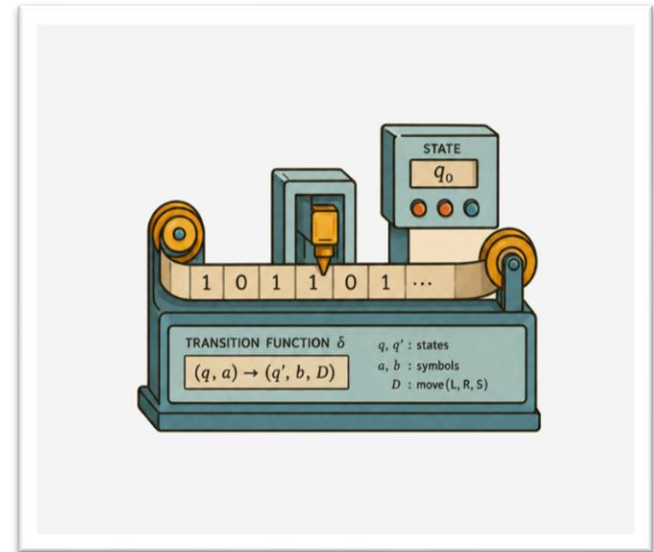


BU CS 332 – Theory of Computation

Lecture 4:

- Nondeterministic Finite Automata
- NFAs vs. DFAs
- Quiz



Alexander Poremba

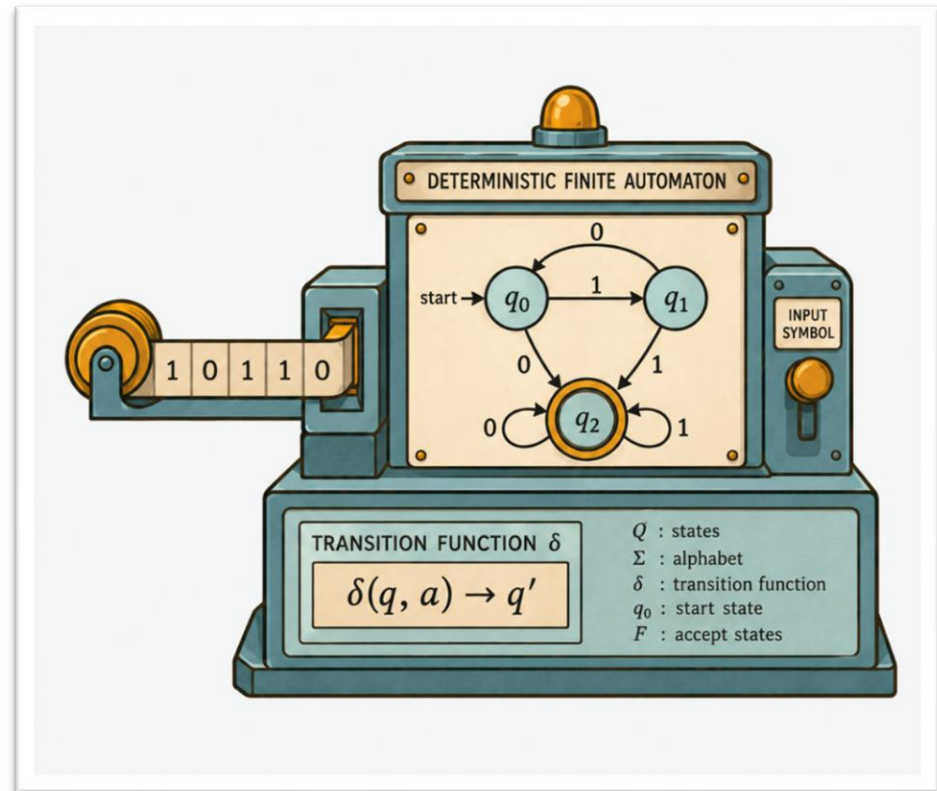
September 15, 2026

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

Today: Nondeterministic Finite Automata (NFAs)

Nondeterministic Finite Automata



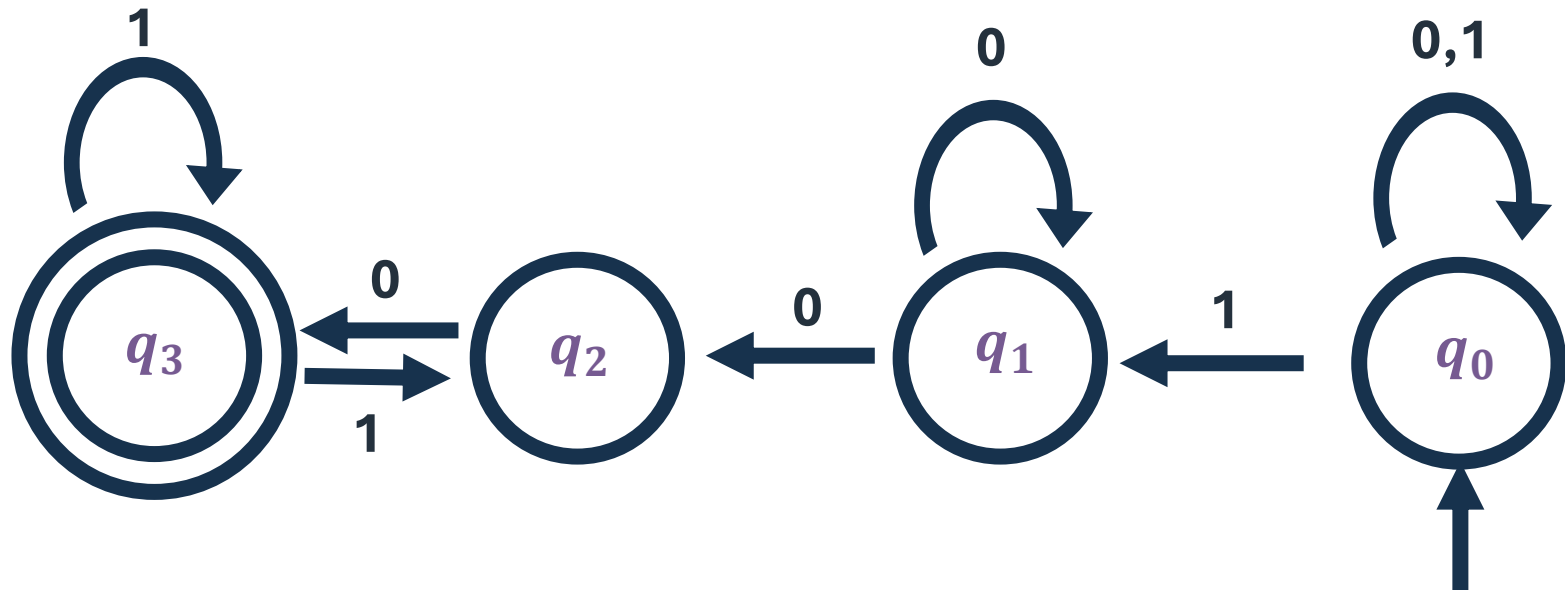
Nondeterminism

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

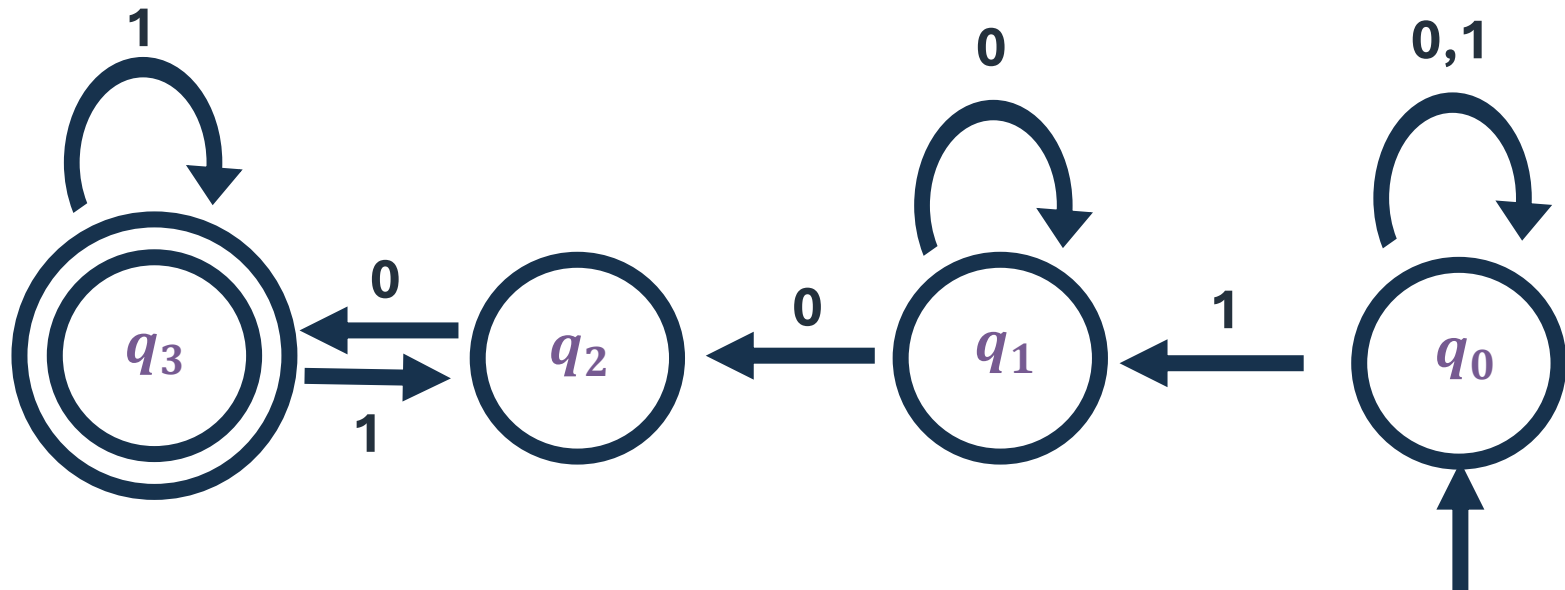
- Next symbol may cause an NFA to “branch” into multiple possible computations
- Next symbol may cause NFA’s computation to fail to enter any state at all

Nondeterminism



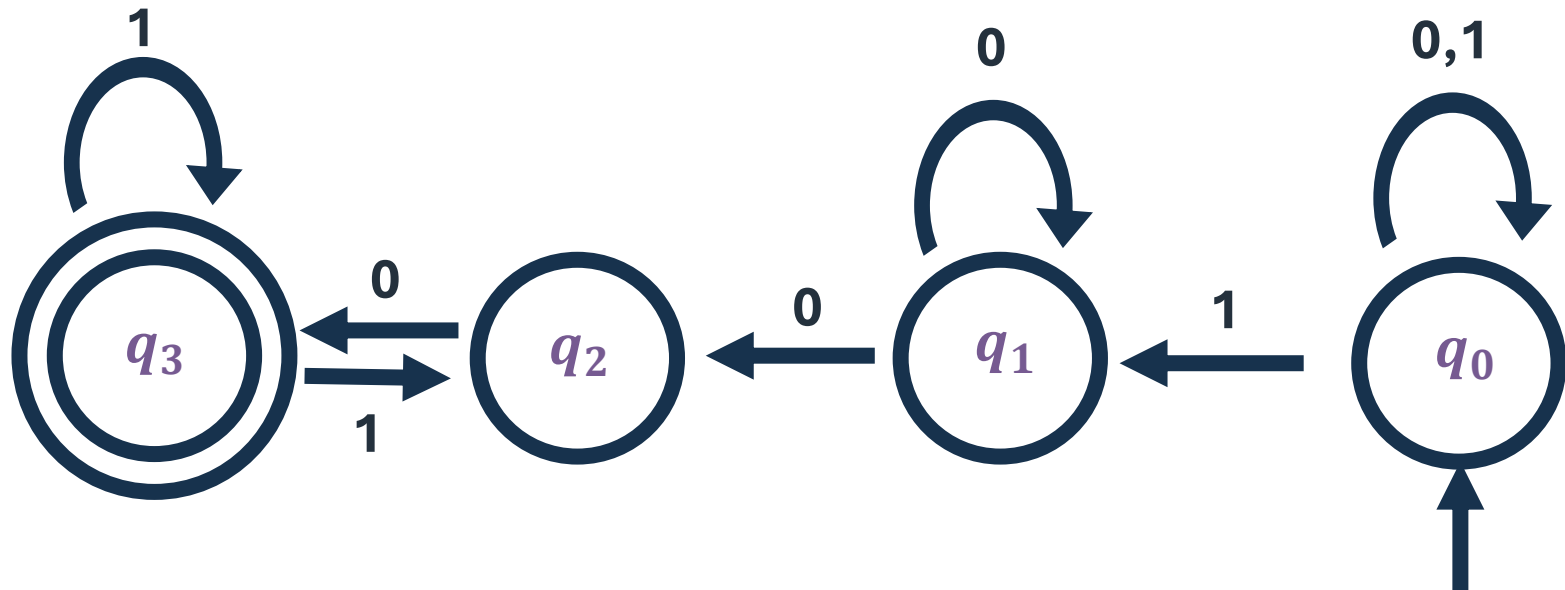
A **Nondeterministic Finite Automaton** (NFA) accepts if there ***exists*** a way to make it reach an accept state.

Nondeterminism



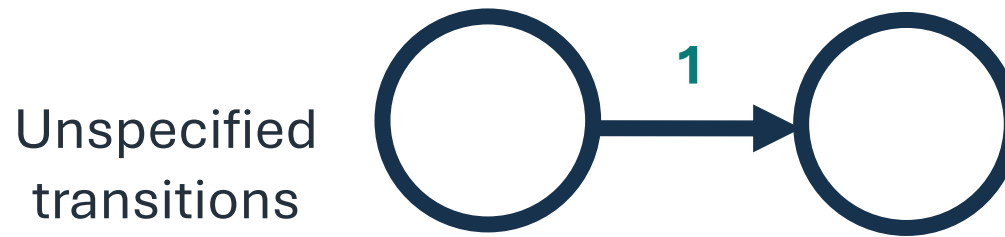
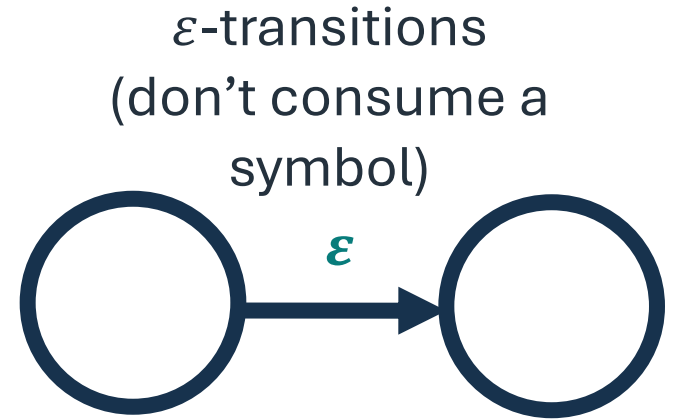
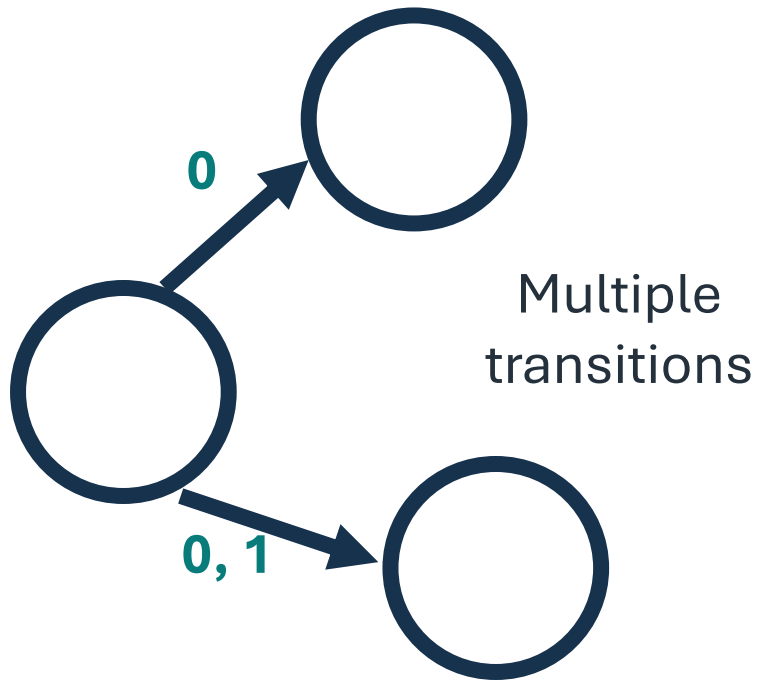
Question: Does this NFA accepts input 1100?

Nondeterminism

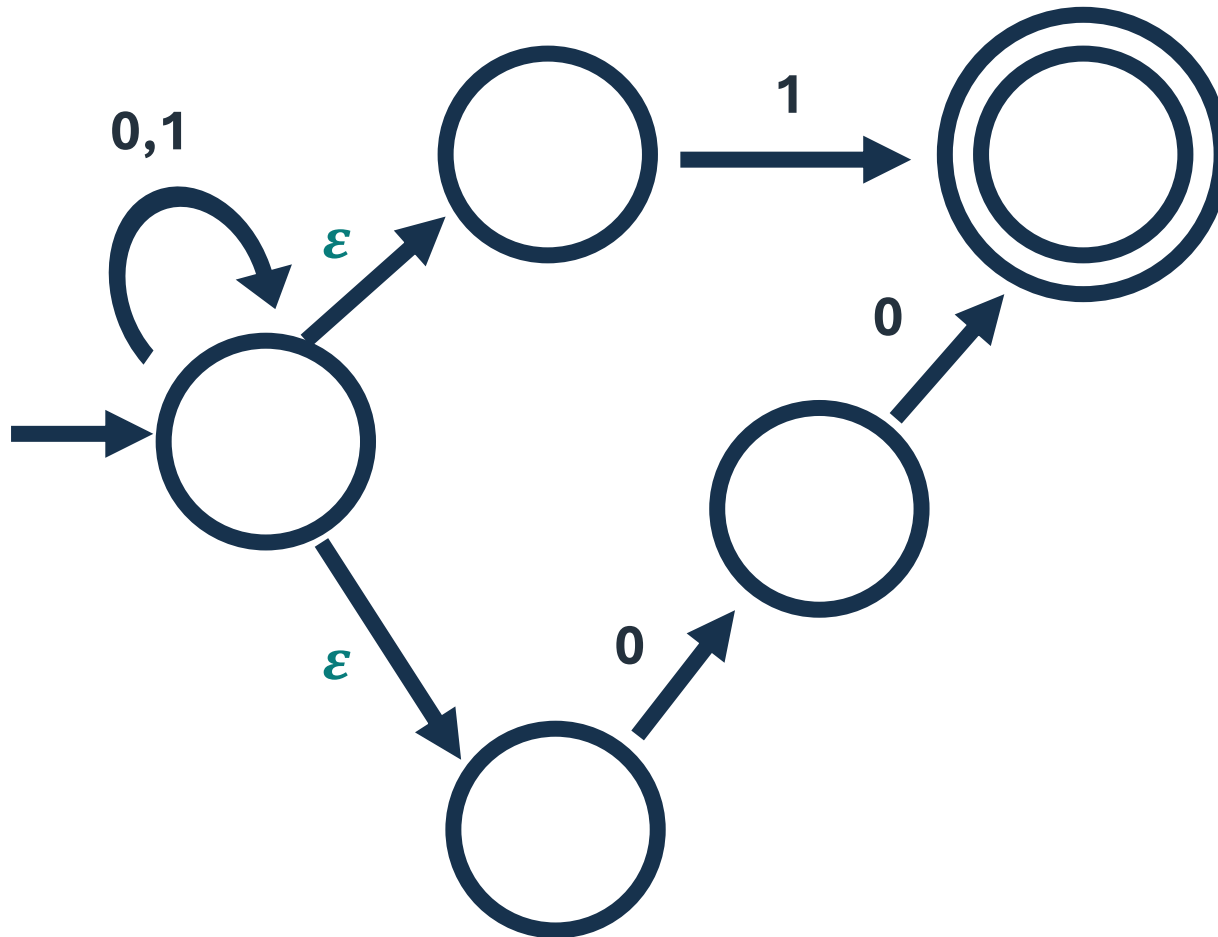


Question: What about 11? Does it accept this string?

Some special transitions (allowed for NFAs but not DFAs)

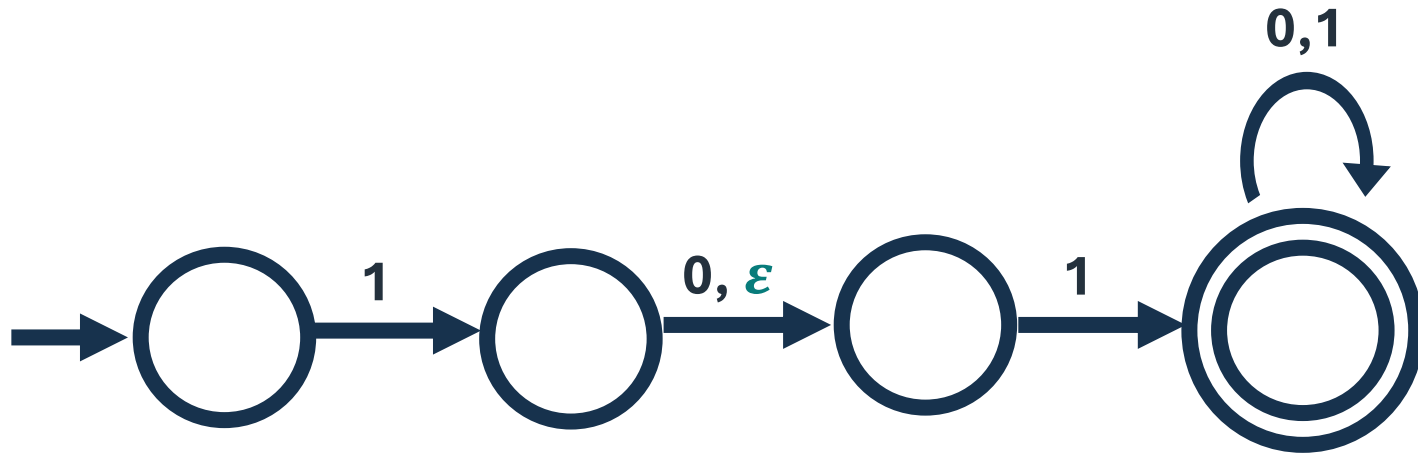


Example



$L(M) =$

Question: Which language does this NFA recognize?



- $L(N) =$
- 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 an NFA

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

Q is the set of states

Σ is the alphabet

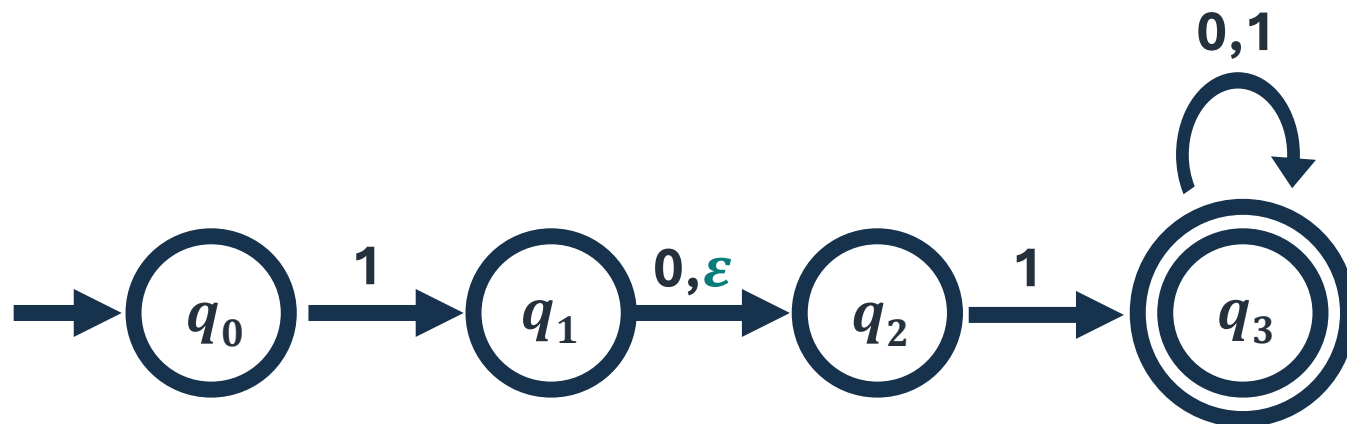
$\delta: Q \times \Sigma_\epsilon \rightarrow P(Q)$ is the transition function

$q_0 \in Q$ 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



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

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

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

$$F = \{q_3\}$$

$$\delta(q_0, 0) =$$

$$\delta(q_0, 1) =$$

$$\delta(q_1, \epsilon) =$$

$$\delta(q_2, 0) =$$

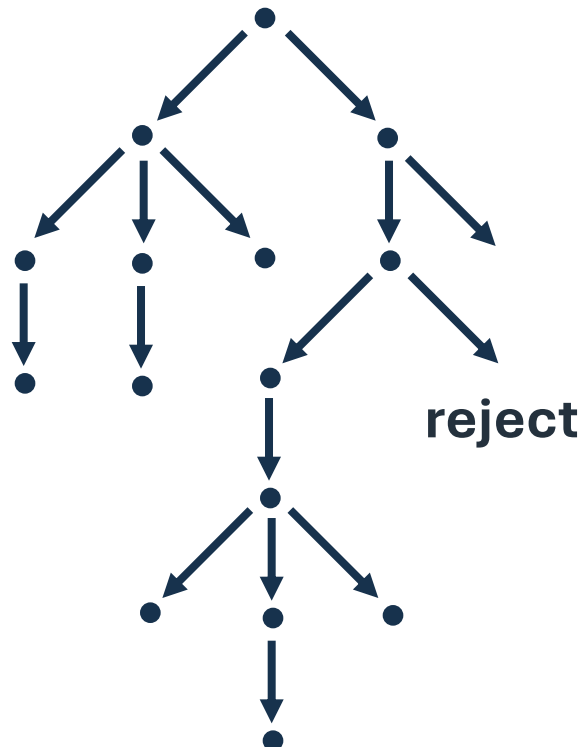
Nondeterminism

Deterministic Computation



accept or reject

Nondeterministic Computation



reject

accept

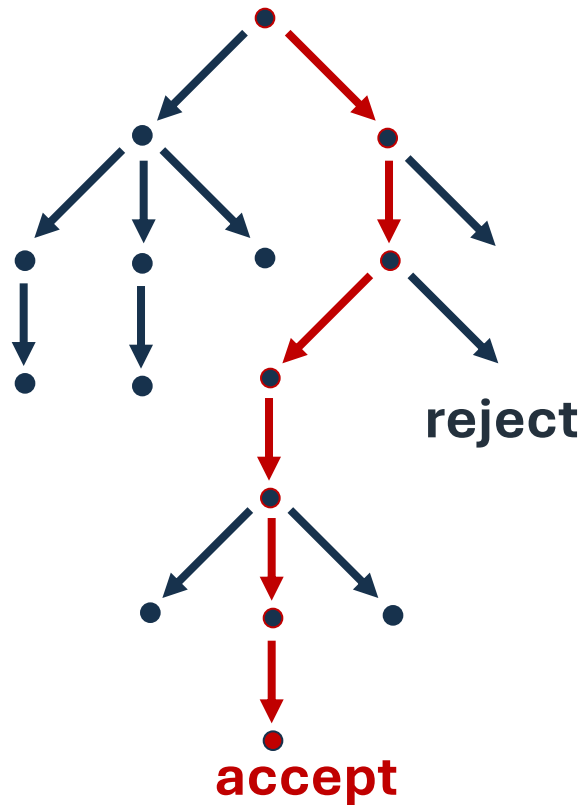
Nondeterminism

Deterministic Computation



accept or reject

Nondeterministic Computation



*Ways to think about
nondeterminism*

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

Why study NFAs?

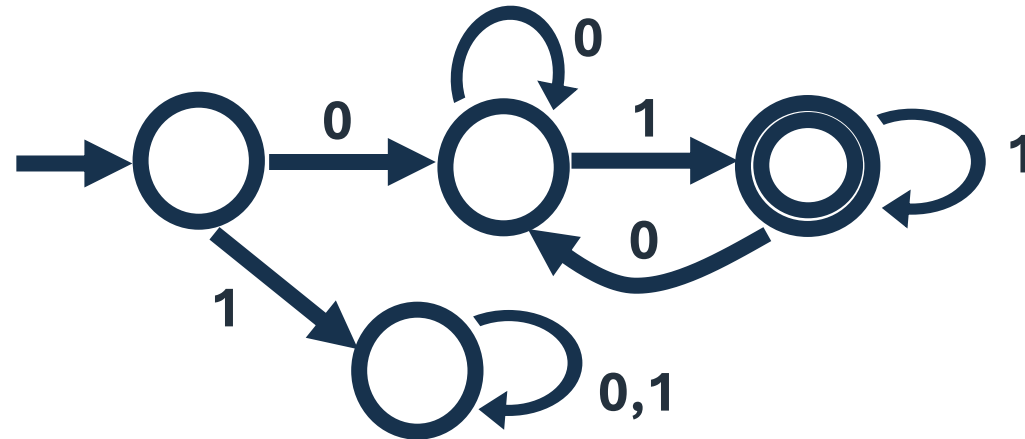
- Not really a realistic model of computation: Real computing devices can't really try many possibilities in parallel

But:

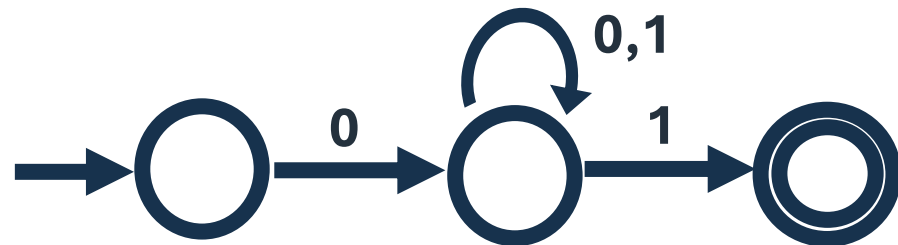
- NFAs can be simpler than DFAs
- Useful for understanding power of DFAs/regular languages
- Lets us study “nondeterminism” as a resource
(cf. P vs. NP)

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

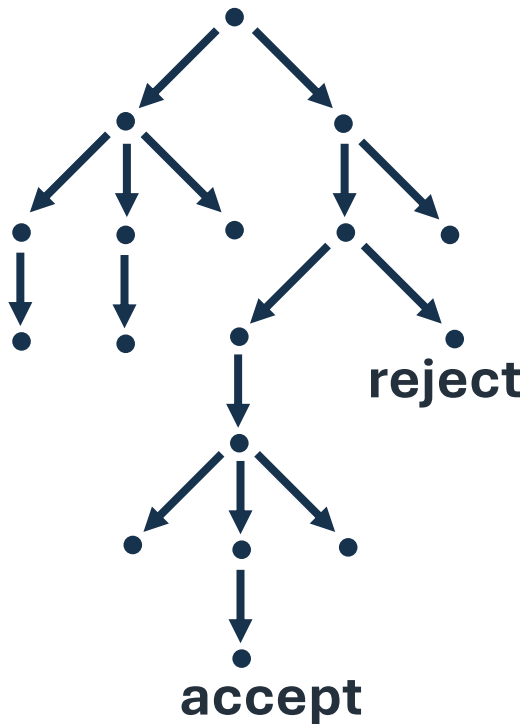
Theorem: For every NFA N , there is a DFA M such that $L(M) = L(N)$

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 $L(N)$

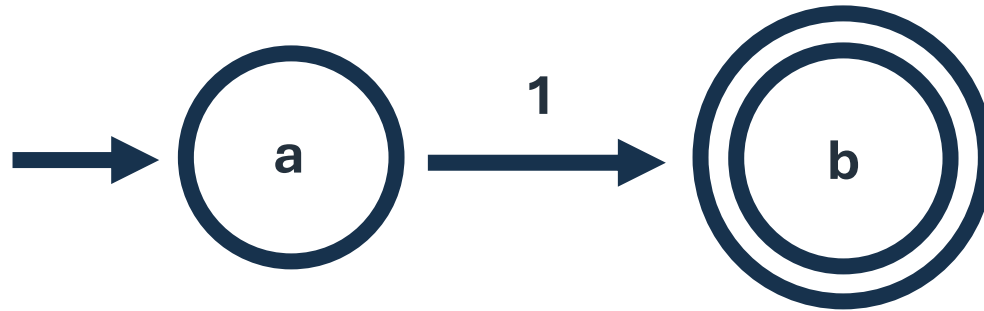


Intuition: Run all threads of N in parallel, maintaining the set of states where all threads are.

Formally: $Q' = P(Q)$

“The Subset Construction”

NFA → DFA Example



Subset Construction

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

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

$$Q' = P(Q)$$

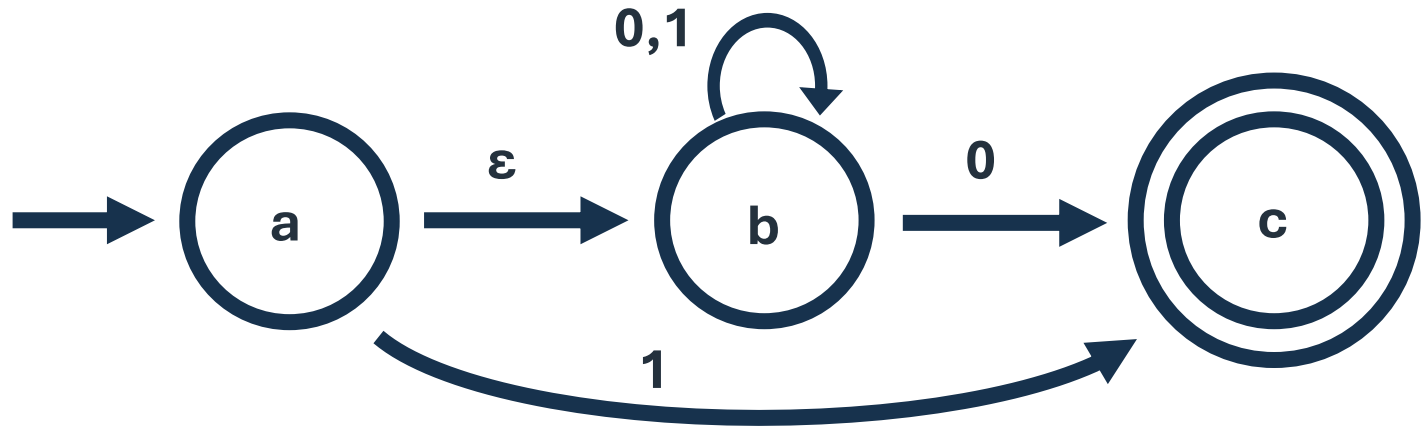
$$\delta' : Q' \times \Sigma \rightarrow Q'$$

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

$$q_0' = \{q_0\}$$

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

NFA → DFA Example



Proving the Construction Works

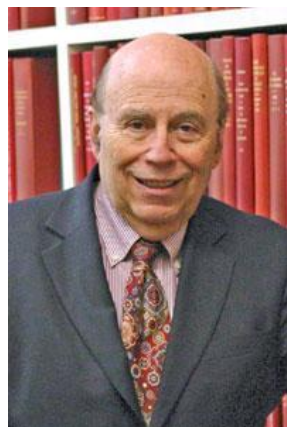
Claim: For every string w , running M on w leads to state

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

Their joint paper introduced the idea of **nondeterministic** machines, which has been enormously influential.

Question: The Catch with NFA $\rightarrow$ DFA?

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).

Quiz time!

