BU CS 332 – Theory of Computation

https://forms.gle/2CcC2ZVTBd12ziwX6

Lecture 4:

- More on NFAs
- NFAs vs. DFAs
- Closure Properties?

Reading:

Sipser Ch 1.1-1.2

Alexander Poremba & Mark Bun

January 29, 2026

Last Time

- Deterministic Finite Automata (DFAs)
  - Informal description: State diagram
  - Formal description: What are they?
  - Formal description: How do they compute?
  - A language is regular if it is recognized by a DFA
- Intro to Nondeterministic Finite Automata (NFAs)

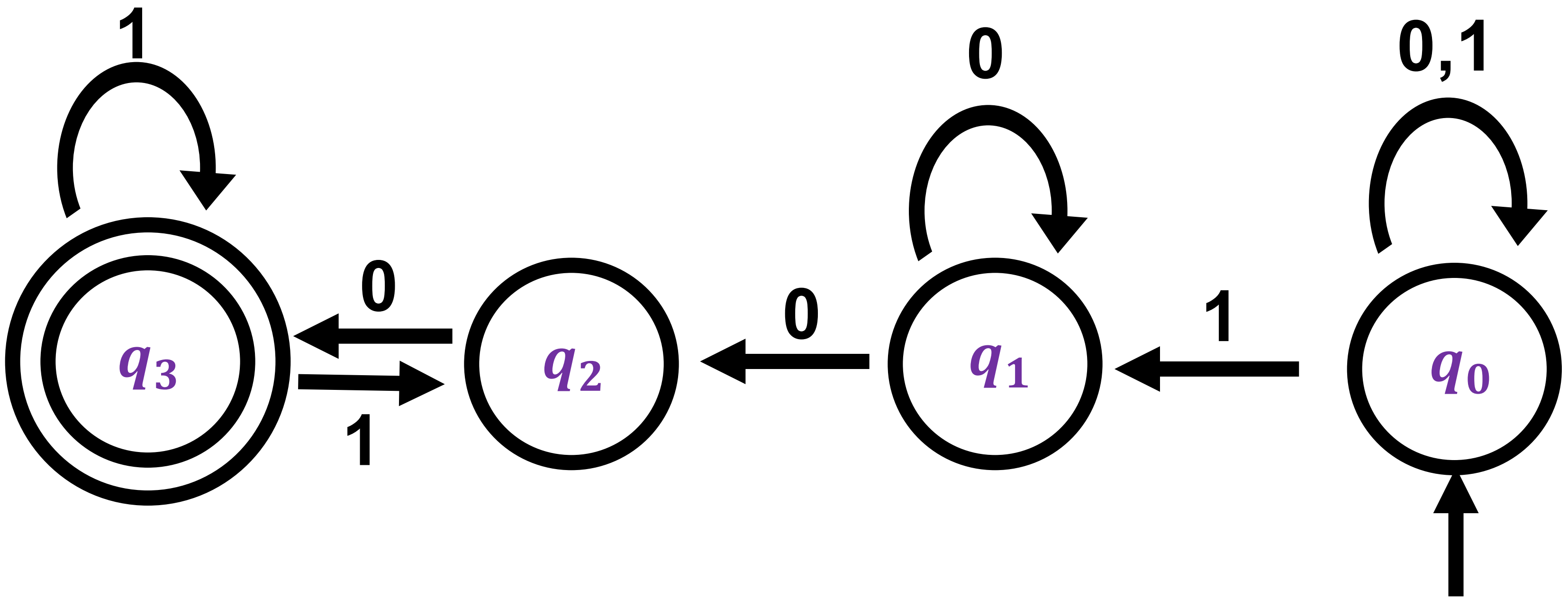
Nondeterminism

In a DFA, the machine is always in exactly one state upon reading each input symbol

In a nondeterministic FA, the machine can try out many different ways of reading the same string

- Next symbol may cause an NFA to “branch” into multiple possible computations
- Next symbol may cause NFA’s computation to fail to enter any state at all

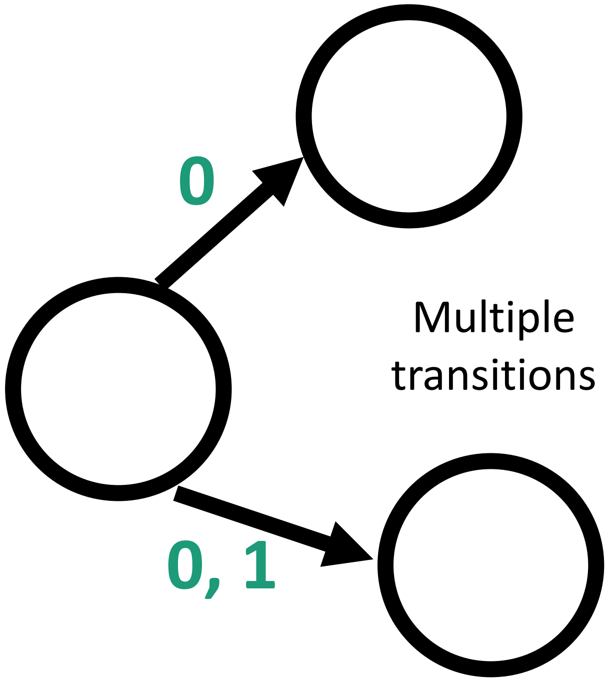
Nondeterminism



A Nondeterministic Finite Automaton (NFA) accepts if there ***exists*** a way to make it reach an accept state.

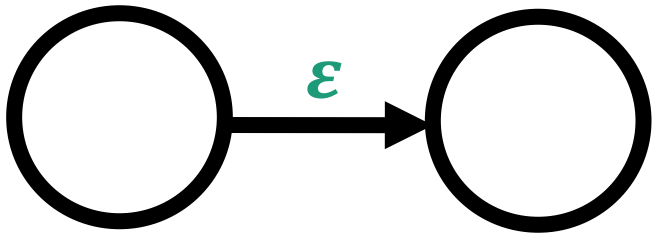
Ex. This NFA accepts input 1100, but does not accept input 11

Some special transitions

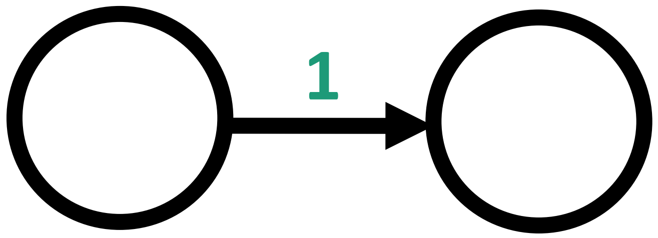


Multiple transitions

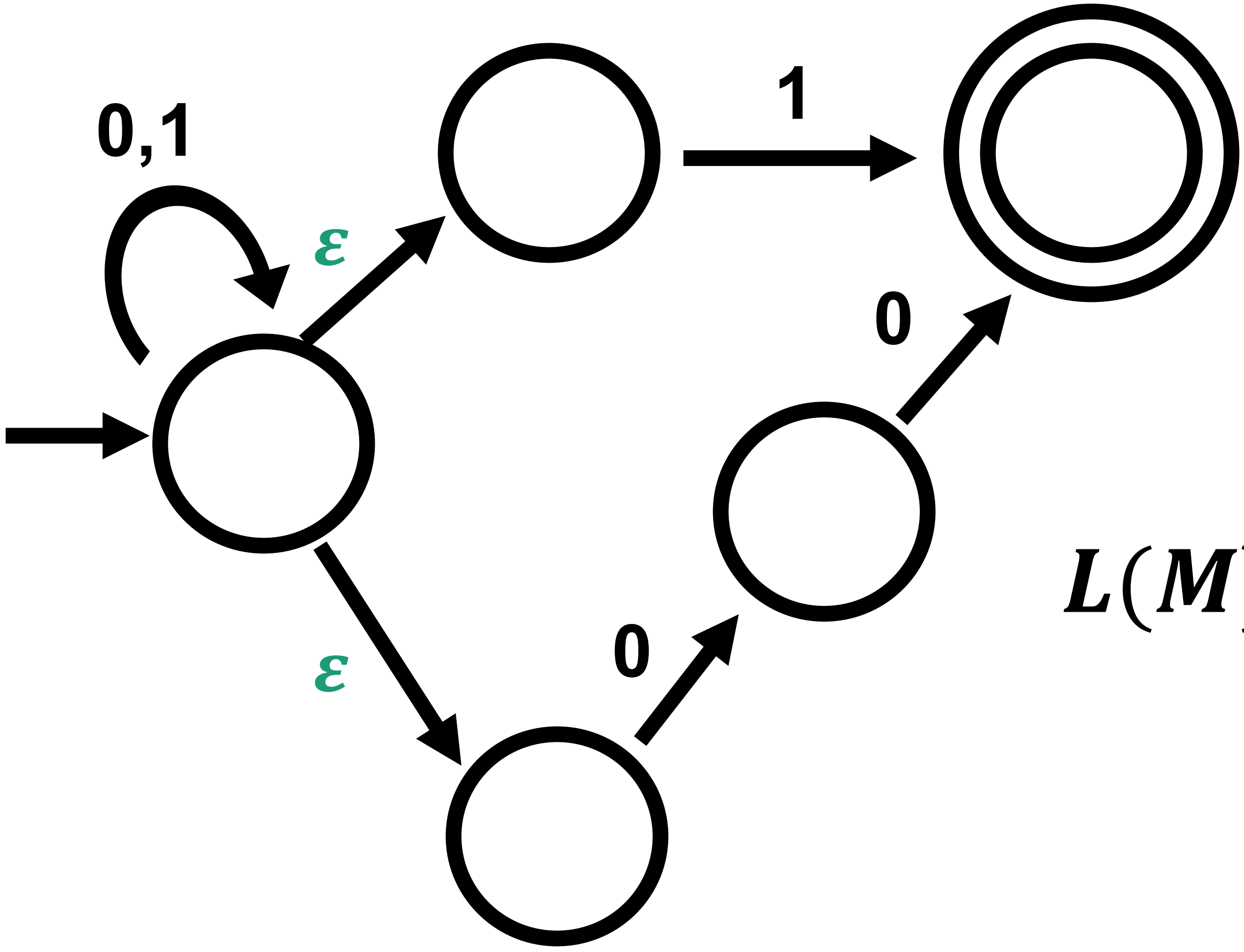
ε-transitions
(don’t consume a symbol)



No transition

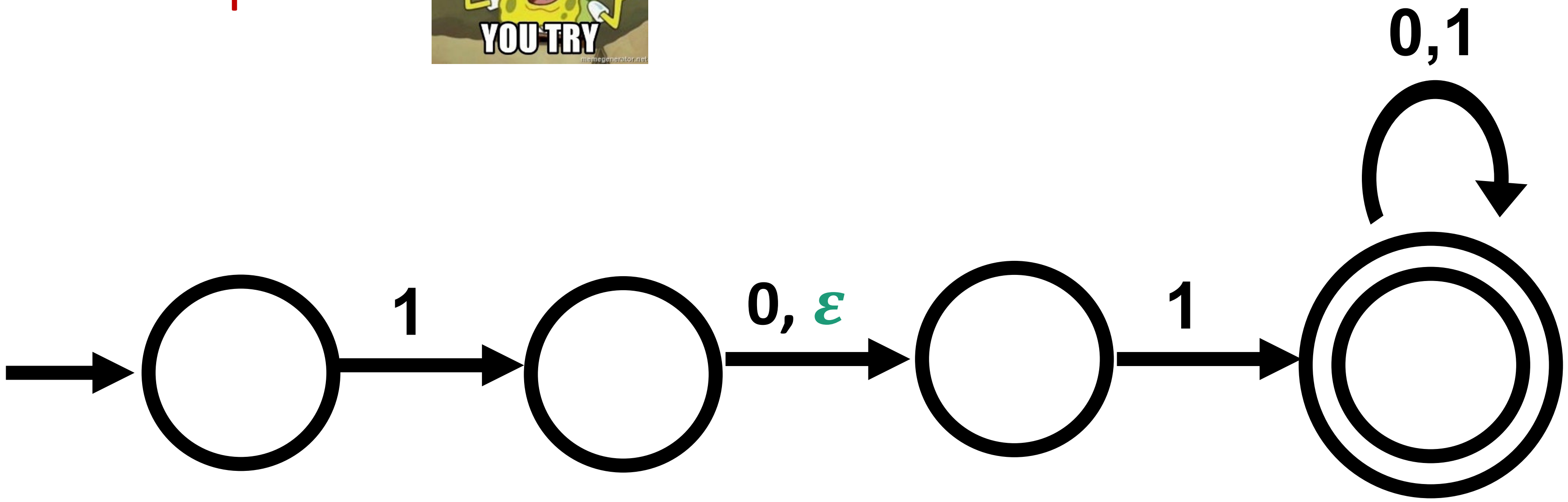


Example



$L(M) =$

Example


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "1"] & q_1 \arrow[r, "{0,\ \varepsilon}"] & q_2 \arrow[r, "1"] & \boxed{q_3} \arrow[loop above, "{0,1}"]
\end{tikzcd}$$

$\boldsymbol{L(N)} =$

a) $\{w \mid w \text{ contains } 101\}$
b) $\{w \mid w \text{ contains } 11 \text{ or } 101\}$
c) $\{w \mid w \text{ starts with } 101\}$
d) $\{w \mid w \text{ starts with } 11 \text{ or } 101\}$

Formal Definition of a NFA

An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

Q is the set of states

Σ is the alphabet

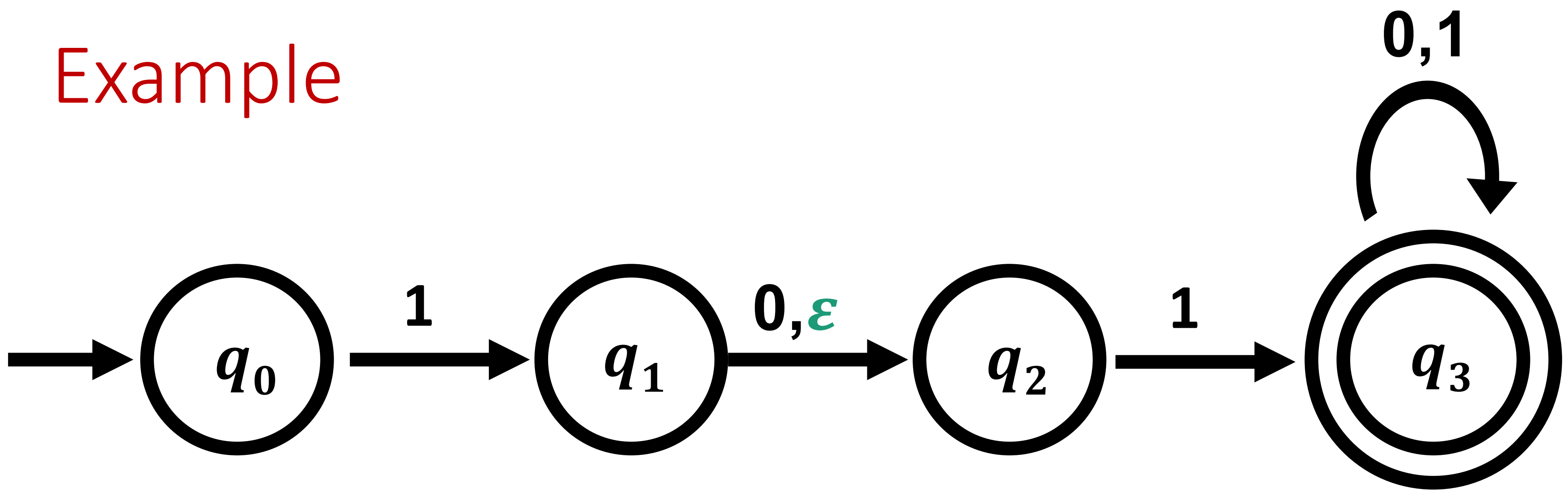
$\delta\colon Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept states

M ***accepts*** a string w if **there exists** a path from q_0 to an accept state that can be followed by reading w.

Example


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "1"] & q_1 \arrow[r, "{0,\varepsilon}"] & q_2 \arrow[r, "1"] & \boxed{q_3} \arrow[loop above, "{0,1}"]
\end{tikzcd}$$

$N = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$F = \{q_3\}$

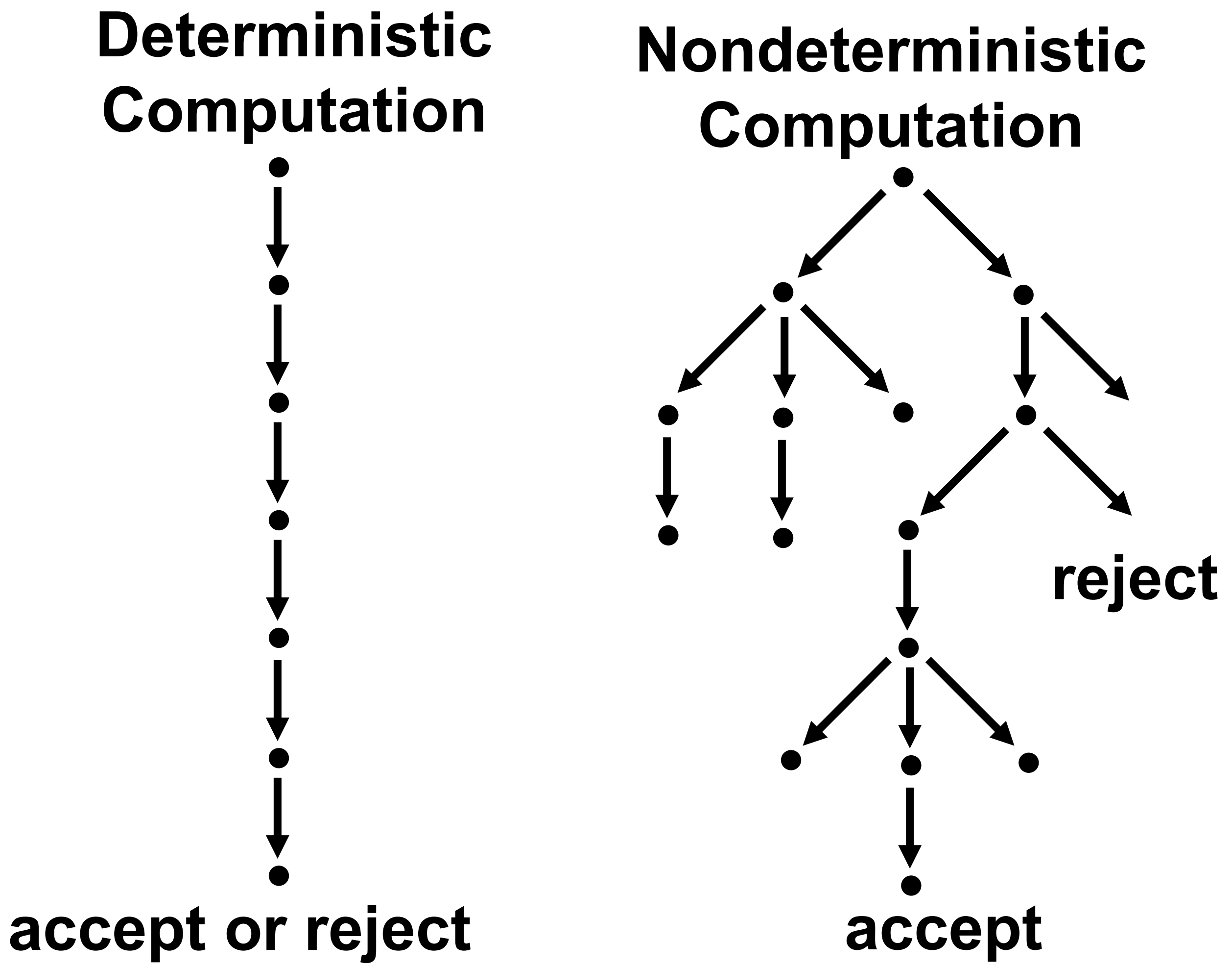
$\delta(q_0, 0) =$

$\delta(q_0, 1) =$

$\delta(q_1, \varepsilon) =$

$\delta(q_2, 0) =$

Nondeterminism



Ways to think about nondeterminism

- (restricted) parallel computation
- tree of possible computations
- guessing and verifying the "right" choice

Why study NFAs?

- Not really a realistic model of computation: Real computing devices can’t really try many possibilities in parallel

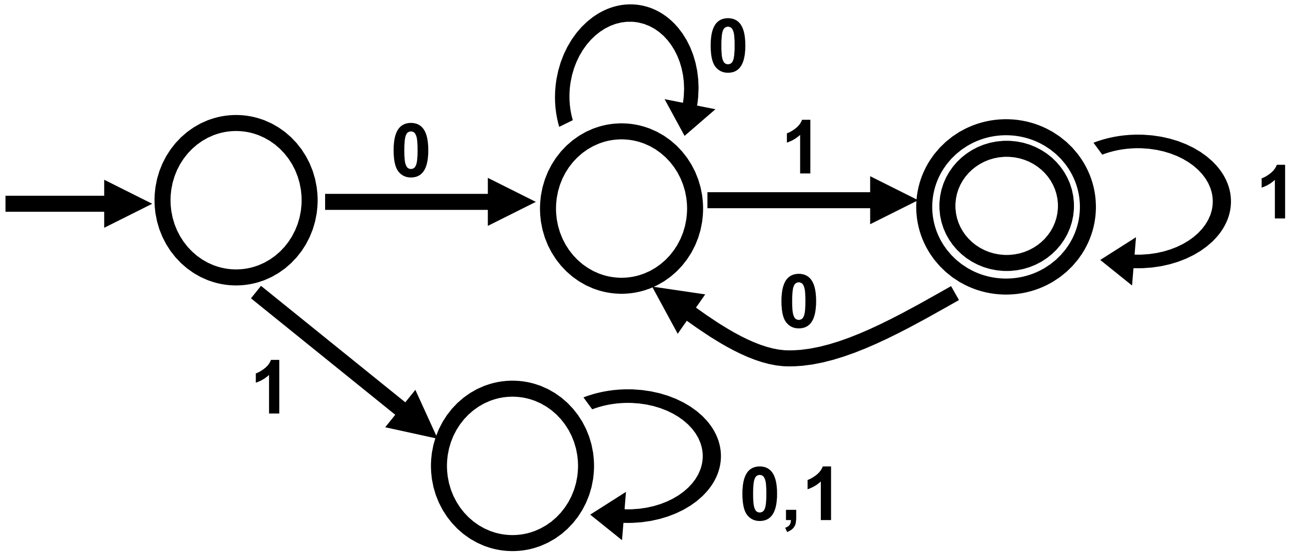
But:

- NFAs can be simpler than DFAs
- Useful for understanding power of DFAs/regular languages
- Lets us study “nondeterminism” as a resource

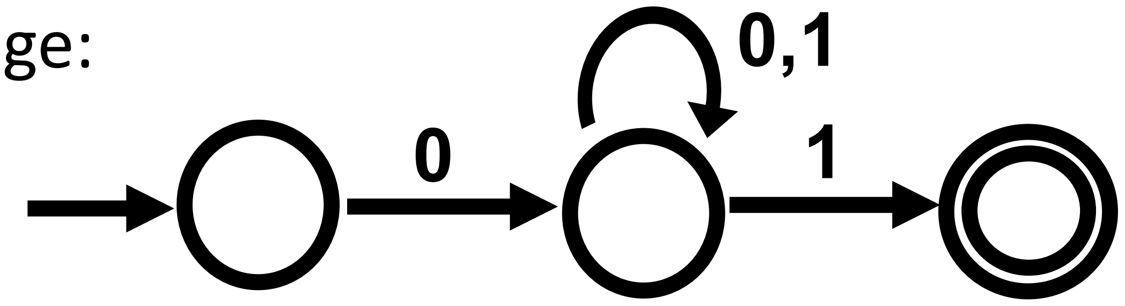
    (cf. P vs. NP)

NFAs can be simpler than DFAs

A DFA that recognizes the language
$\{w \mid w \text{ starts with } 0 \text{ and ends with } 1\}$:



An NFA for this language:



Equivalence of NFAs and DFAs

Equivalence of NFAs and DFAs

Every DFA *is* an NFA, so NFAs are *at least* as powerful as DFAs

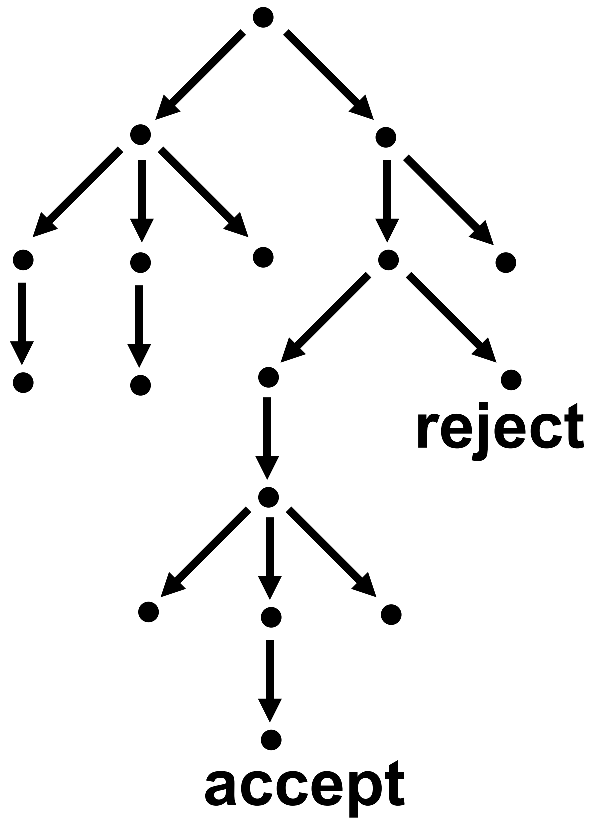
Theorem: For every NFA N, there is a DFA M such that $L(M) = L(N)$

Corollary: A language is regular if and only if it is recognized by an NFA

Equivalence of NFAs and DFAs (Proof)

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA

Goal: Construct DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing $L(N)$

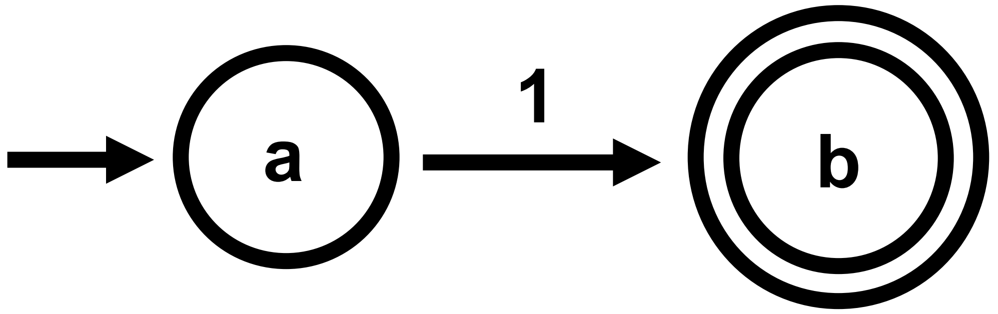


Intuition: Run all threads of N in parallel, maintaining the set of states where all threads are.

Formally: $Q' = P(Q)$

"The Subset Construction"

NFA → DFA Example



Subset Construction (Formally, first attempt)

Input: NFA $N = (Q, \Sigma, \delta, q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing $L(N)$

Q'

$\delta' : Q' \times \Sigma \to Q'$

$\delta'(R, \sigma) =$ for all $R \subseteq Q$ and $\sigma \in \Sigma$.

$q_0' =$

$F' =$

Subset Construction (Formally, for real)

Input: NFA $N = (Q, \Sigma, \delta, q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing $L(N)$

$Q' = P(Q)$

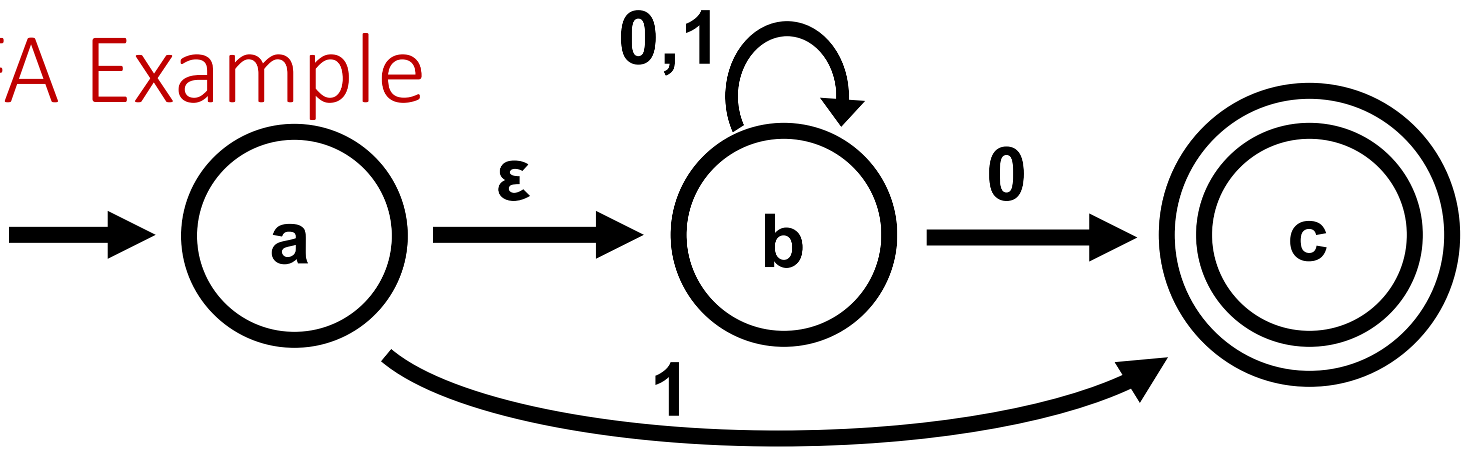
$\delta' : Q' \times \Sigma \rightarrow Q'$

$\delta'(R, \sigma) = \bigcup_{r \in R} \delta(r, \sigma)$ for all $R \subseteq Q$ and $\sigma \in \Sigma$.

$q_0' = \{q_0\}$

$F' = \{ R \in Q' \mid R$ contains some accept state of $N\}$

NFA → DFA Example



Proving the Construction Works

Claim: For every string w, running M on w leads to state

$$\{q \in Q \mid \text{There exists a computation path of } N \text{ on input } w \text{ ending at } q\}$$

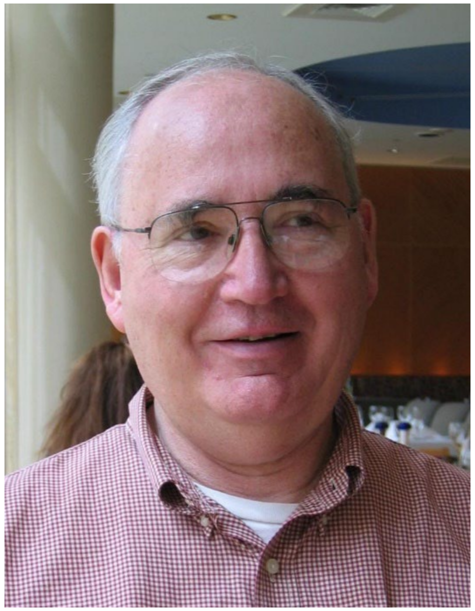
Proof idea: By induction on $|w|$

Historical Note

Subset Construction introduced in Rabin & Scott’s 1959 paper “Finite Automata and their Decision Problems”

1976 ACM Turing Award citation

For their joint paper "Finite Automata and Their Decision Problem," which introduced the idea of nondeterministic machines, which has proved to be an enormously valuable concept. Their (Scott & Rabin) classic paper has been a continuous source of inspiration for subsequent work in this field.



NFA → DFA: The Catch

If N is an NFA with s states, how many states does the DFA obtained using the subset construction have? (In the worst case.)

a) s
b) s^2
c) 2^s
d) None of the above

Is this construction the best we can do?

Subset construction converts an s state NFA into a 2^s-state DFA

Could there be a construction that always produces, say, an s^2-state DFA?

Theorem: For every $s \geq 1$, there is a language L_s such that

1. There is an $(s+1)$-state NFA recognizing L_s.
2. There is no DFA recognizing L_s with fewer than 2^s states.

Conclusion: For finite automata, nondeterminism provides an exponential savings over determinism (in the worst case).

Closure Properties

An Analogy

In algebra, we try to identify operations which are common to many different mathematical structures

Example: The integers $\mathbb{Z} = \{\ldots -2, -1, 0, 1, 2, \ldots\}$ are **closed** under

- Addition: $x + y$
- Multiplication: $x \times y$
- Negation: $-x$
- …but NOT Division: $x \,/\, y$

We'd like to investigate similar closure properties of the class of regular languages

Regular operations on languages

Let $A, B \subseteq \Sigma^*$ be languages. Define

Union: $A \cup B = \{w \mid w \in A \textbf{ or } w \in B\}$

Concatenation: $A \circ B = \{xy \mid x \in A, y \in B\}$

Star: $A^* =$

Other operations

Let $A, B \subseteq \Sigma^*$ be languages. Define

Complement: $\bar{A} = \{w \mid w \notin A\}$

Intersection: $A \cap B = \{w \mid w \in A \textbf{ and } w \in B\}$

Reverse: $A^R = \{w \mid w^R \in A\}$

Closure properties of the regular languages

Theorem: The class of regular languages is closed under all three regular operations (union, concatenation, star), as well as under complement, intersection, and reverse.

That is, if A and B are regular, then so are

$$A \cup B, \quad A \circ B, \quad A^*, \quad \bar{A}, \quad A \cap B, \quad \text{and } A^R$$

Proving Closure Properties

Complement

Complement: $\bar{A} = \{ w \mid w \notin A\}$

Theorem: The regular languages are closed under complement, i.e., if A is regular, then $\bar{A}$ is also regular

Proof idea:

On proving your own closure properties

You'll have homework/test problems of the form "show that the regular languages are closed under some operation"

What would Sipser do?

- Give the "proof idea": Explain how to take machine(s) recognizing regular language(s) and create a new machine
- Explain in a few sentences why the construction works
- Give a formal description of the construction
- No need to formally prove that the construction works