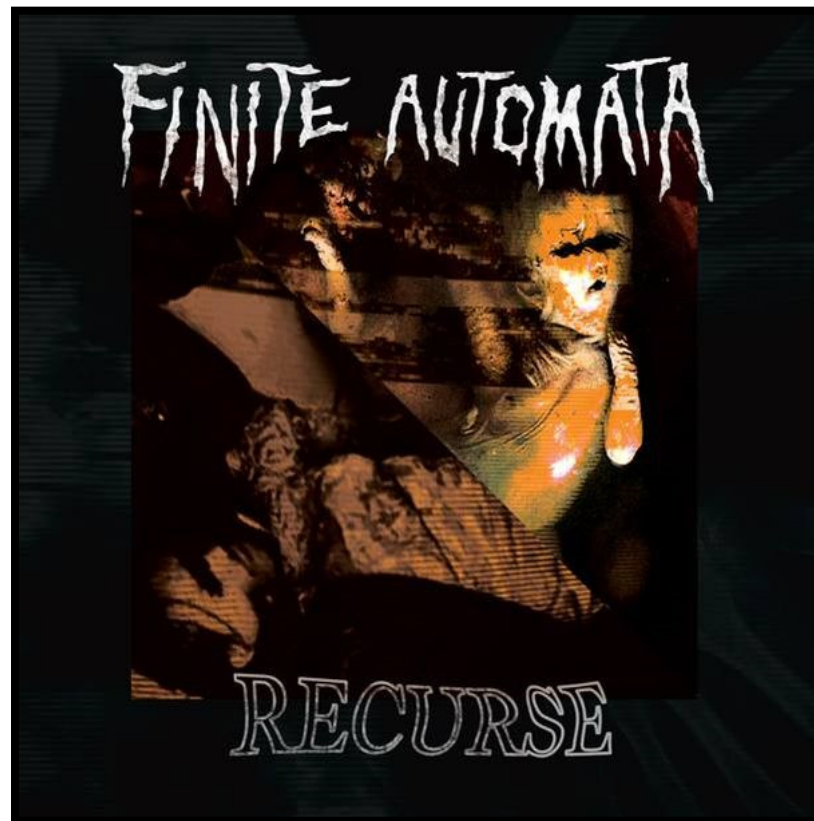


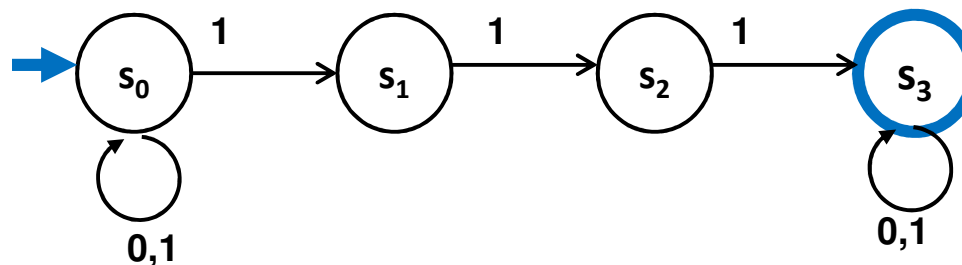
CSE 311: Foundations of Computing

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



Last time: Nondeterministic Finite Automata (NFA)

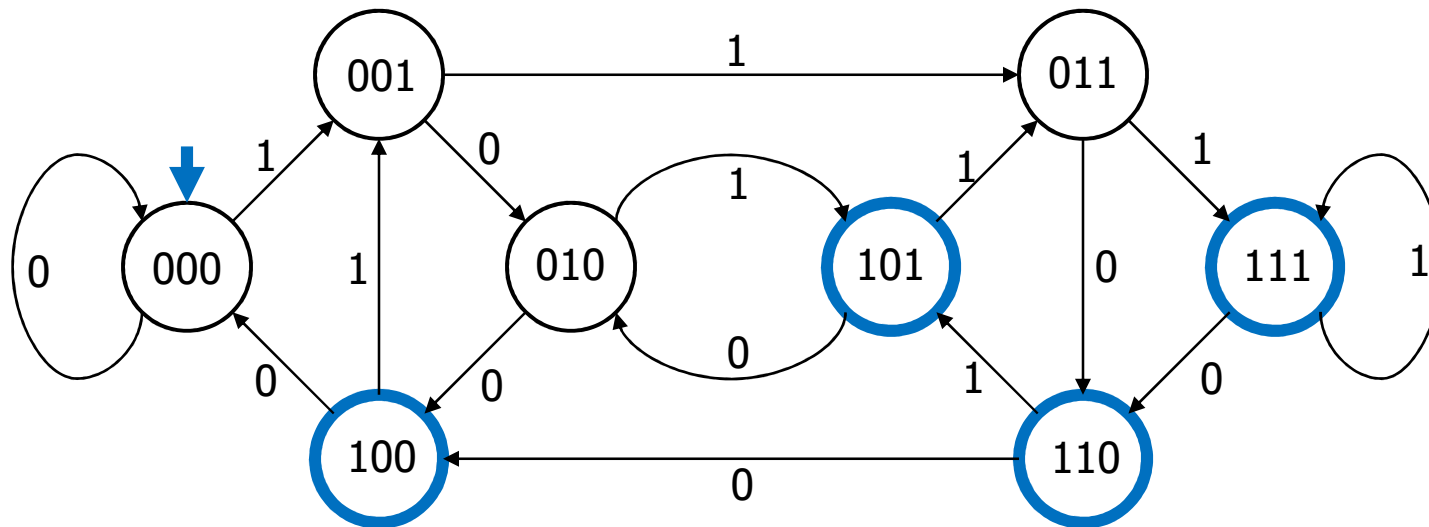
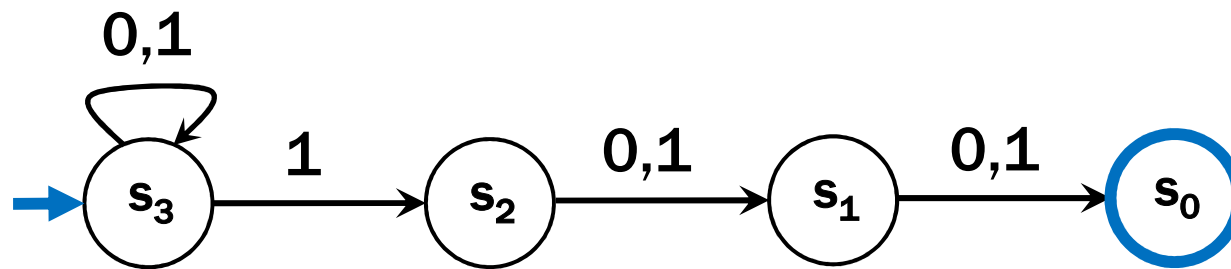
- Graph with start state, final states, edges labeled by symbols (like DFA) but
 - Not required to have exactly 1 edge out of each state labeled by each symbol— can have 0 or >1
 - Also can have edges labeled by empty string ϵ
- **Defn:** x is in the language recognized by an NFA if and only if x labels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



Last time: 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**

Last time: Compare with the smallest DFA



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:**

- \emptyset, 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

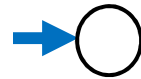
- **Case \emptyset :**

- **Case ε :**

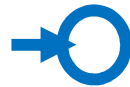
- **Case a :**

Base Case

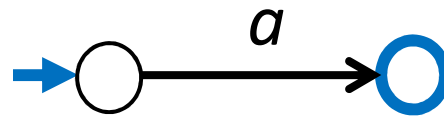
- Case \emptyset :



- Case ε :

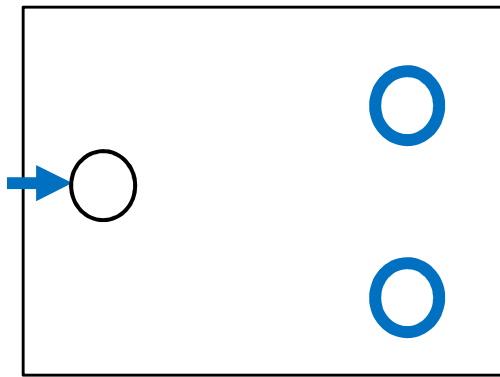


- Case a :

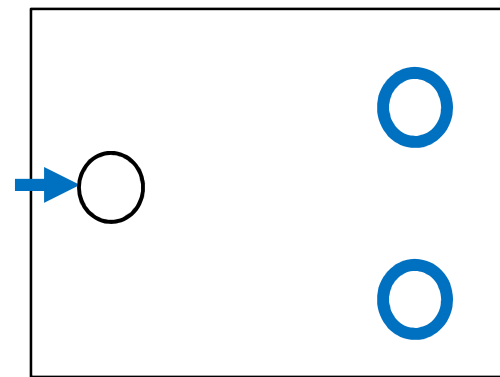


Inductive Hypothesis

- 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



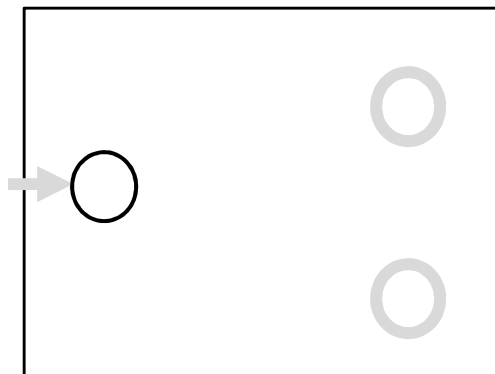
N_A



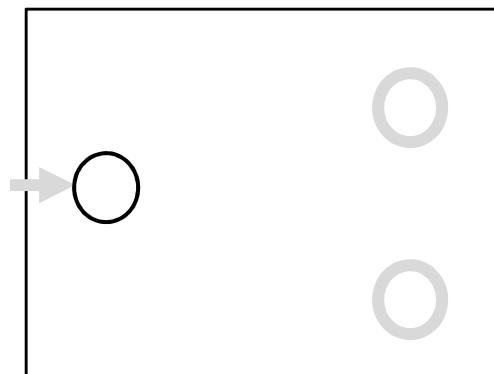
N_B

Inductive Step

Case $(A \cup B)$:



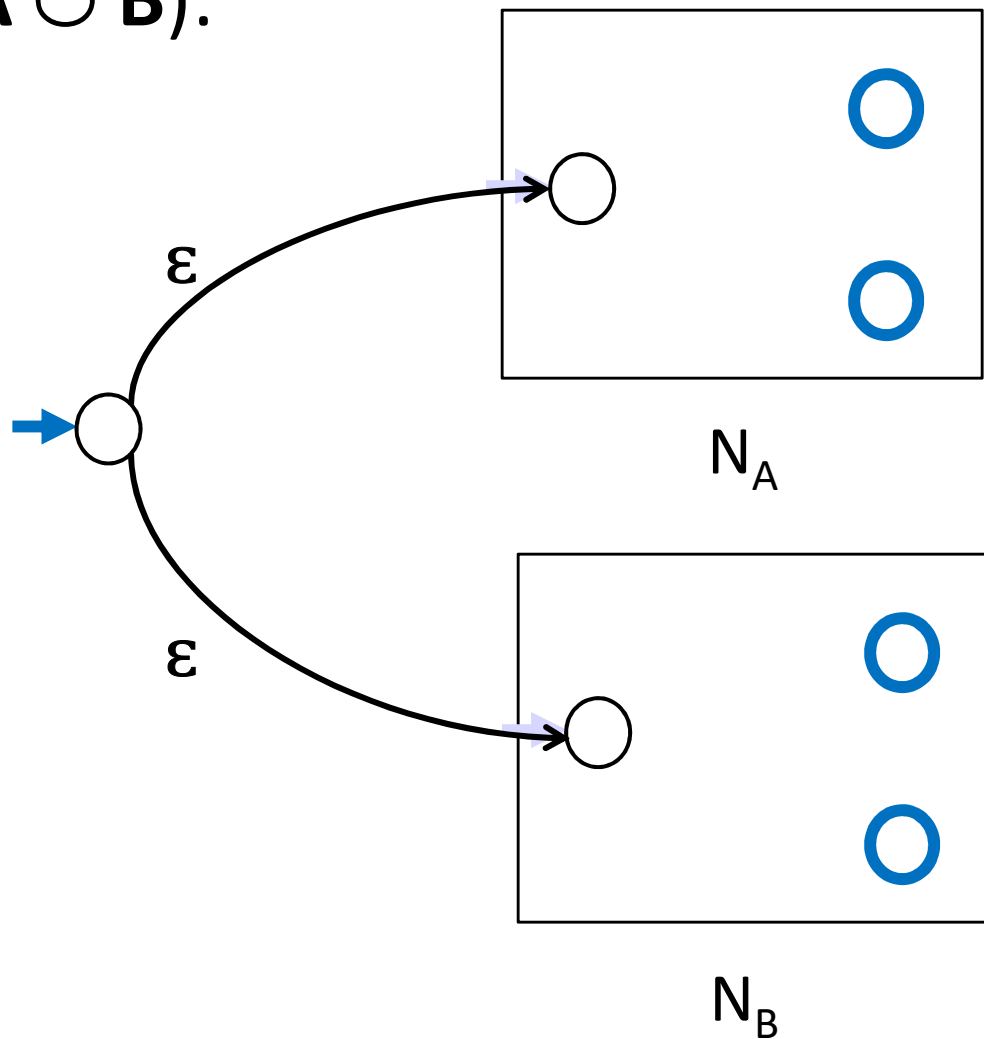
N_A



N_B

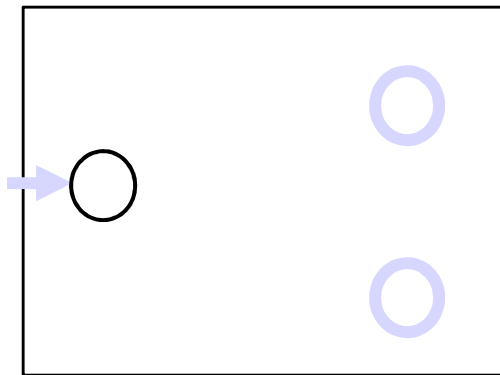
Inductive Step

Case $(A \cup B)$:

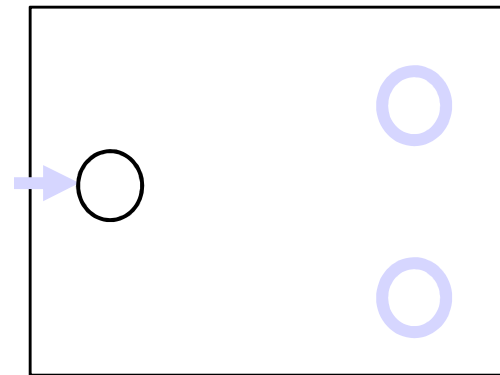


Inductive Step

Case (AB):



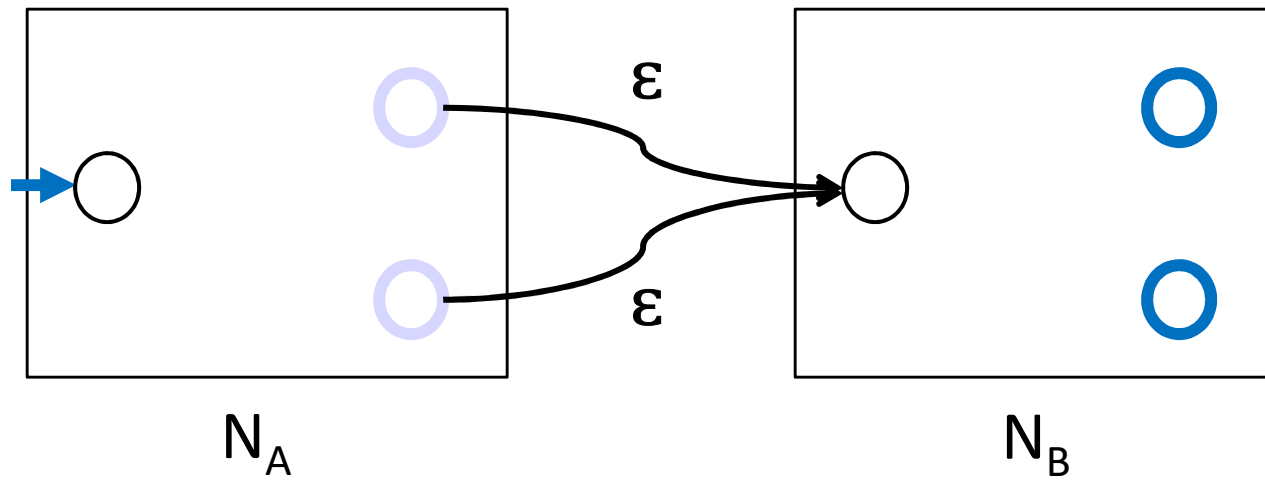
N_A



N_B

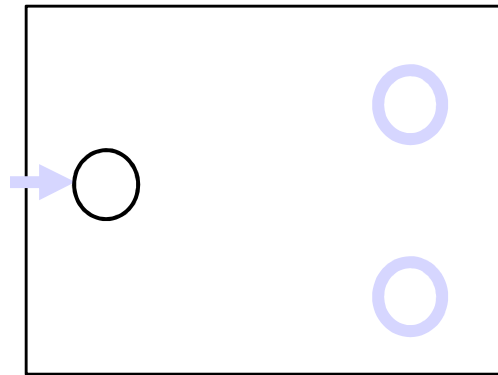
Inductive Step

Case (AB):



Inductive Step

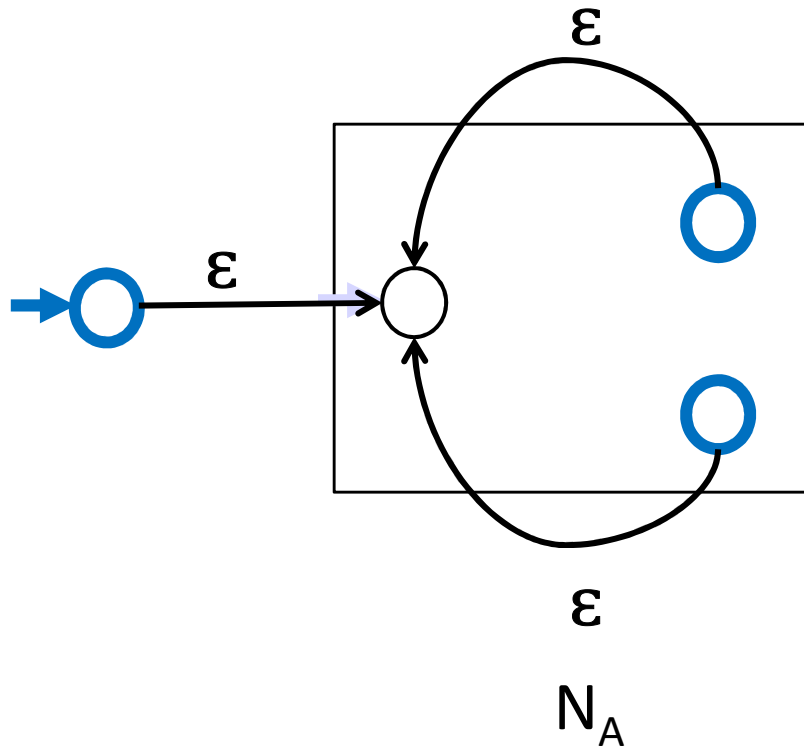
Case A*



N_A

Inductive Step

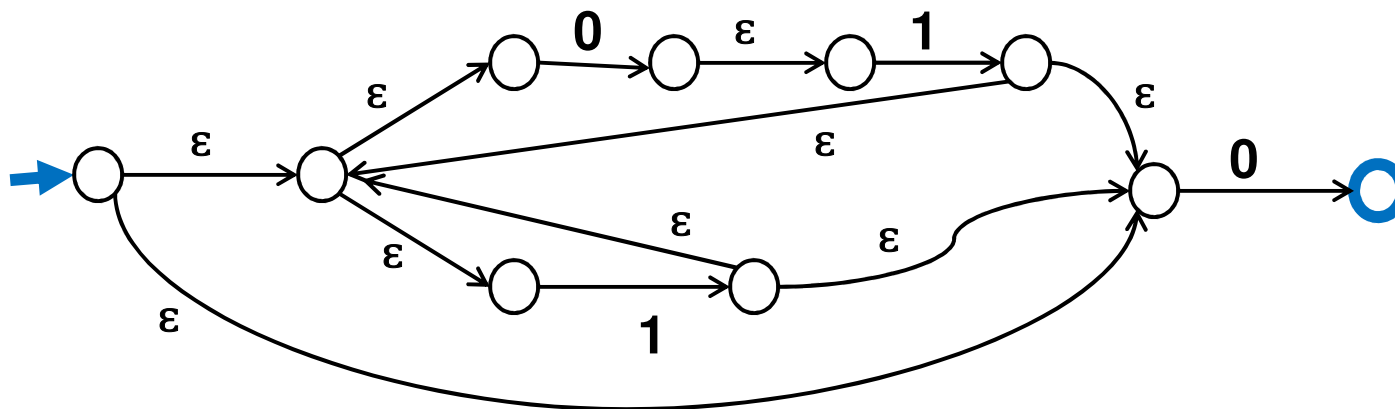
Case A*



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

Solution

$(01 \cup 1)^*0$



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

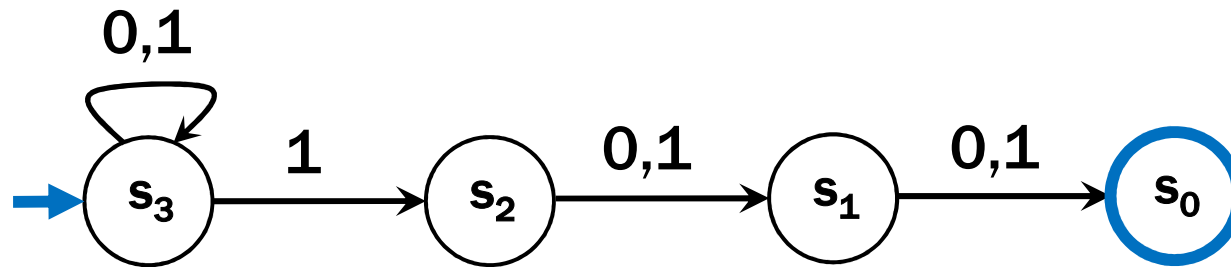
Three ways of thinking about NFAs

- **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

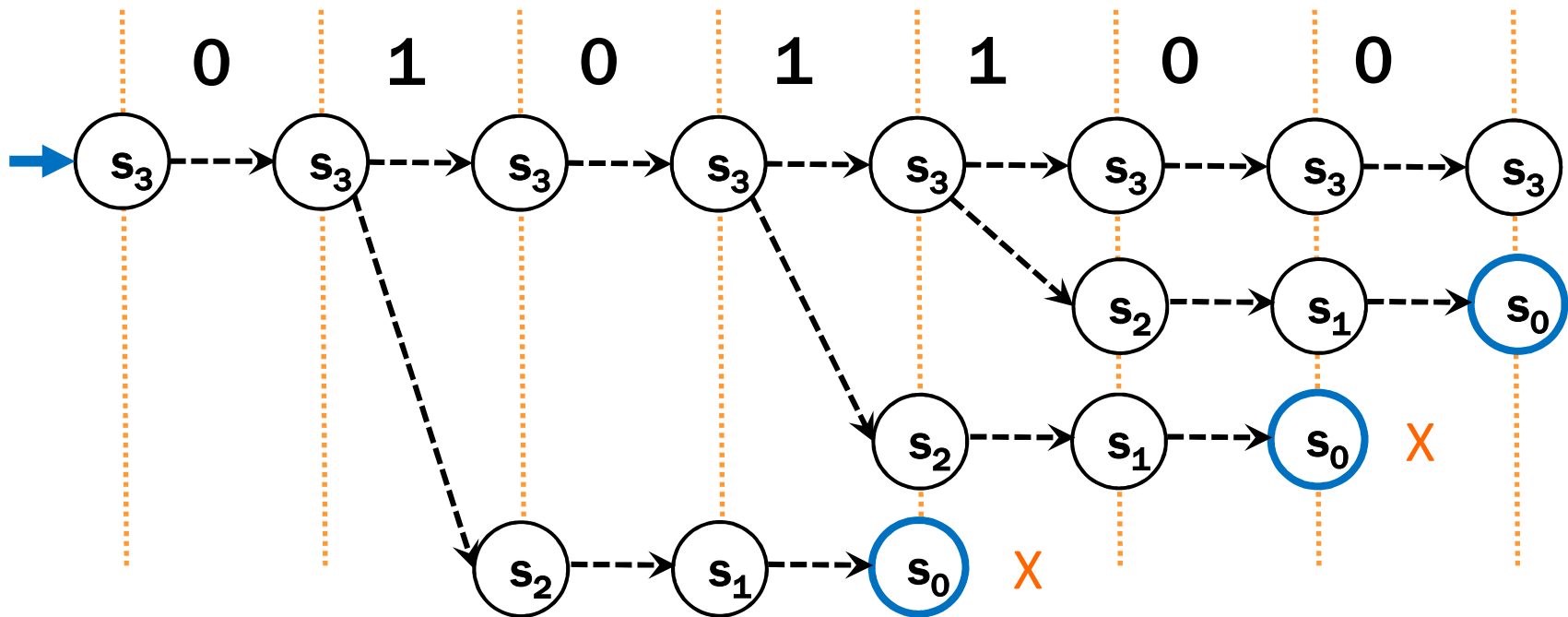
Conversion of NFAs to a DFAs

- **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



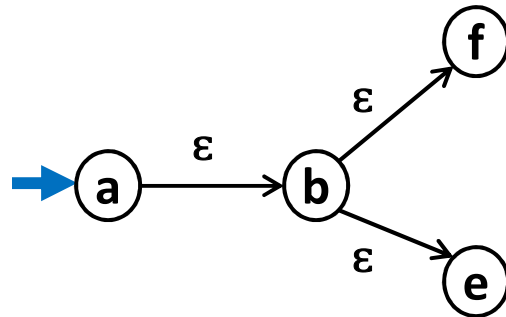
Input string 0101100



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

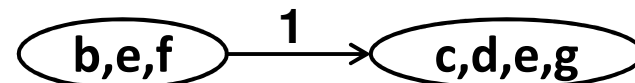
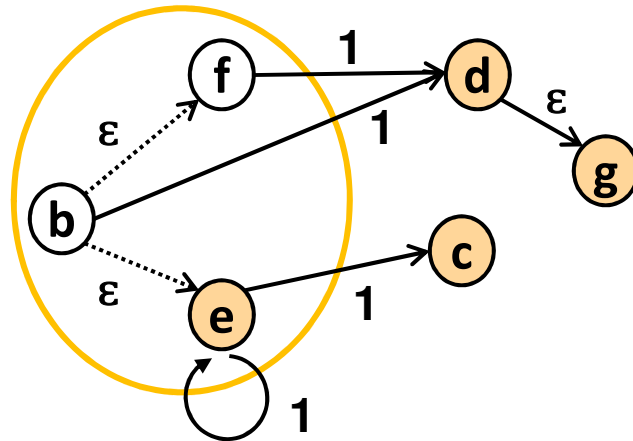


DFA

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

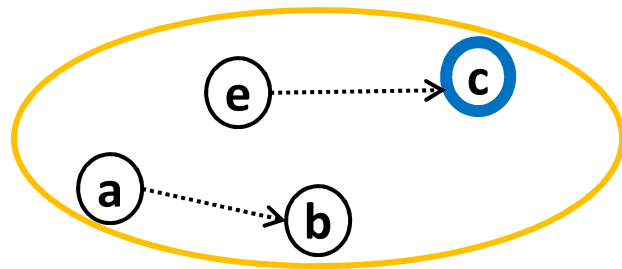
- Add an edge labeled s to state corresponding to T , the set of states of the NFA reached by
 - starting from some state in S , then
 - following one edge labeled by s , and then following some number of edges labeled by ϵ
- T will be \emptyset if no edges from S labeled s exist



Conversion of NFAs to a DFAs

Final states for the DFA

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

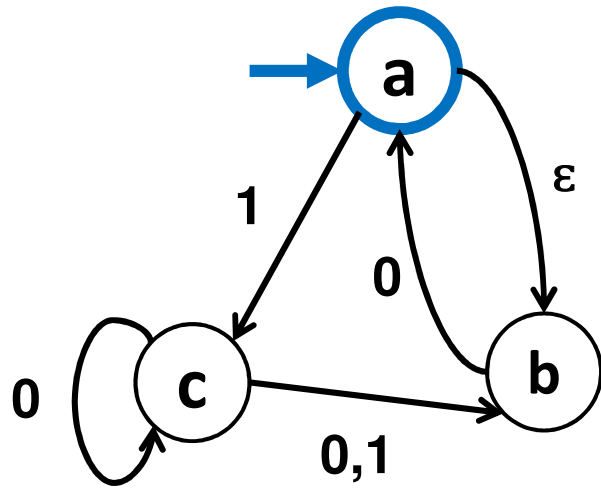


NFA



DFA

Example: NFA to DFA

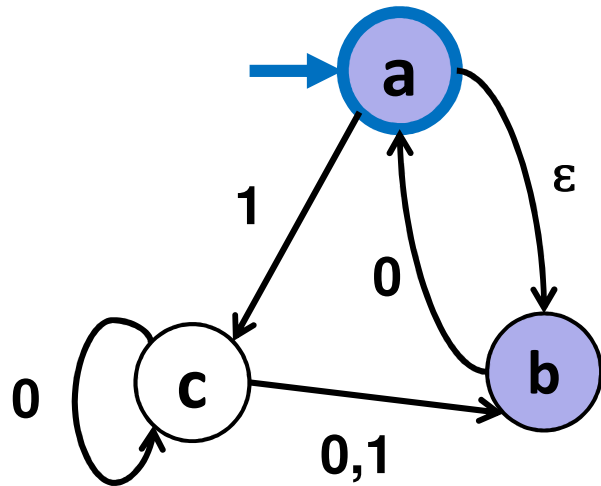


NFA

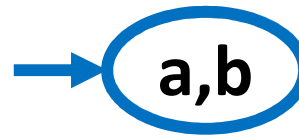


DFA

Example: NFA to DFA

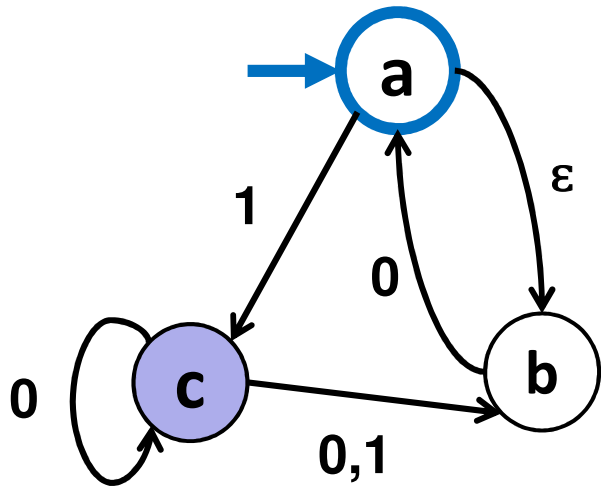


NFA

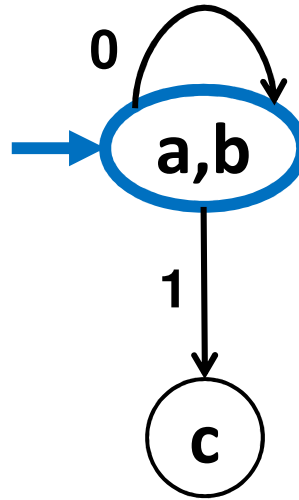


DFA

Example: NFA to DFA

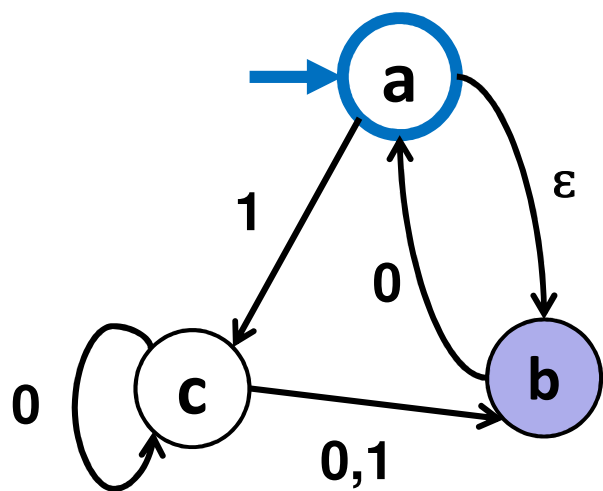


NFA

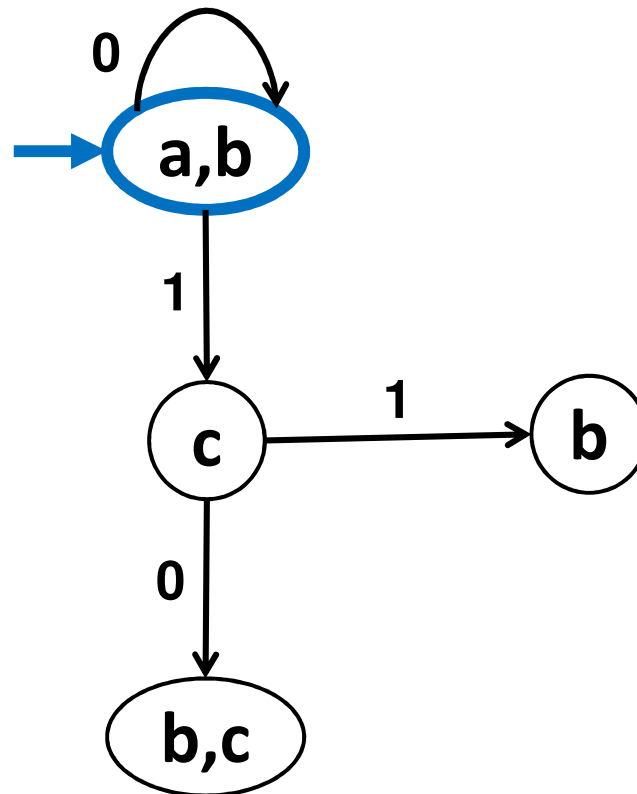


DFA

Example: NFA to DFA

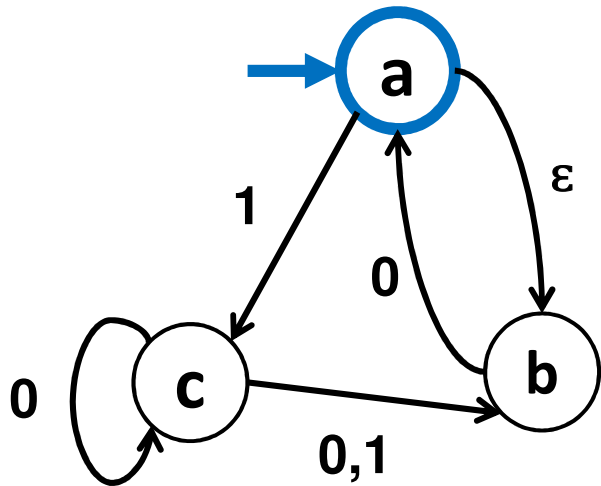


NFA

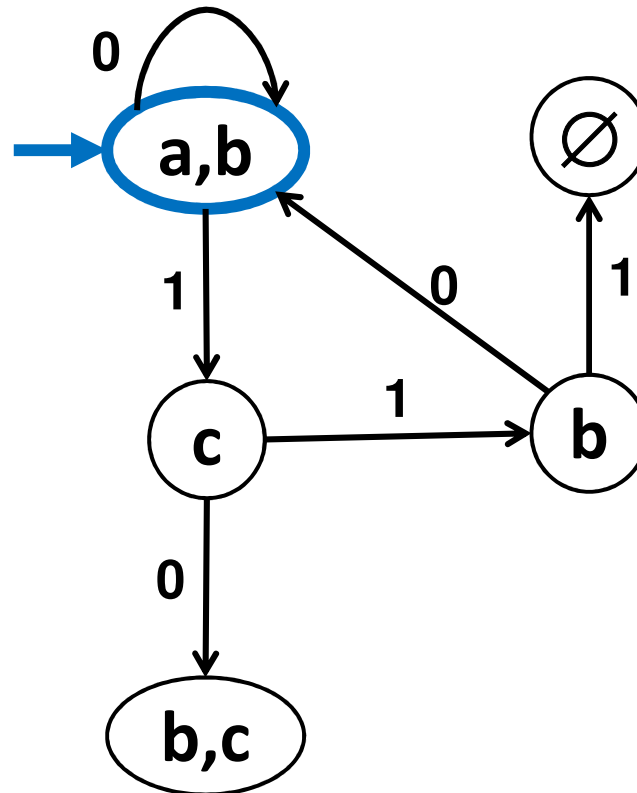


DFA

Example: NFA to DFA

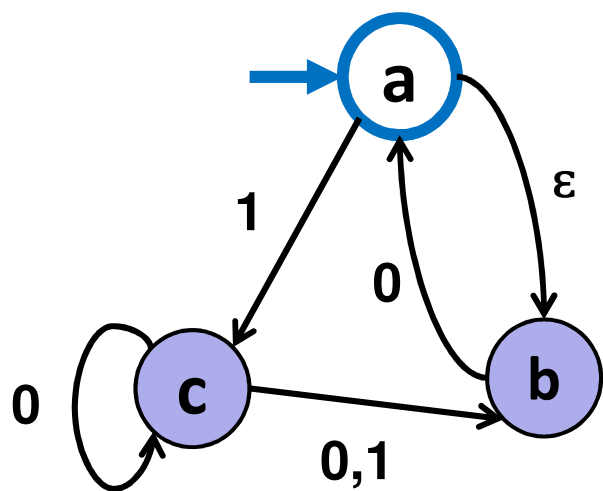


NFA

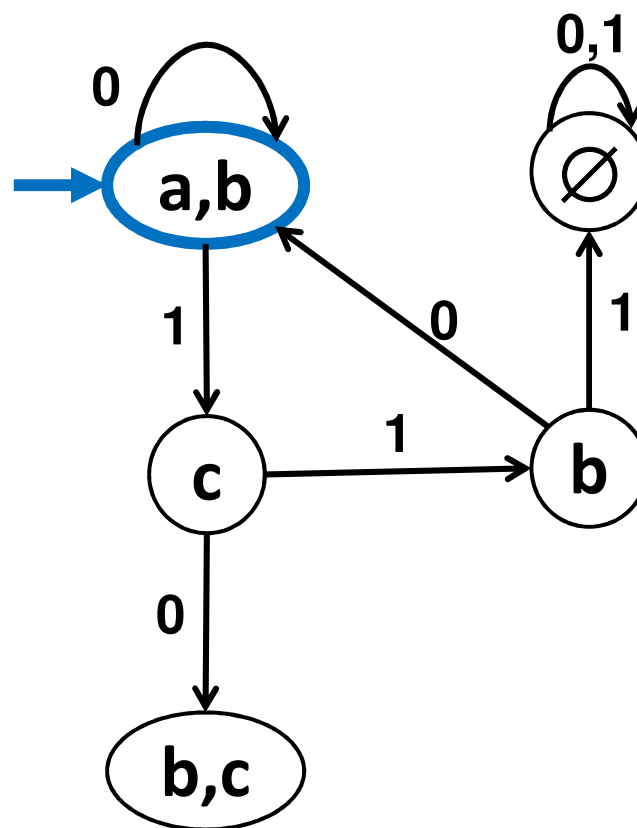


DFA

Example: NFA to DFA

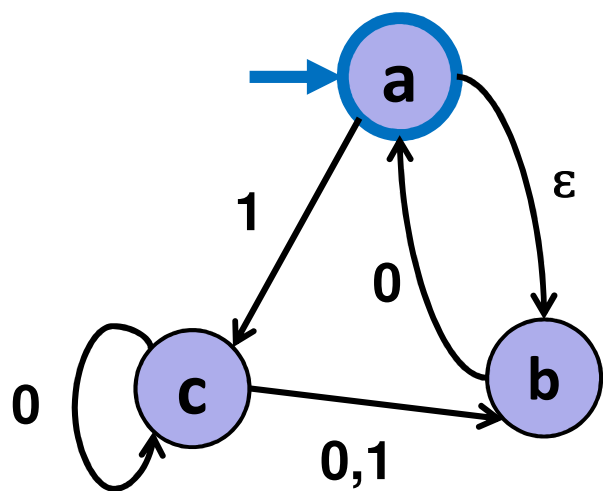


NFA

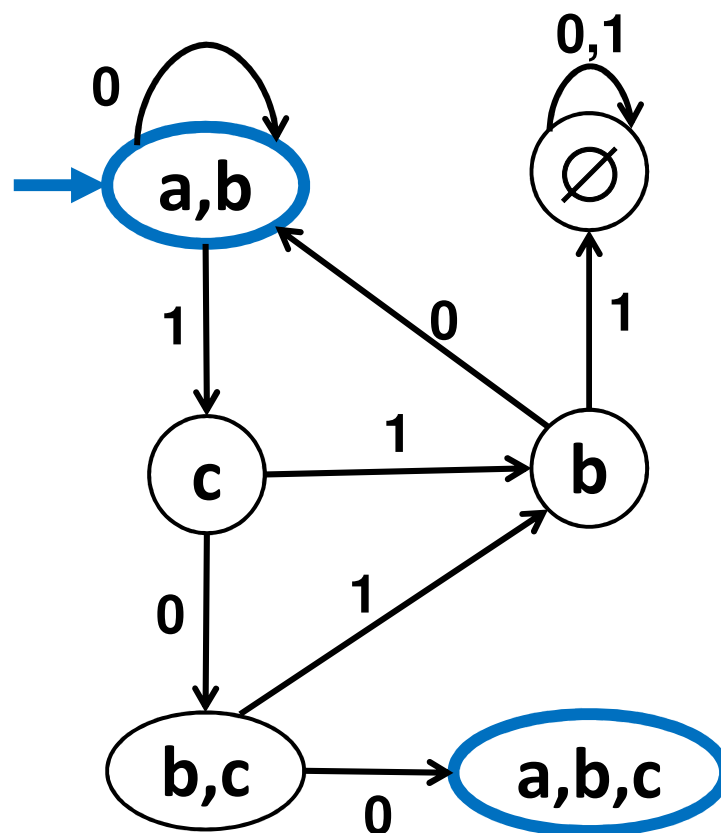


DFA

Example: NFA to DFA

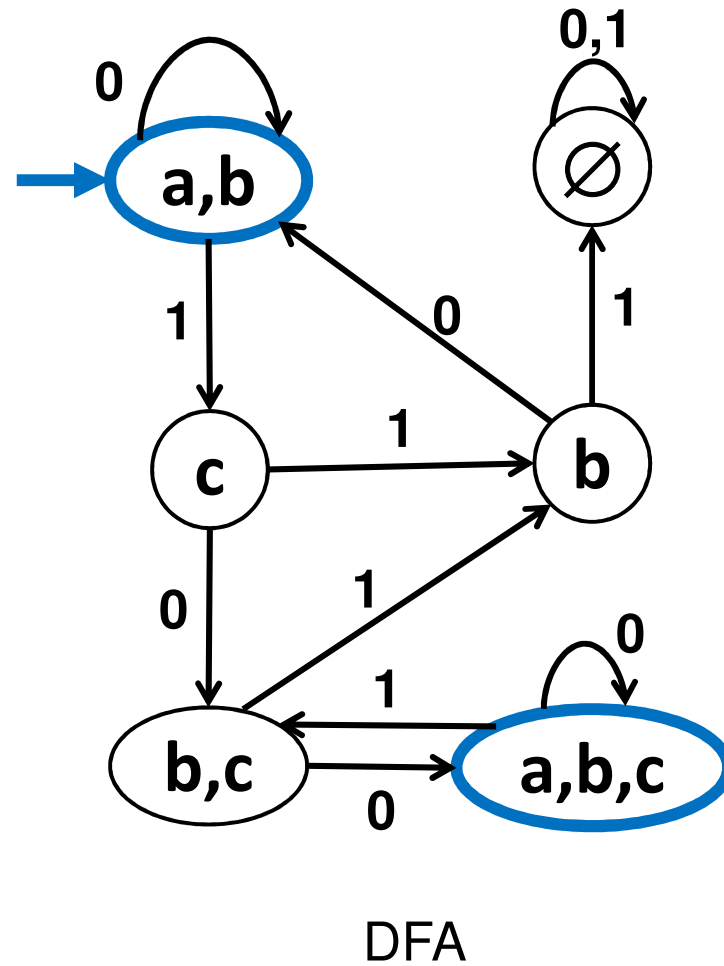
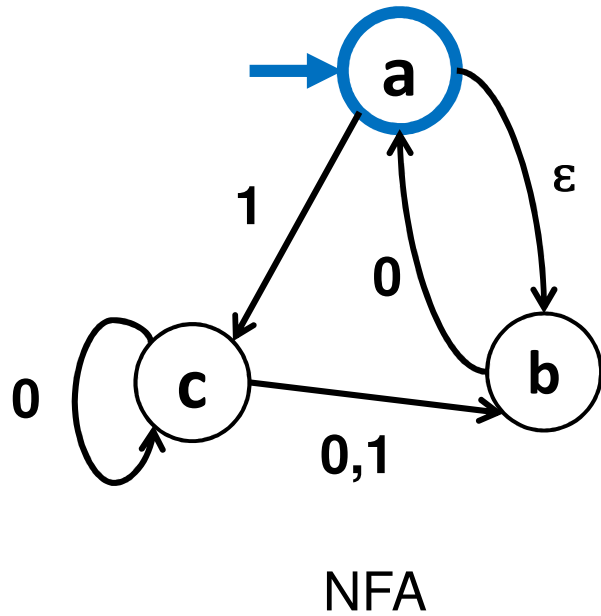


NFA



DFA

Example: NFA to DFA



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

DFAs \equiv NFAs \equiv Regular expressions

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

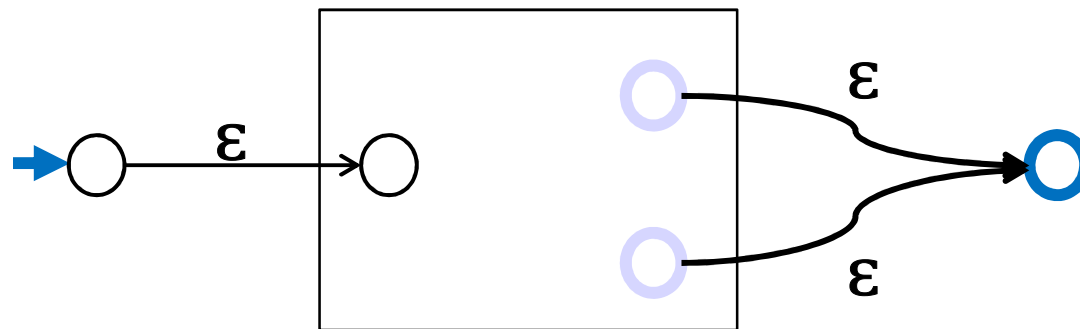
You need to know this fact but we won't ask you anything about the "only if" direction from DFA/NFA to regular expression. For fun, we sketch the idea.

Generalized NFAs

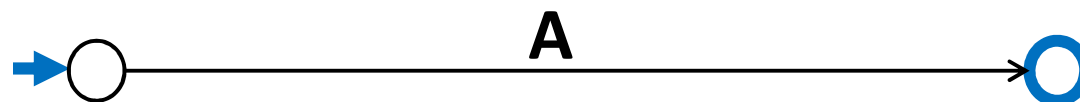
- Like NFAs but allow
 - Parallel edges
 - Regular Expressions as edge labels
 - NFAs already have edges labeled ϵ or a
- An edge labeled by A can be followed by reading a string of input chars that is in the language represented by A
- Defn: A string x is accepted iff there is a *path* from start to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

Starting from an NFA

Add new start state and final state



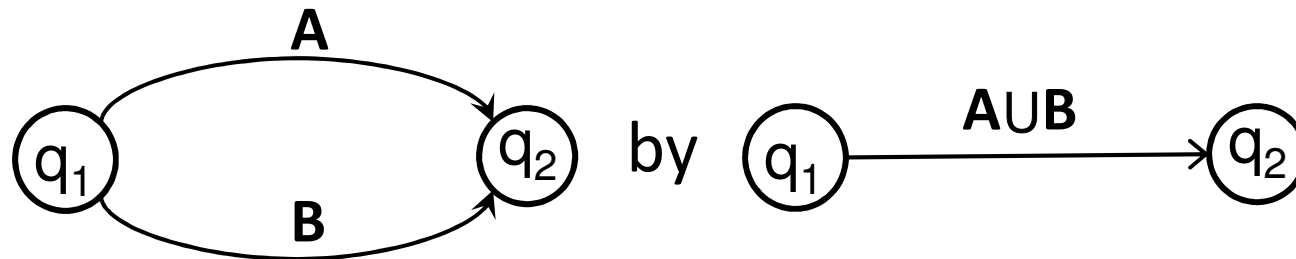
Then eliminate original states one by one, keeping the same language, until it looks like:



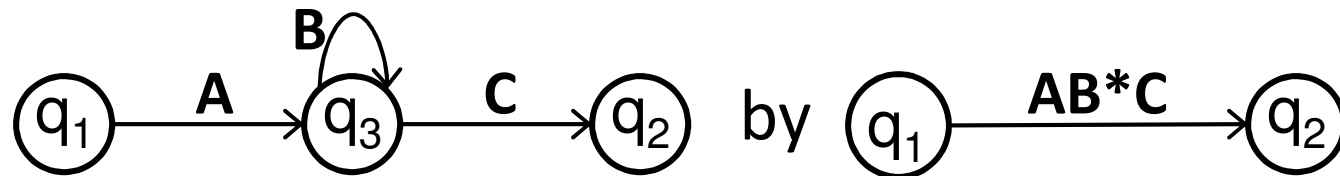
Final regular expression will be A

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

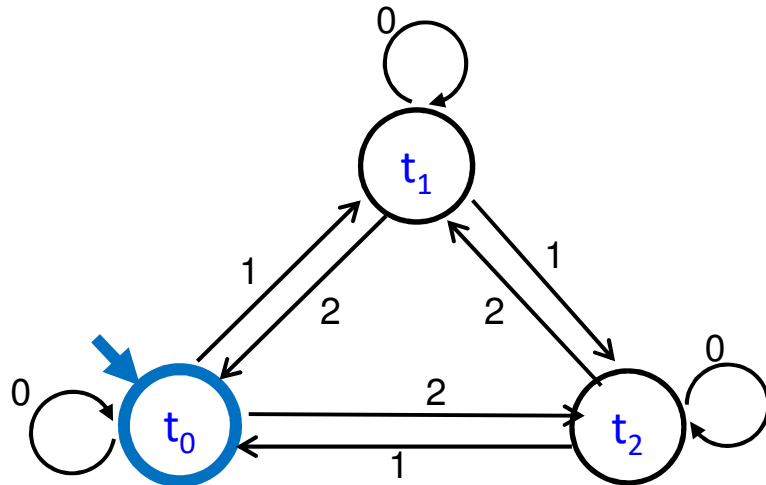


for every pair of states q_1, q_2 (even if $q_1=q_2$)

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

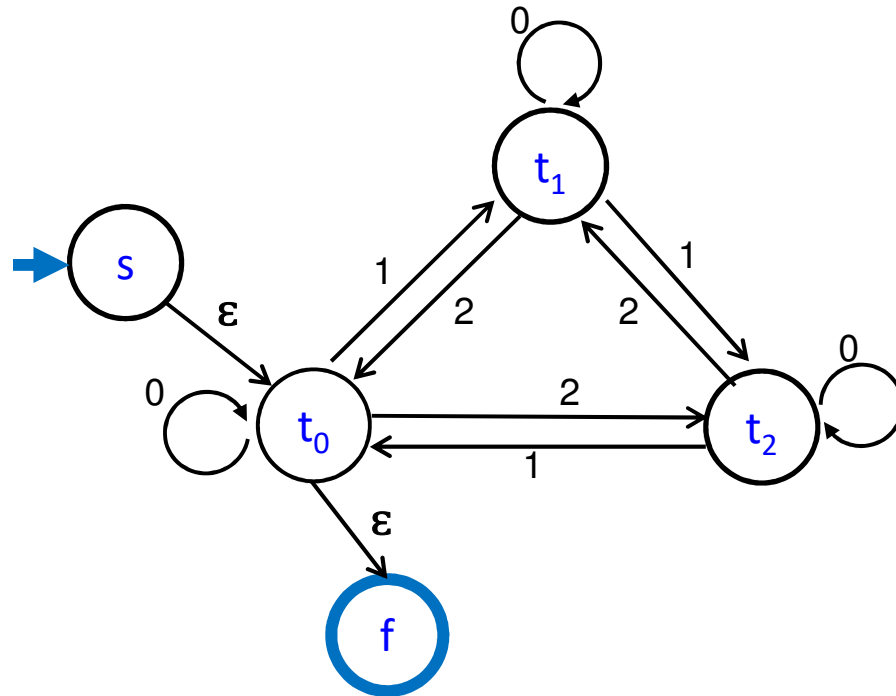
Regular expressions to add to edges

$t_0 \rightarrow t_1 \rightarrow t_0$: 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

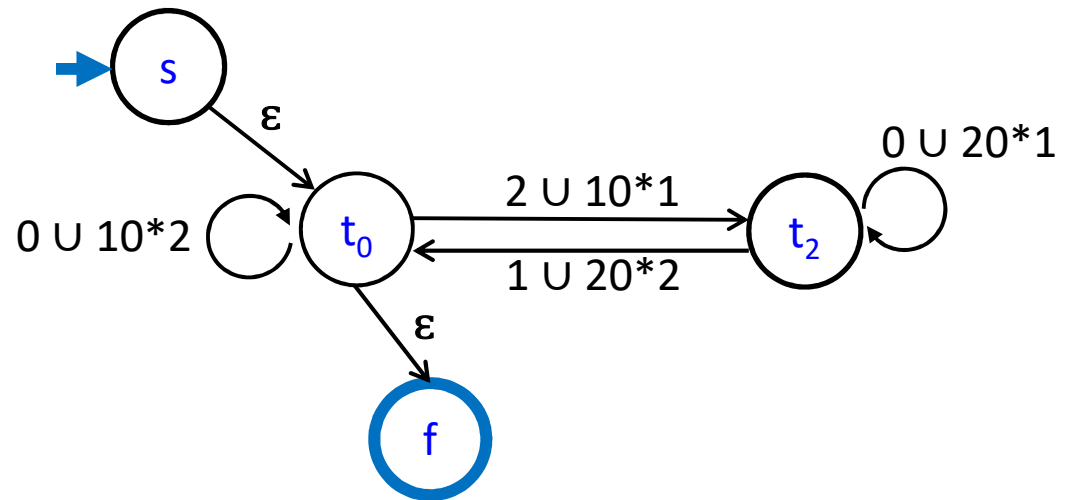
Regular expressions to add to edges

$t_0 \rightarrow t_1 \rightarrow t_0$: 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



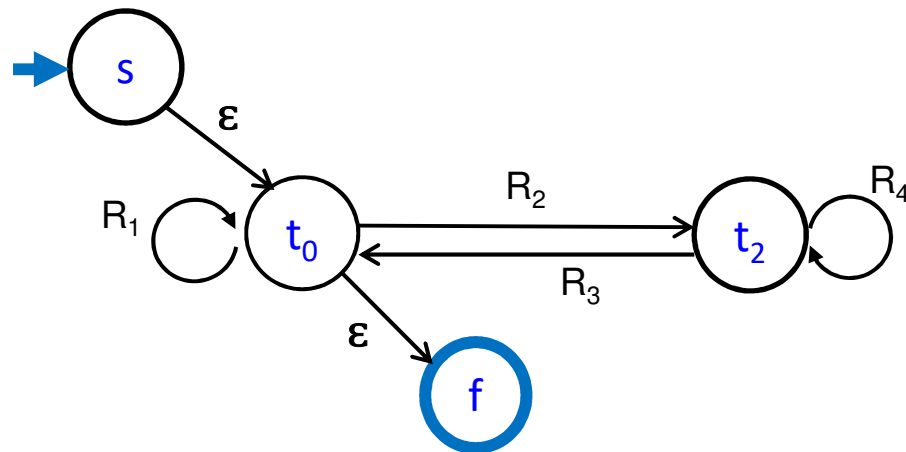
Splicing out state t_2 (and then t_0)

$$R_1: 0 \cup 10^*2$$

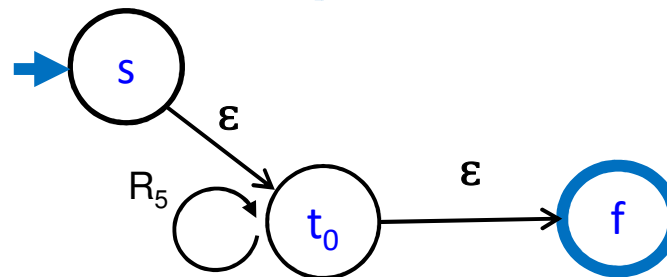
$$R_2: 2 \cup 10^*1$$

$$R_3: 1 \cup 20^*2$$

$$R_4: 0 \cup 20^*1$$



$$R_5: R_1 \cup R_2 R_4^* R_3$$



Final regular expression: $R_5^* =$

$$(0 \cup 10^*2 \cup (2 \cup 10^*1)(0 \cup 20^*1)^*(1 \cup 20^*2))^*$$