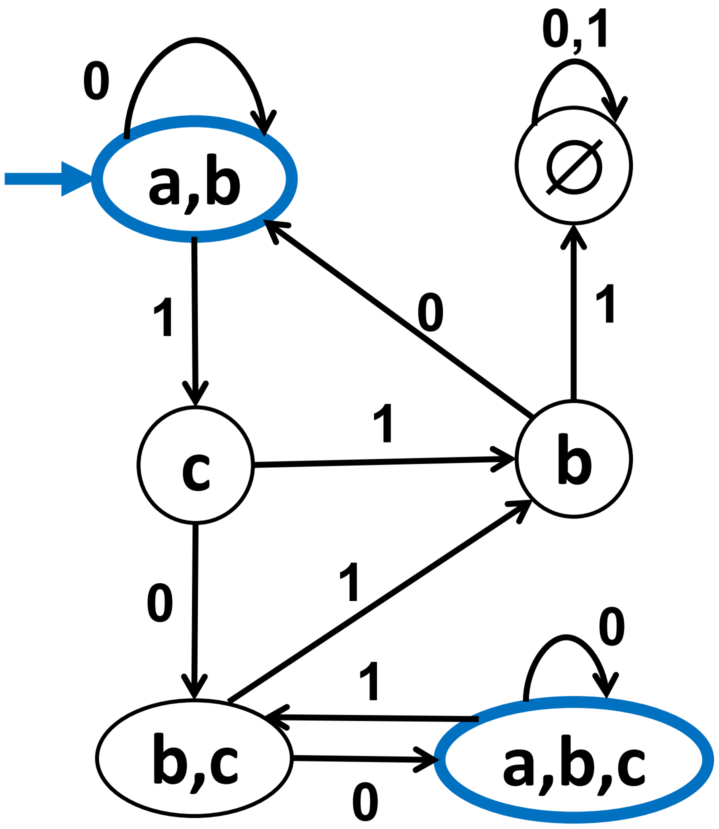


DFA

Example: NFA to DFA



NFA



DFA

The story so far...

$$\begin{tikzcd}
\boxed{\text{REs}} \arrow[r, phantom, "\subseteq"] & \boxed{\text{CFGs}} \\
\boxed{\text{DFAs}} \arrow[r, phantom, "="] \arrow[ur, phantom, "\subseteq" sloped] & \boxed{\text{NFAs}}
\end{tikzcd}$$

Regular expressions $\subseteq$ NFAs $\equiv$ DFAs

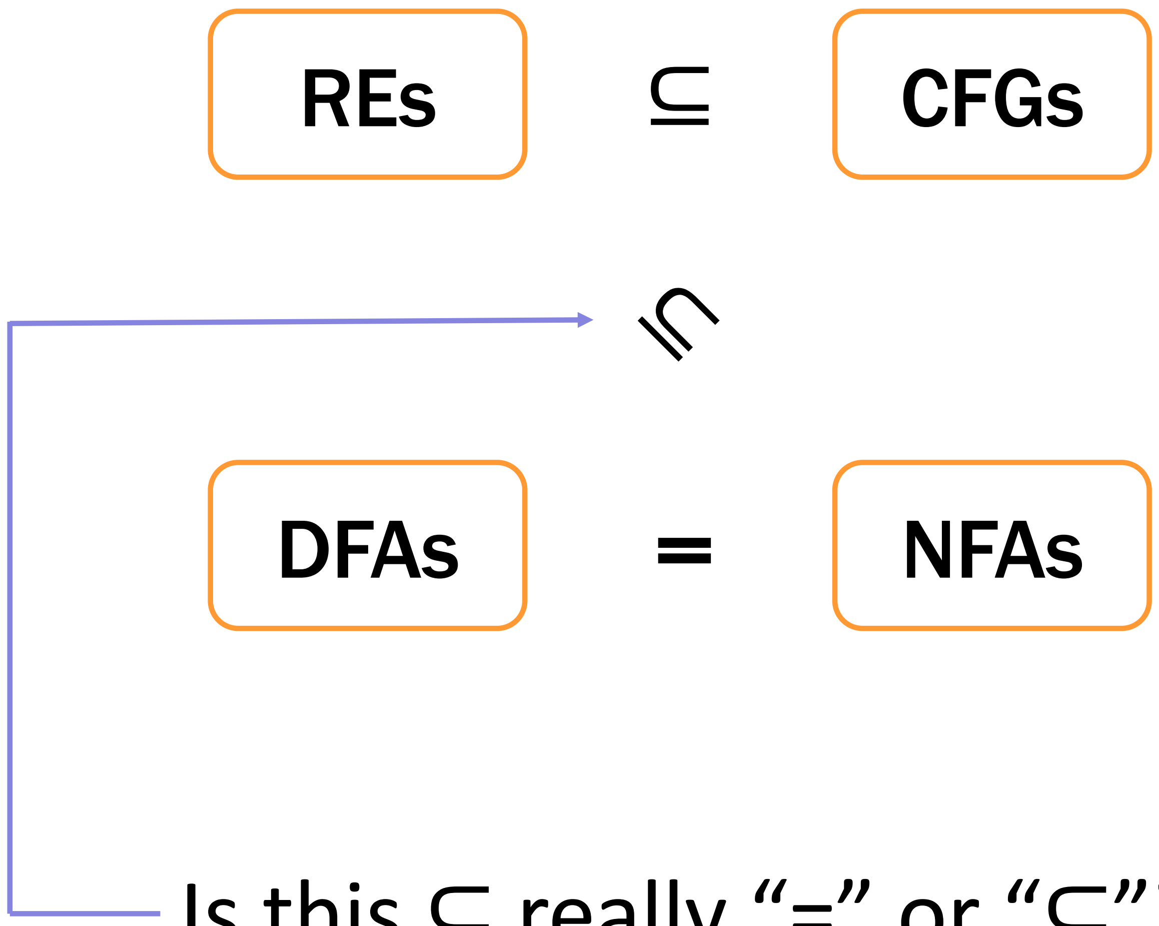
We have shown how to build an optimal DFA for every regular expression

- Build NFA
- Convert NFA to DFA using subset construction
- Minimize resulting DFA

Thus, we could now implement a RegExp library

- most RegExp libraries actually simulate the NFA
- (even better: one can combine the two approaches:
  apply DFA minimization lazily while simulating the NFA)

The story so far…



Is this ⊆ really “=” or “⊊”?

Regular expressions ≡ NFAs ≡ DFAs

Theorem: For any NFA, there is a regular expression that accepts the same language

Corollary: A language is recognized by a DFA (or NFA) if and only if it has a regular expression

You need to know these facts

– the construction for the Theorem is included in the slides after this, but you will not be tested on it

The story so far...

$$\begin{tikzcd}
\boxed{\text{REs}} \arrow[r, phantom, "\subseteq"] \arrow[d, phantom, "\|"] & \boxed{\text{CFGs}} \\
\boxed{\text{DFAs}} \arrow[r, phantom, "="] & \boxed{\text{NFAs}}
\end{tikzcd}$$

The story so far…

$$\begin{tikzcd}
\boxed{\text{REs}} \arrow[r, phantom, "\subseteq"] \arrow[d, phantom, "\parallel"] & \boxed{\text{CFGs}} \\
\boxed{\text{DFAs}} \arrow[r, phantom, "="] & \boxed{\text{NFAs}}
\end{tikzcd}$$

<u>Next time</u>: Is this ⊆ really "=" or "⊊"?