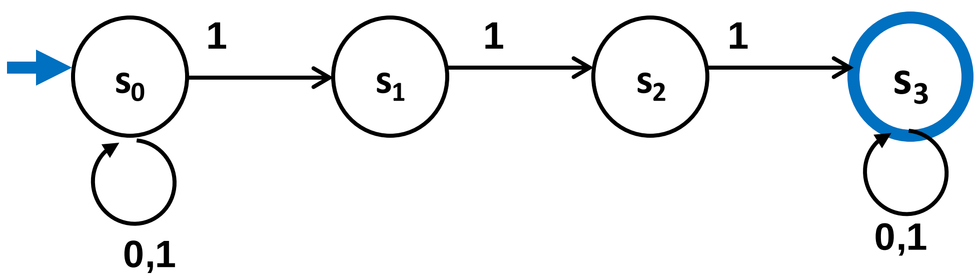
CSE 311: Foundations of Computing

Lecture 25: Equivalence RE and NFAs

Recap: Nondeterministic Finite Automata (NFA)

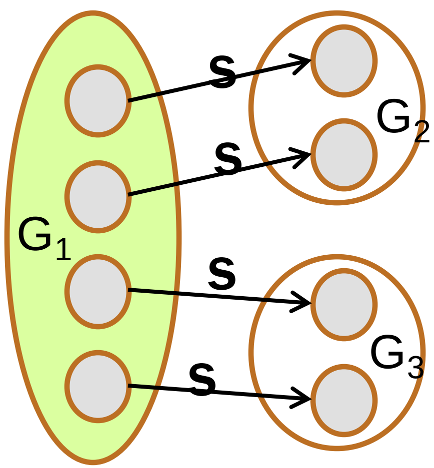
- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol— can have 0 or >1
  - Also can have edges labeled by empty string ε
- **Definition:** x is in the language recognized by an NFA if and only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

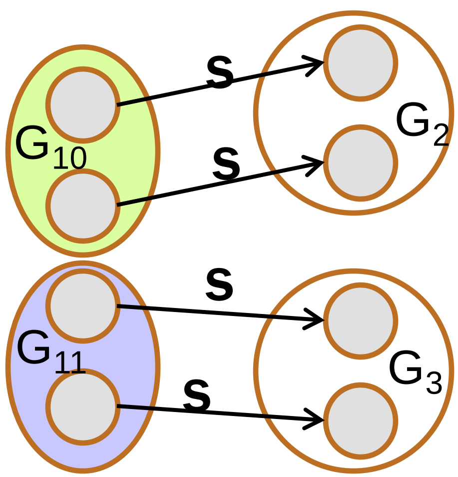


Recognized language: $(0 \cup 1)^*111(0 \cup 1)^*$ as RE

Recap: 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

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.**

Three ways of thinking about NFAs

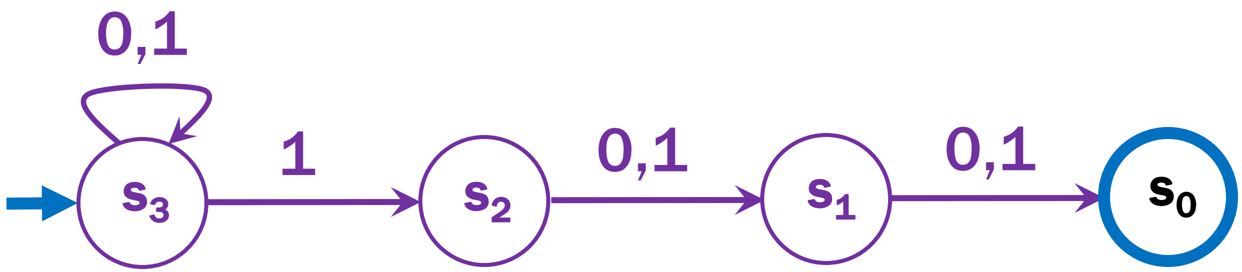
- Outside observer: Is there a path labeled by x from the start state to some final state?

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

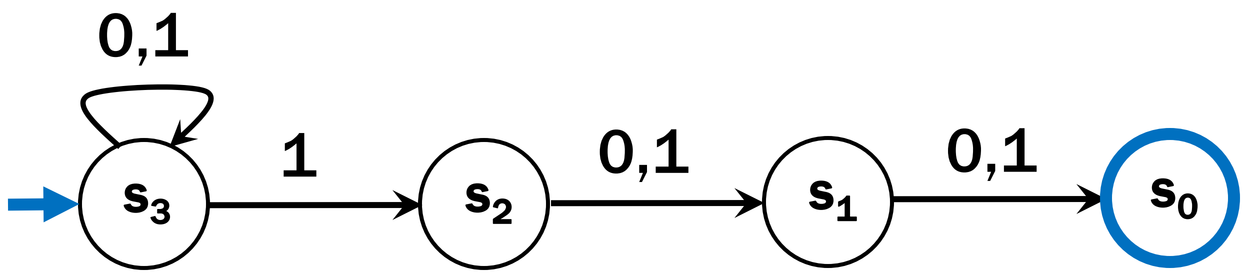
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)

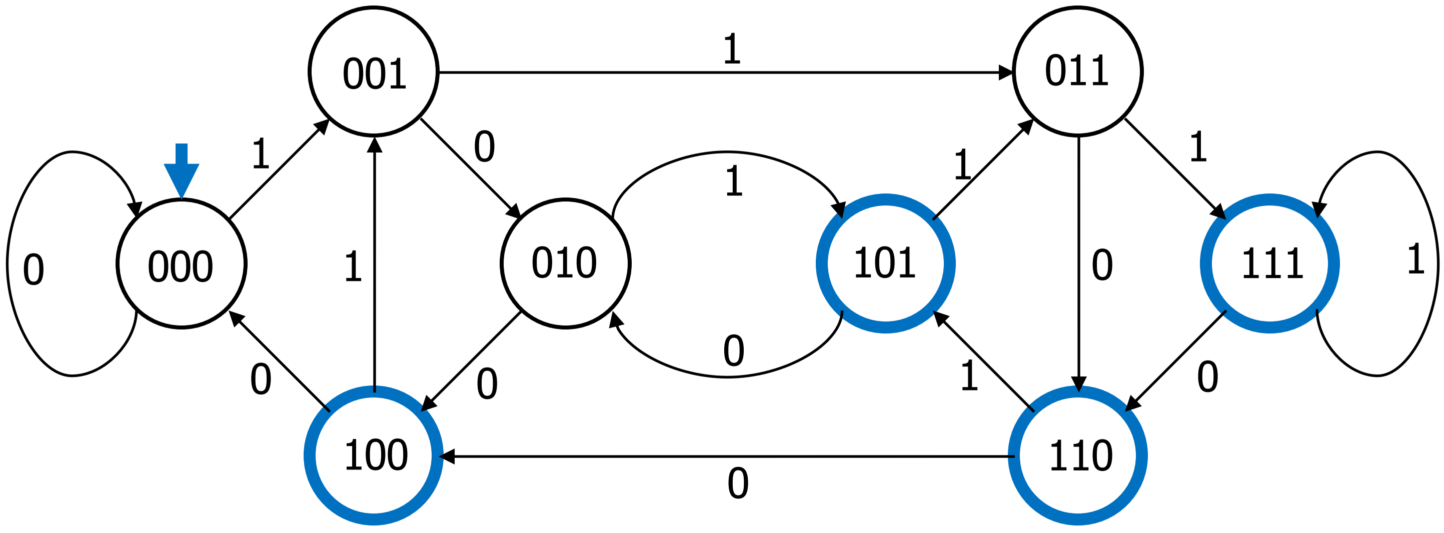
NFA for set of binary strings with a 1 in the 3rd position from the end

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

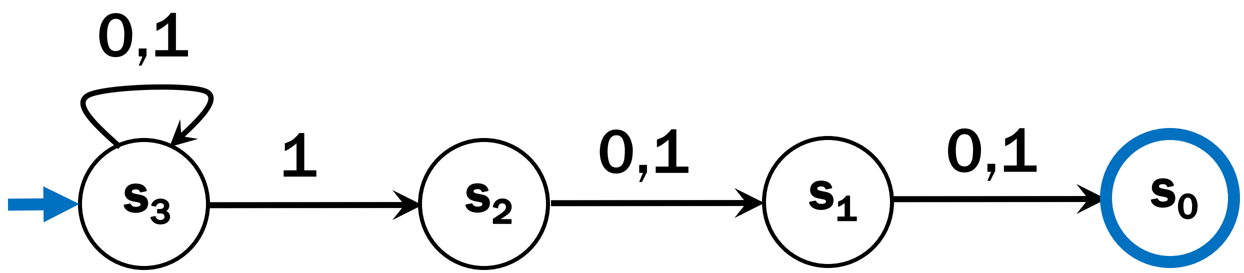


Compare with the smallest DFA

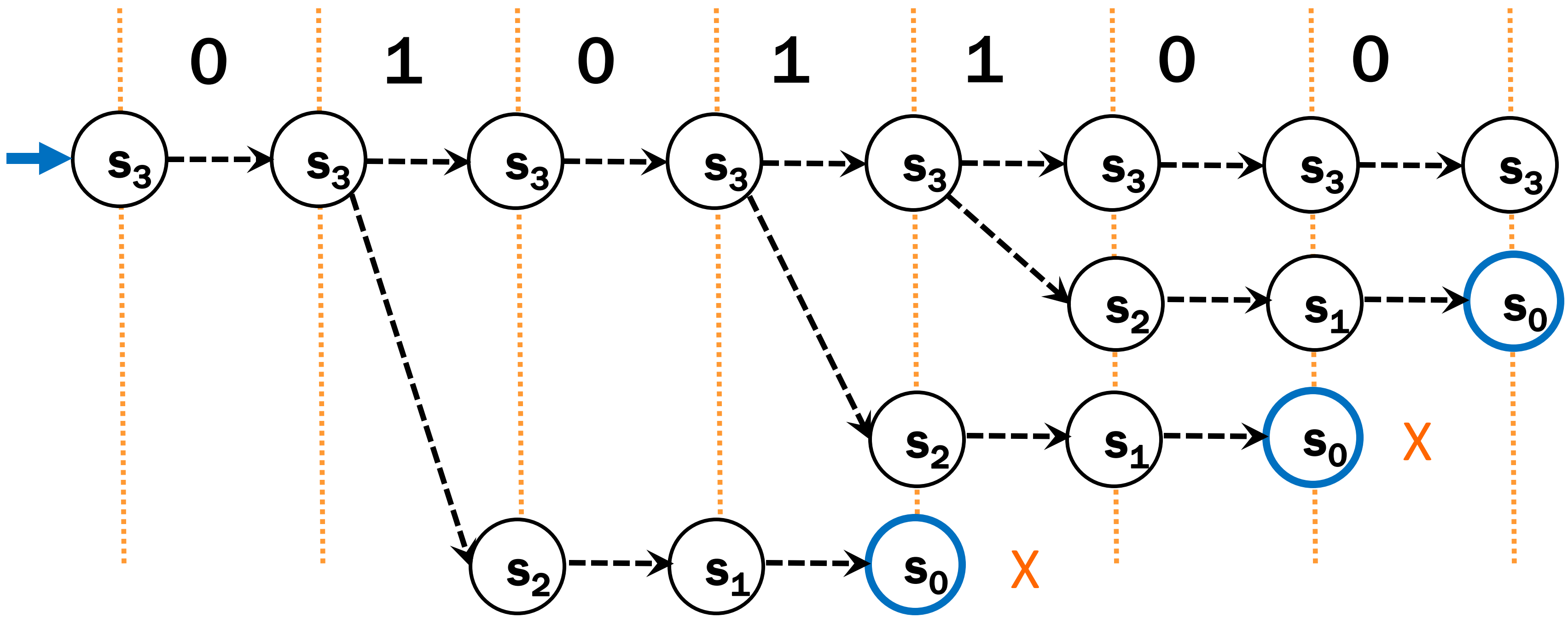




Parallel Exploration view of an NFA



Input string 0101100



Lecture 25 Activity

You will be assigned to breakout rooms. Please:

- Introduce yourself
- Choose someone to share their screen, showing this PDF
- How do you think NFAs relate to DFAs, regular expressions, and CFGs? (in terms of power to describe languages)

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/philipmg and login with your UW identity

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

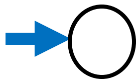
- Case ∅:

- Case ε:

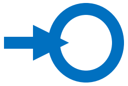
- Case *a*:

Base Case

- Case ∅:



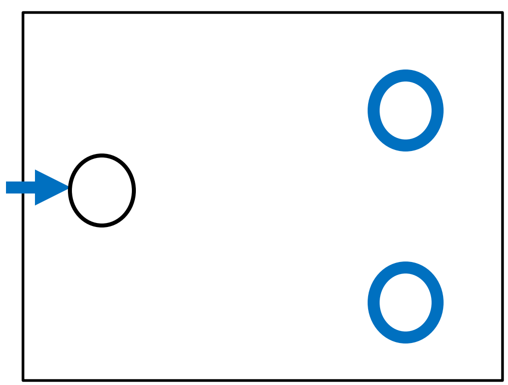
- Case ε:



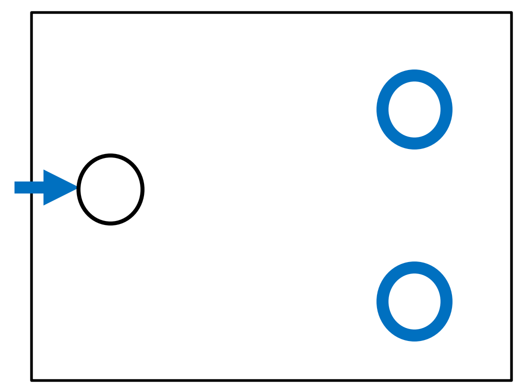
- Case *a*:

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



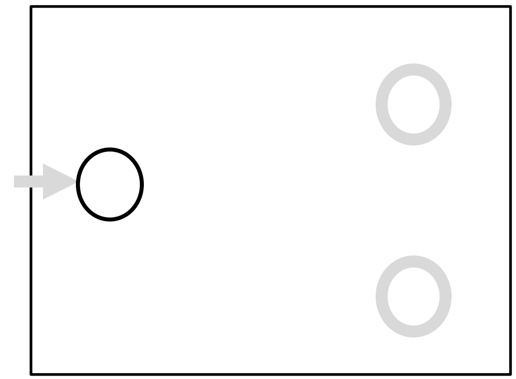
N_A



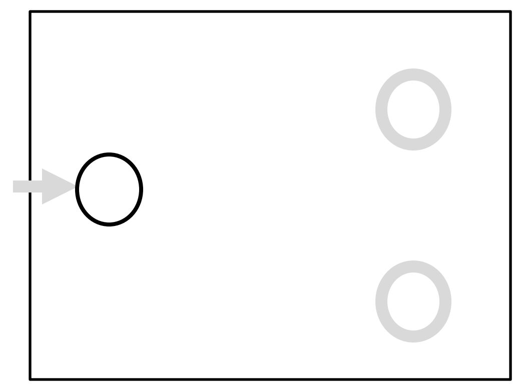
N_B

Inductive Step

Case (**A** ∪ **B**):



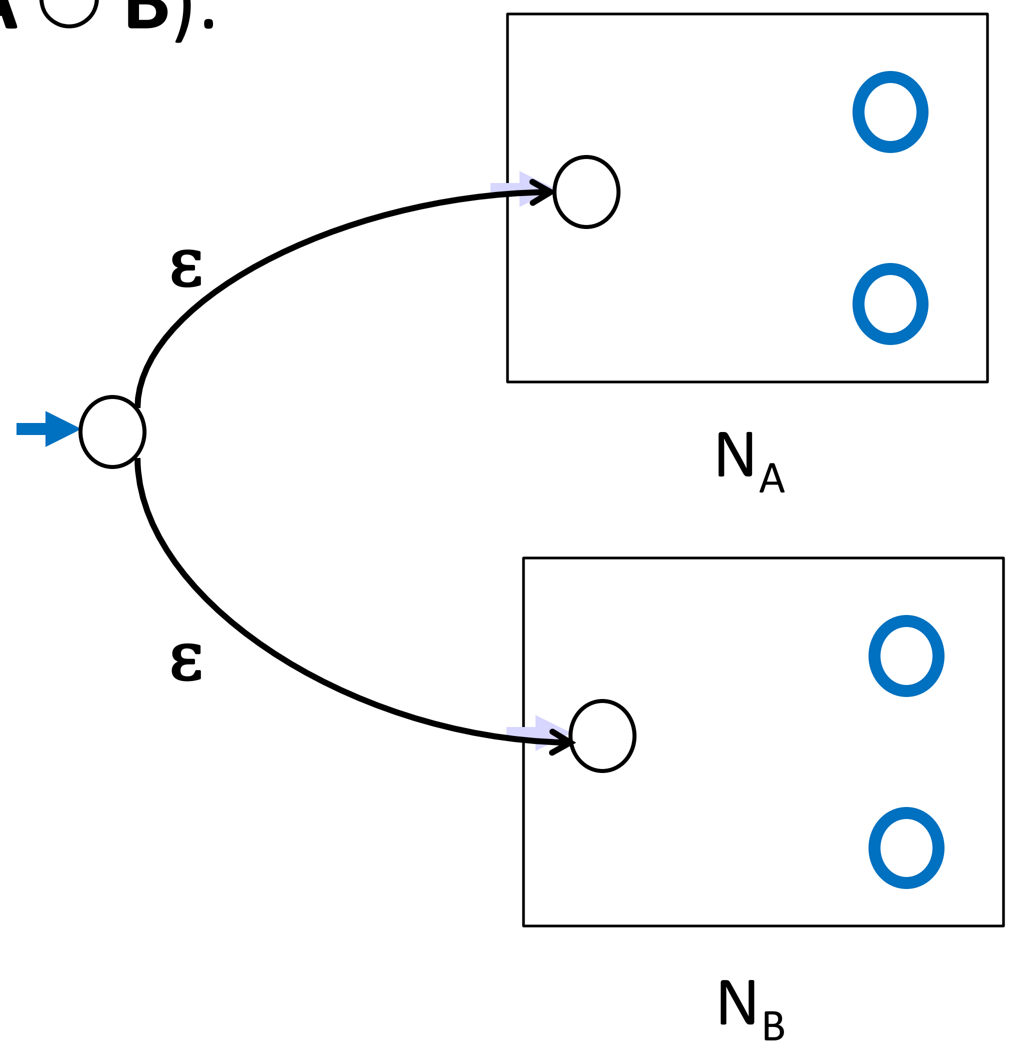
N_A



N_B

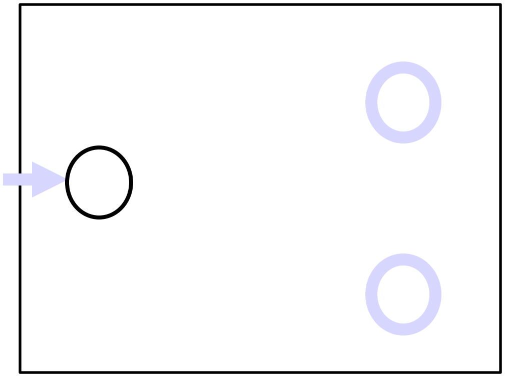
Inductive Step

Case ($A \cup B$):

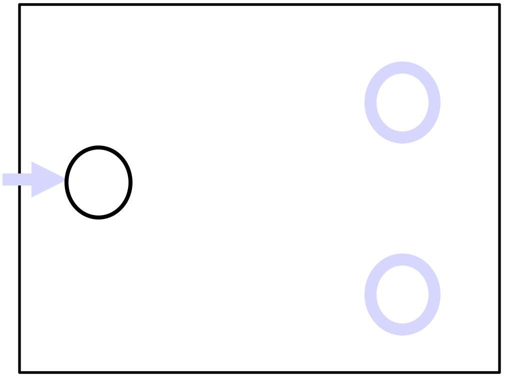


Inductive Step

Case (AB):



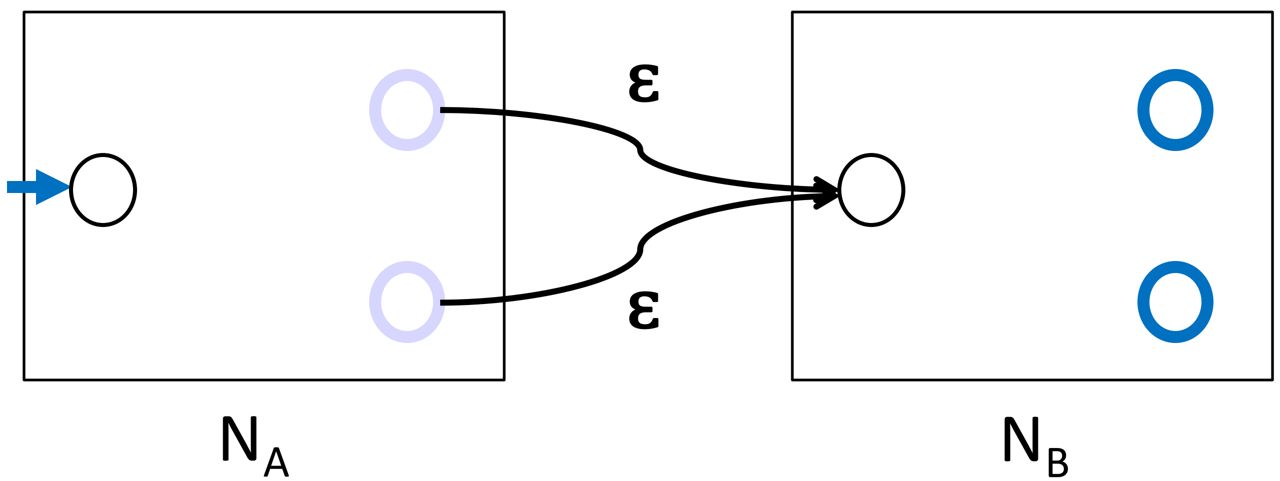
N_A



N_B

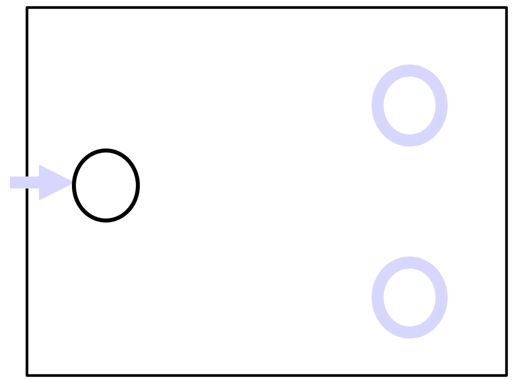
Inductive Step

Case (AB):



Inductive Step

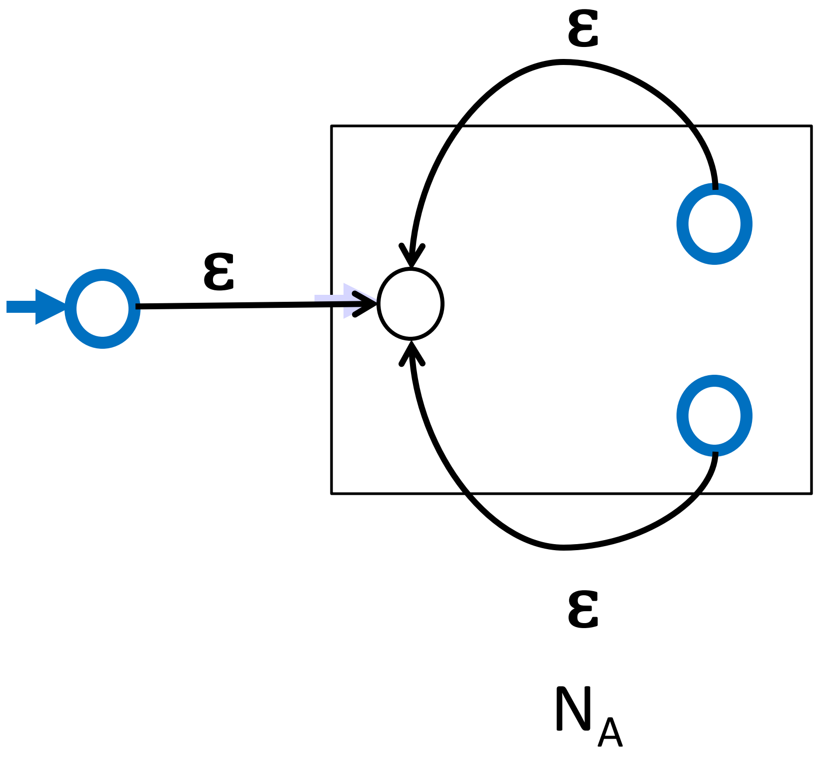
Case A*



N_A

Inductive Step

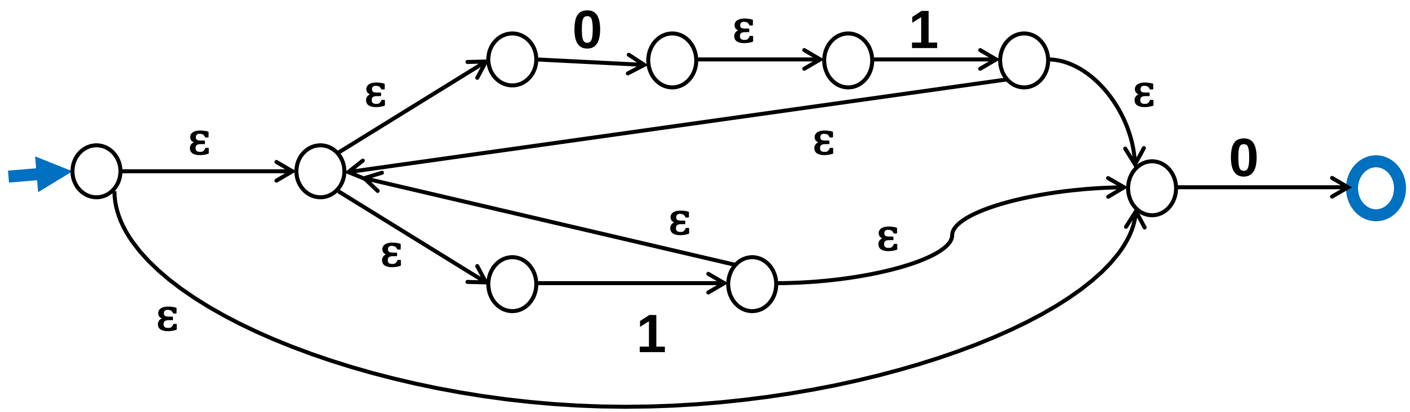
Case A*



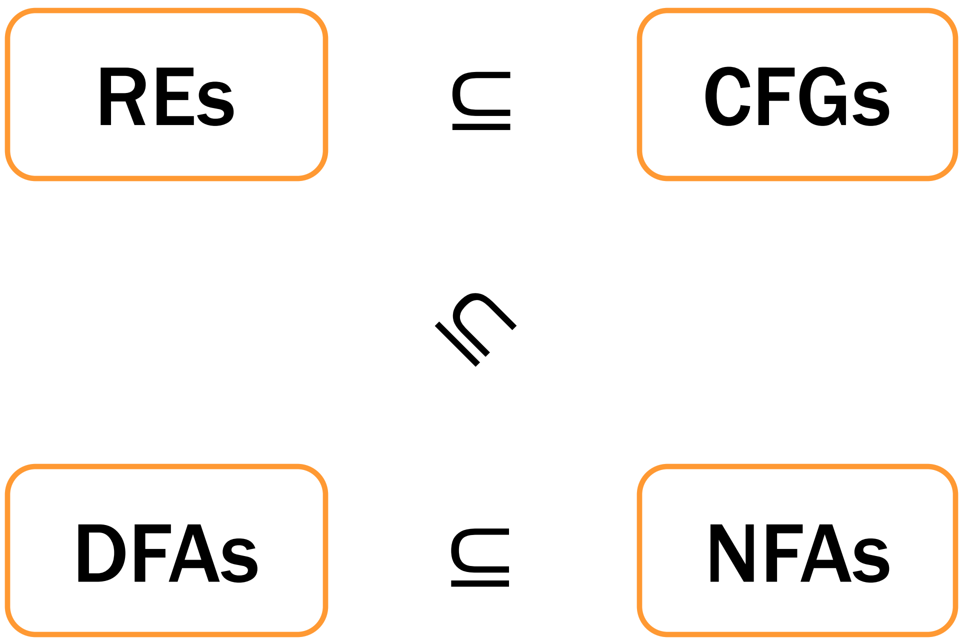
Build an NFA for $(01 \cup 1)^*0$

Solution

(01 ∪1)*0



The story so far…



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

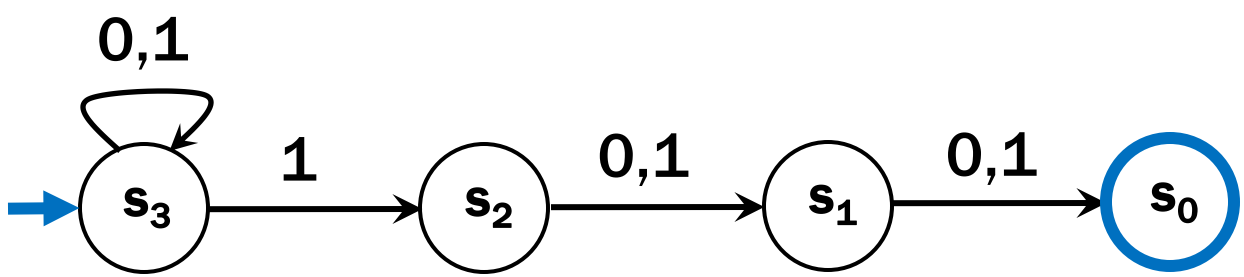
Three ways of thinking about NFAs

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

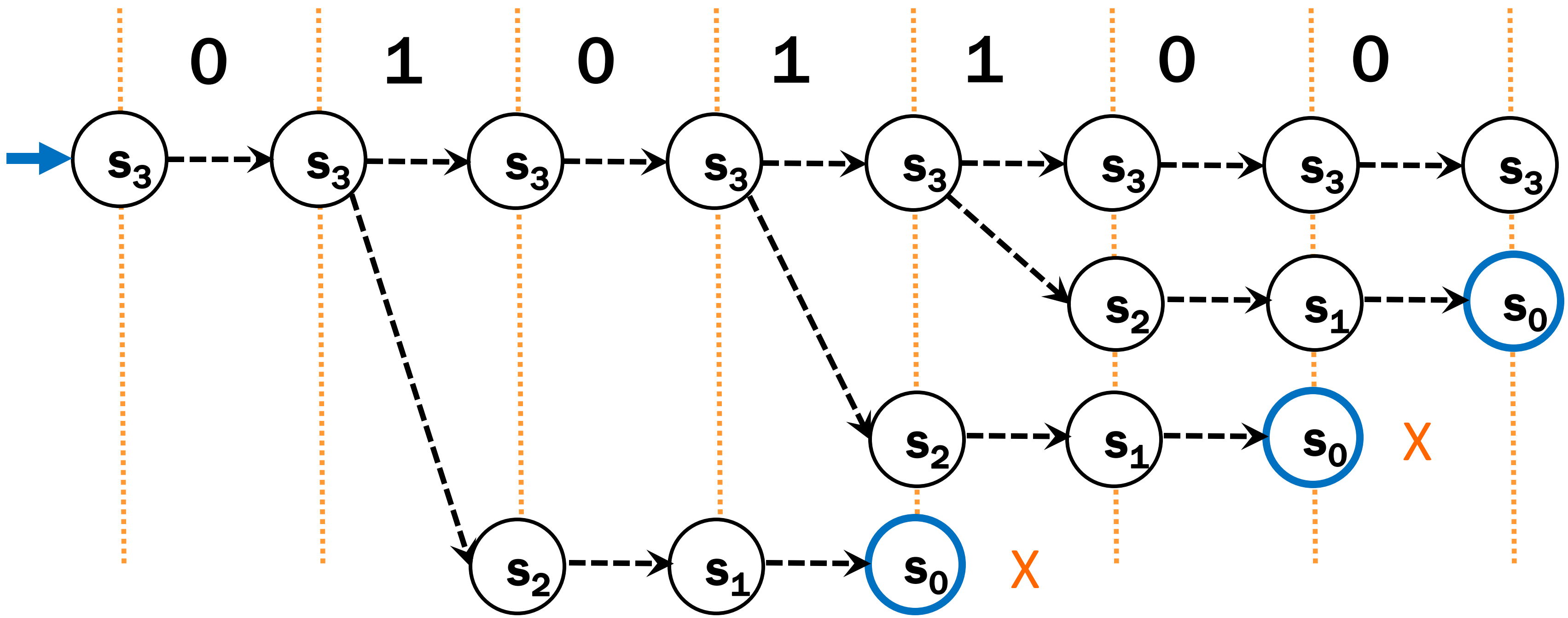
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)

- 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



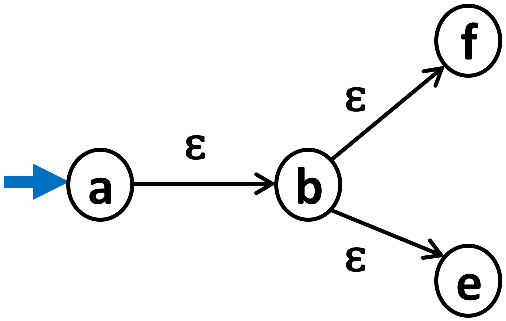
Conversion of NFAs to a DFAs

- 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

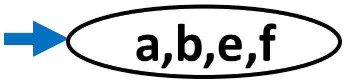
Conversion of NFAs to a DFAs

New start state for DFA

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



NFA

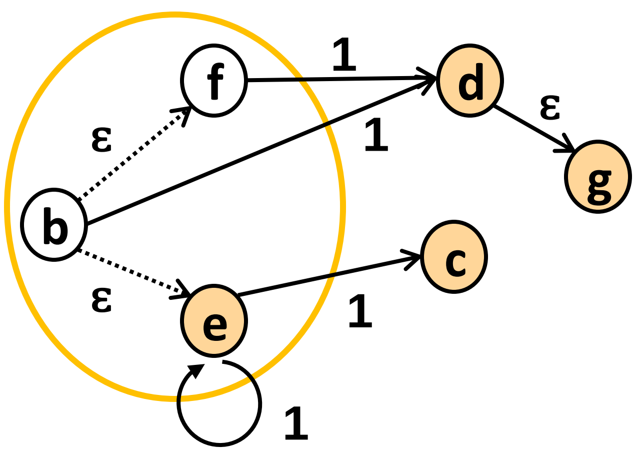


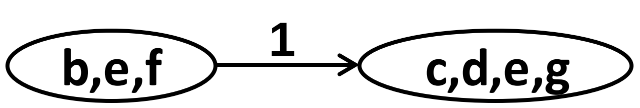
DFA

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**

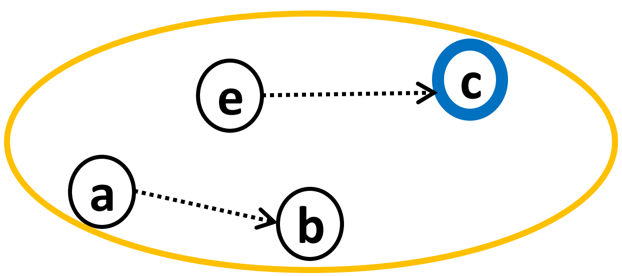




Conversion of NFAs to a DFAs

Final states for the DFA

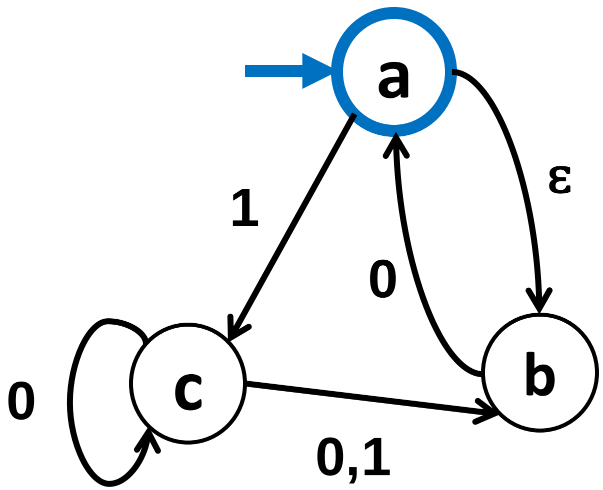
- All states whose set contain some final state of the NFA



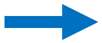
NFA

DFA

Example: NFA to DFA

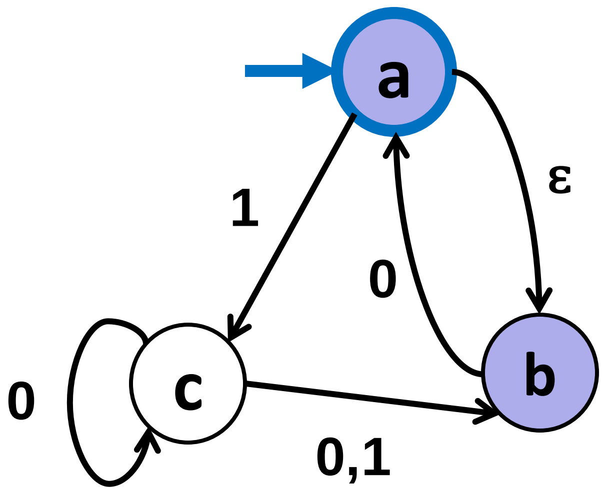


NFA

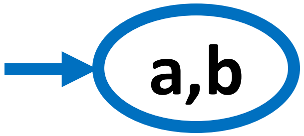


DFA

Example: NFA to DFA

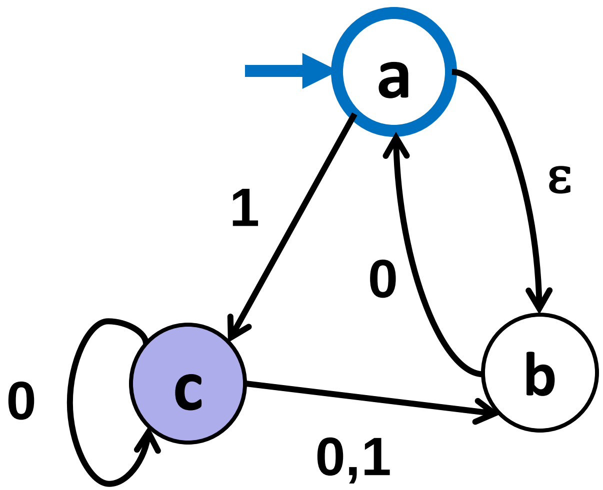


NFA

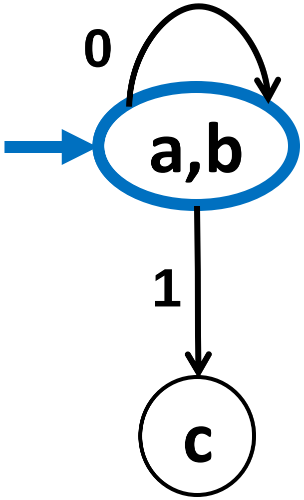


DFA

Example: NFA to DFA

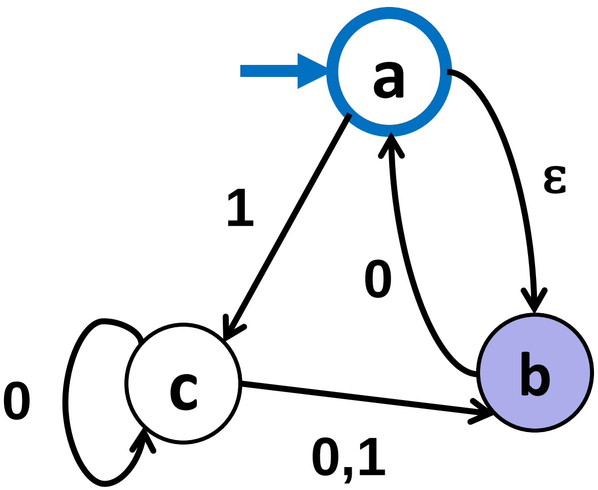


NFA

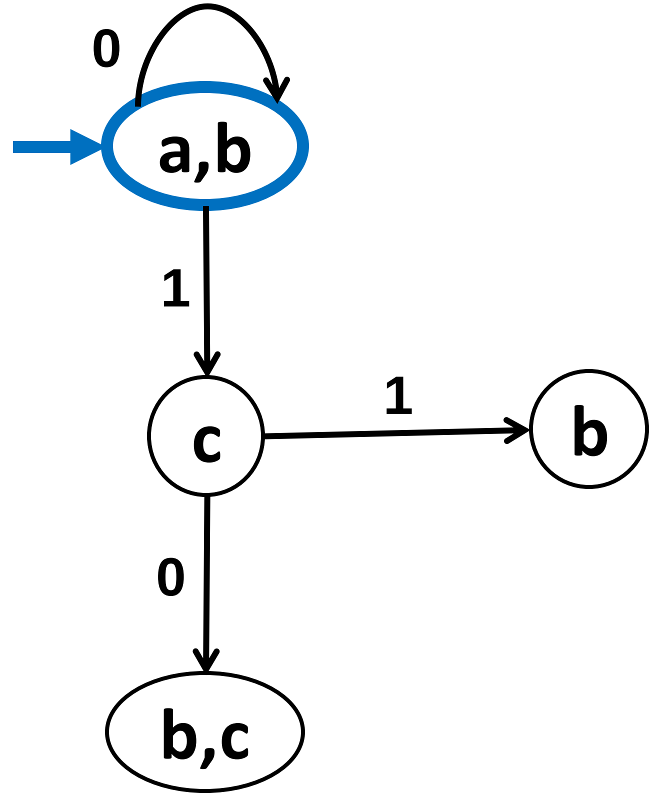


DFA

Example: NFA to DFA

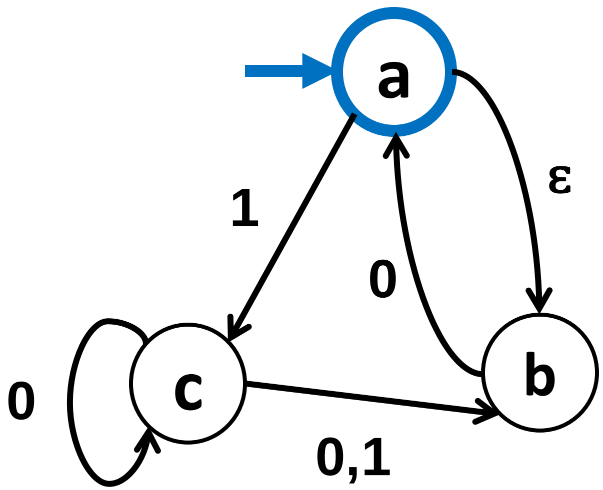


NFA

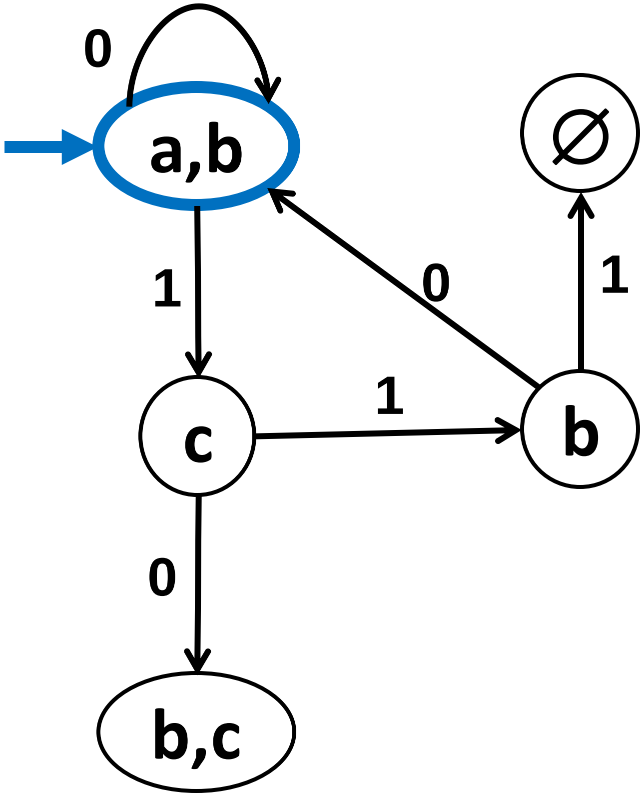


DFA

Example: NFA to DFA

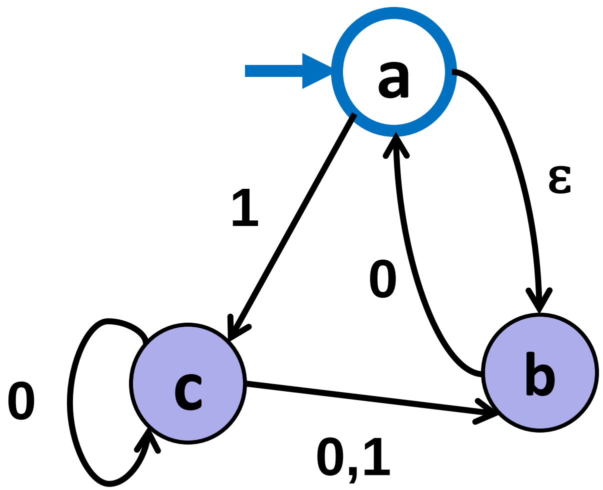


NFA

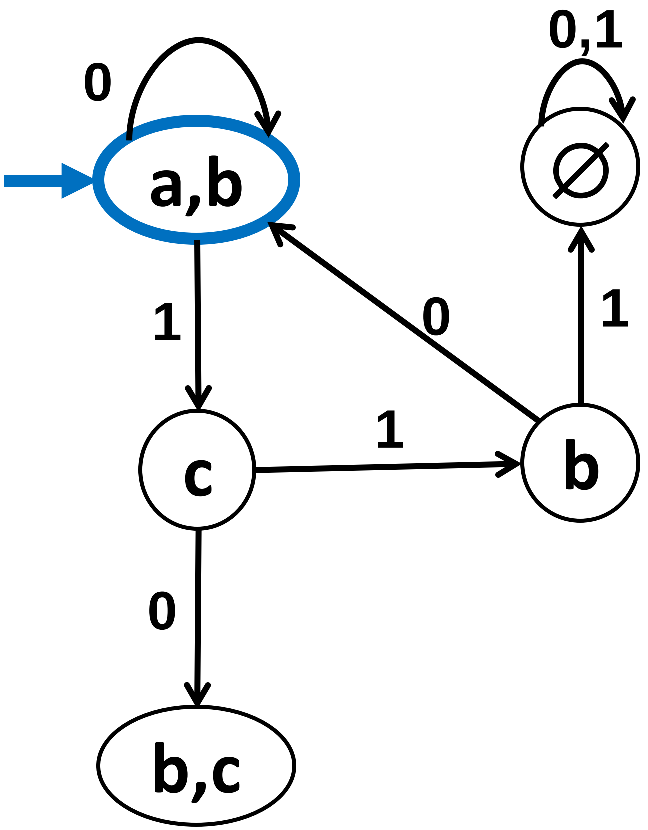


DFA

Example: NFA to DFA

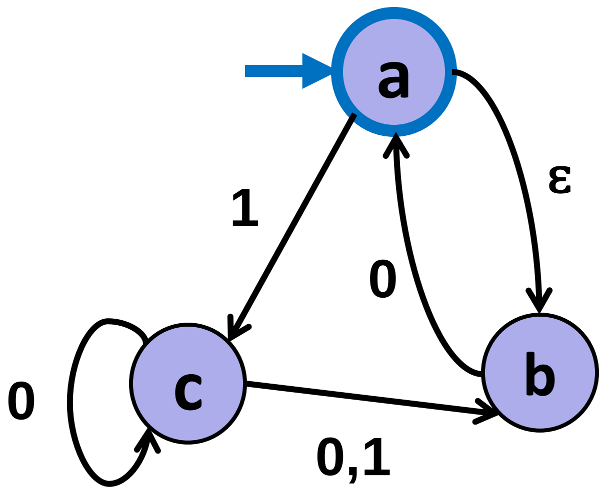


NFA

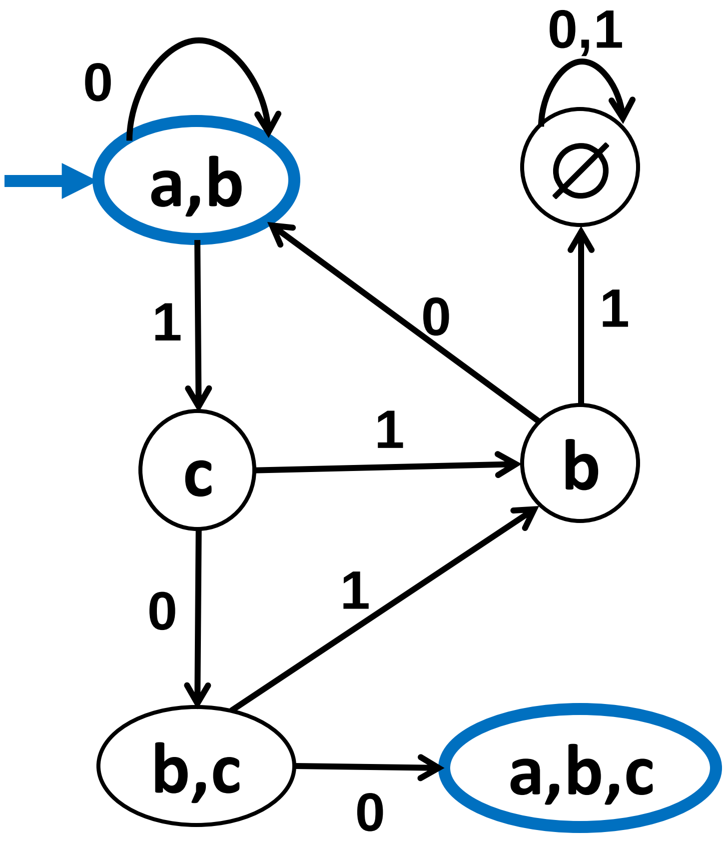


DFA

Example: NFA to DFA

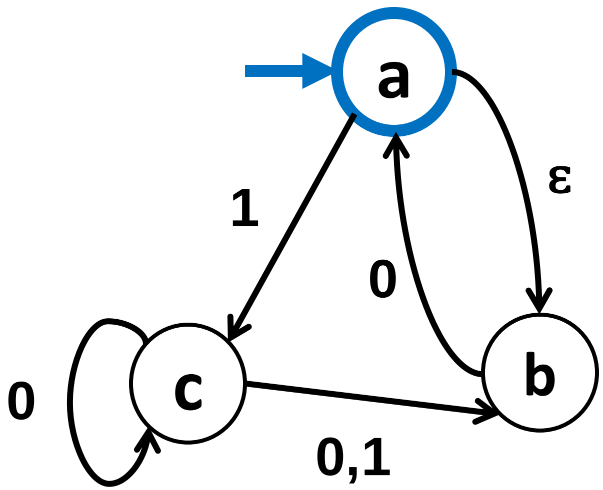


NFA

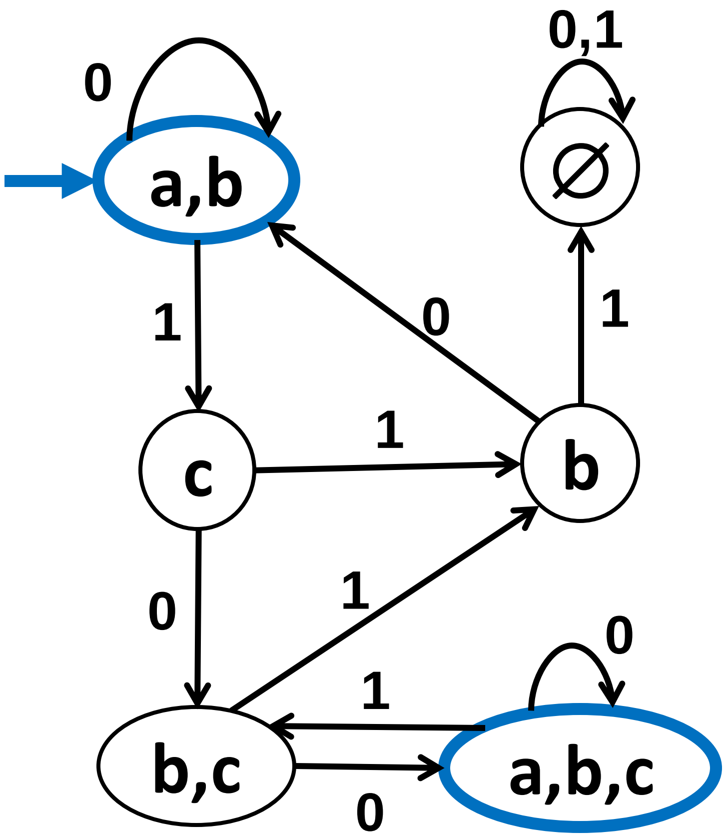


DFA

Example: NFA to DFA



NFA



DFA

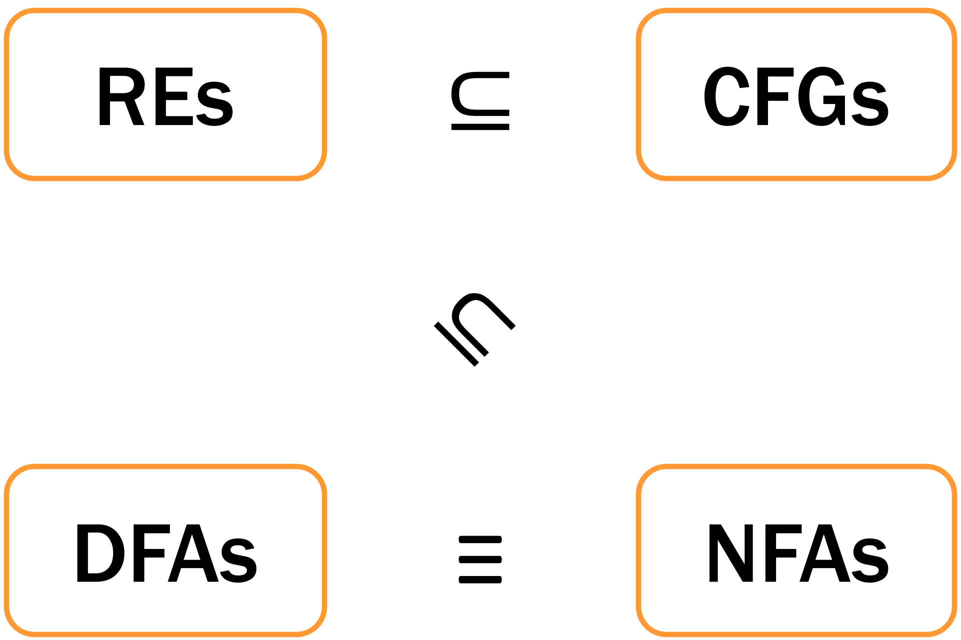
Exponential Blow-up in Simulating Nondeterminism

- 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

The story so far...



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

Regular expressions ≡ NFAs ≡ 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

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

You need to know this fact but we won't ask you anything about the "only if" direction from DFA/NFA to regular expression. For fun, we sketch the idea.

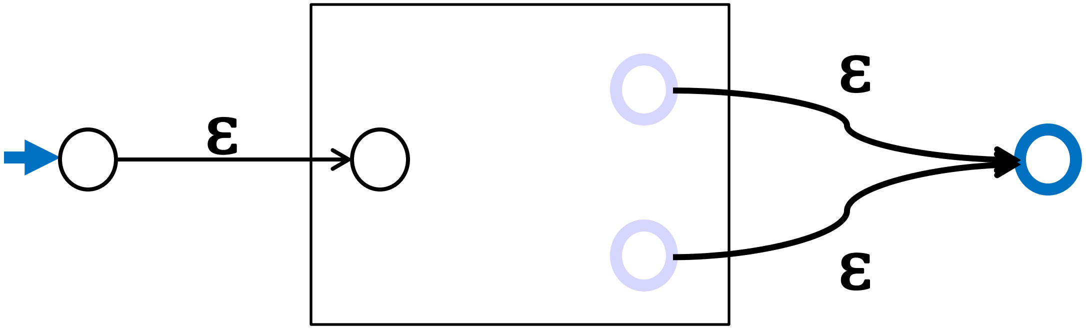
Generalized NFAs

- **Like NFAs but allow**
  - **Parallel edges**
  - **Regular Expressions as edge labels**

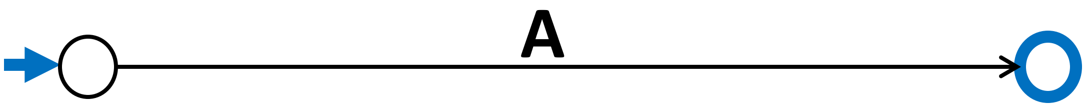
    NFAs already have edges labeled **ε** or ***a***
- **An edge labeled by A can be followed by reading a string of input chars that is in the language represented by A**
- **Defn: A string** x **is accepted iff there is a *path* from start to final state *labeled by a regular expression* whose language contains** x

Starting from an NFA

Add new start state and final state



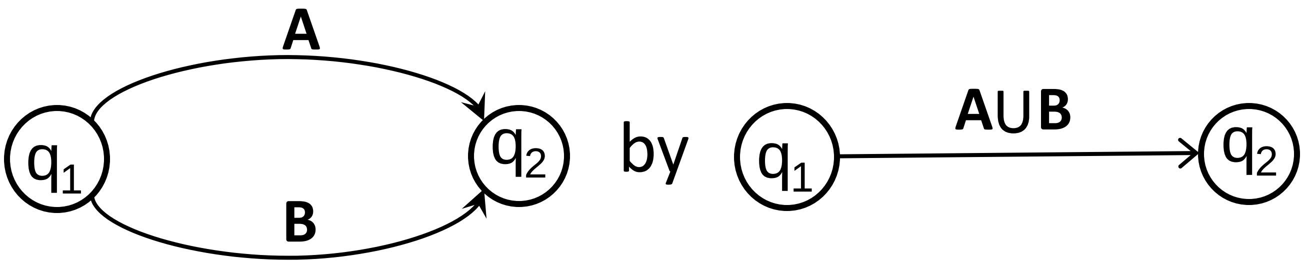
Then eliminate original states one by one, keeping the same language, until it looks like:



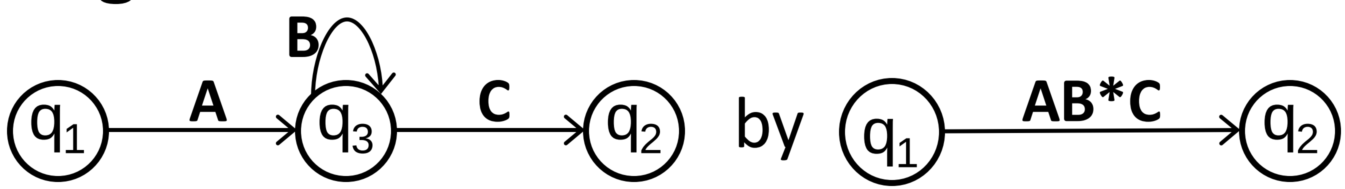
Final regular expression will be **A**

Only two simplification rules

- **Rule 1**: For any two states q_1 and q_2 with parallel edges (possibly $q_1=q_2$), replace



- **Rule 2**: Eliminate non-start/final state q_3 by replacing all

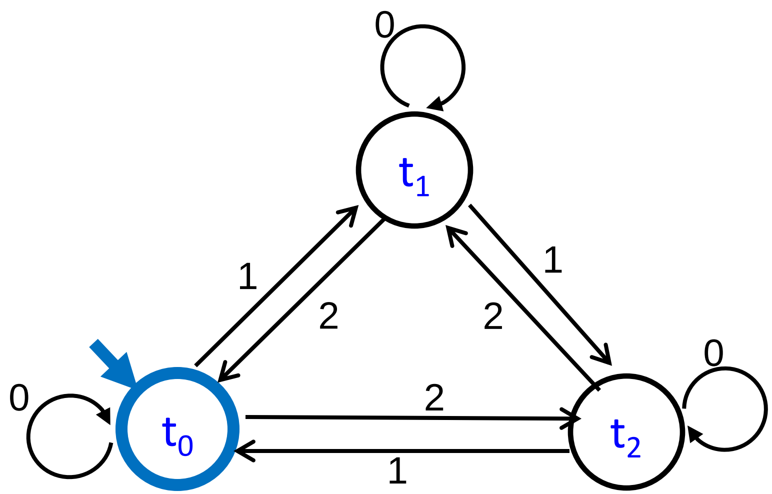


for *every* pair of states q_1, q_2 (even if $q_1=q_2$)

Converting an NFA to a regular expression

Consider the DFA for the mod 3 sum

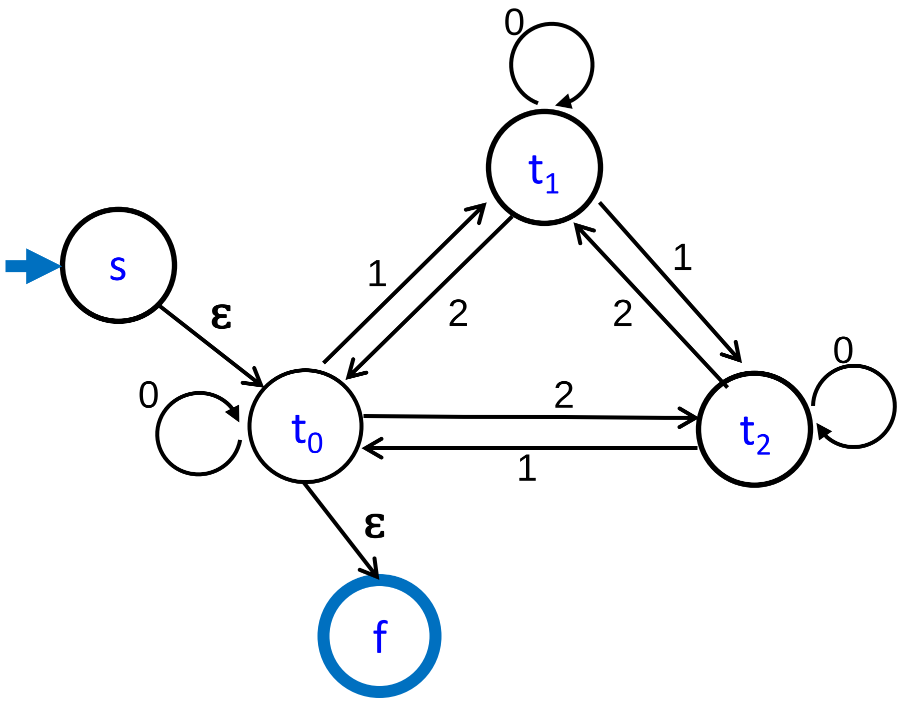
- Accept strings from {0,1,2}* where the digits mod 3 sum of the digits is 0



Splicing out a state t_1

Regular expressions to add to edges

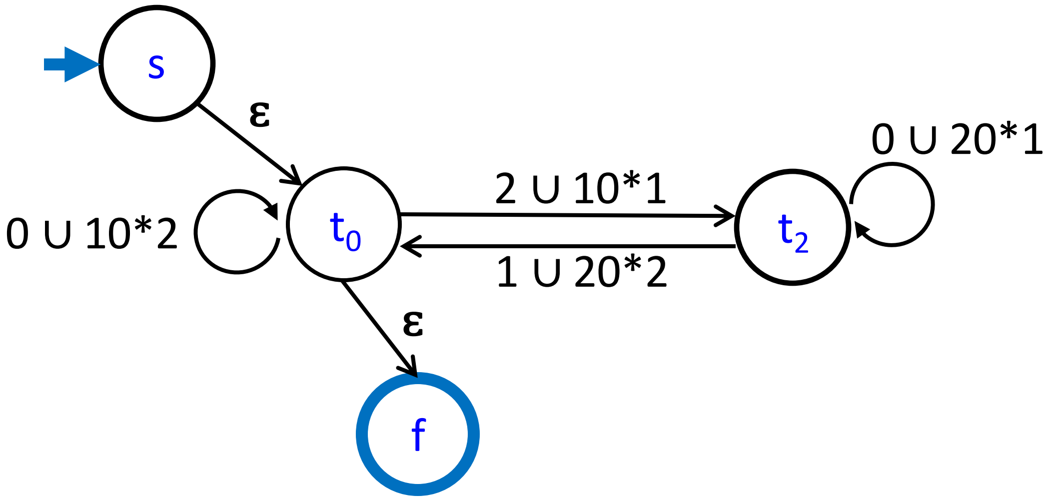
$t_0 \rightarrow t_1 \rightarrow t_0$: 10*2
$t_0 \rightarrow t_1 \rightarrow t_2$: 10*1
$t_2 \rightarrow t_1 \rightarrow t_0$: 20*2
$t_2 \rightarrow t_1 \rightarrow t_2$: 20*1



Splicing out a state t_1

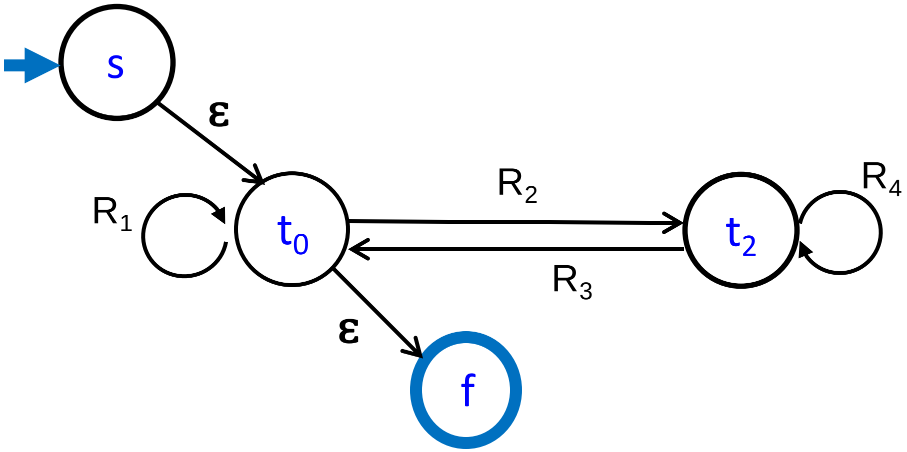
Regular expressions to add to edges

$t_0 \rightarrow t_1 \rightarrow t_0$: 10*2
$t_0 \rightarrow t_1 \rightarrow t_2$: 10*1
$t_2 \rightarrow t_1 \rightarrow t_0$: 20*2
$t_2 \rightarrow t_1 \rightarrow t_2$: 20*1

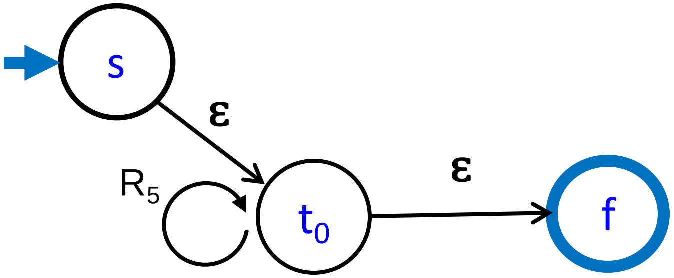


Splicing out state t_2 (and then t_0)

R_1: 0 ∪ 10*2
R_2: 2 ∪ 10*1
R_3: 1 ∪ 20*2
R_4: 0 ∪ 20*1



R_5: $R_1 \cup R_2R_4{}^*R_3$



Final regular expression: $R_5{}^*$=
(0 ∪ 10*2 ∪ (2 ∪ 10*1)(0 ∪ 20*1)*(1 ∪ 20*2))*

The story so far...

REs $\subseteq$ CFGs

$\equiv$

DFAs $\equiv$ NFAs