BU CS 332 – Theory of Computation

https://forms.gle/p6SmhquxaKDe94P39

Lecture 4:

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

Reading:

Sipser Ch 1.1-1.2

HW1 + HW0 self-assessment due tomorrow @ 11:59PM

Diptaksho's Wednesday office hour: 4:30-6PM (SOC B61)

Test 1: Wednesday 2/21 (not Tue)

Mark Bun

January 31, 2024

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)

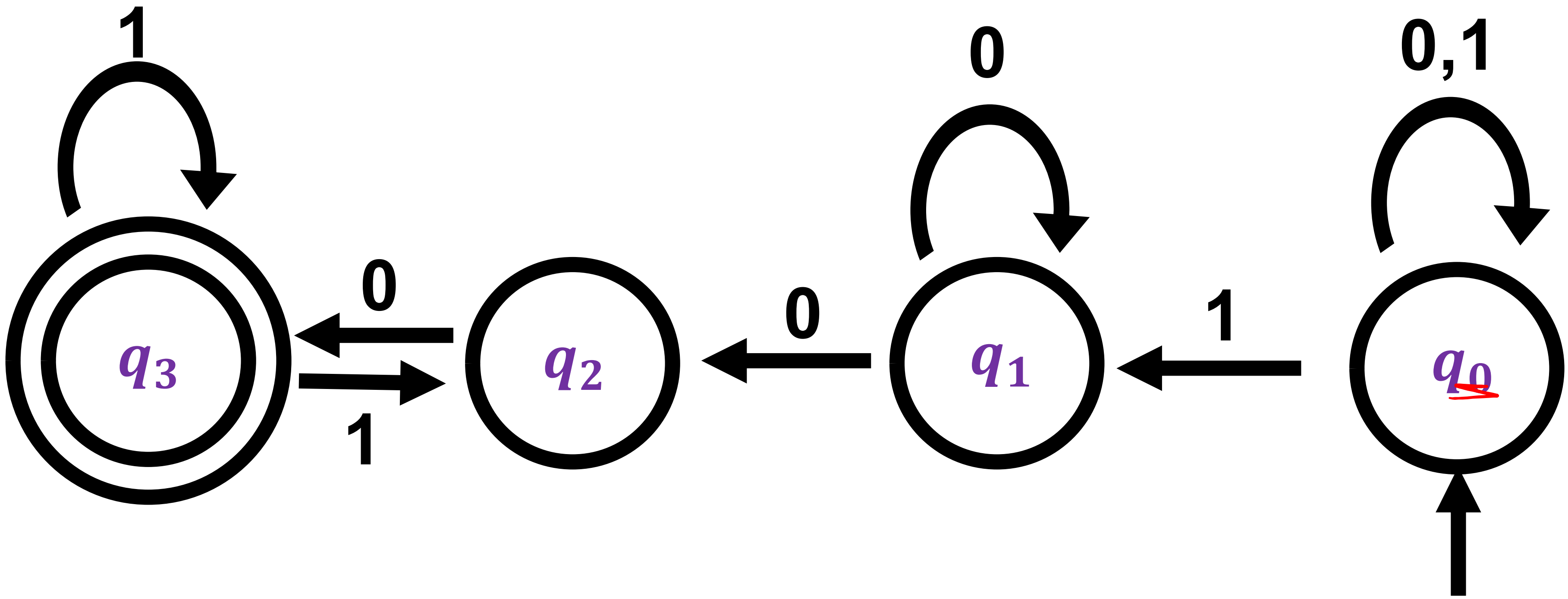
Nondeterminism

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

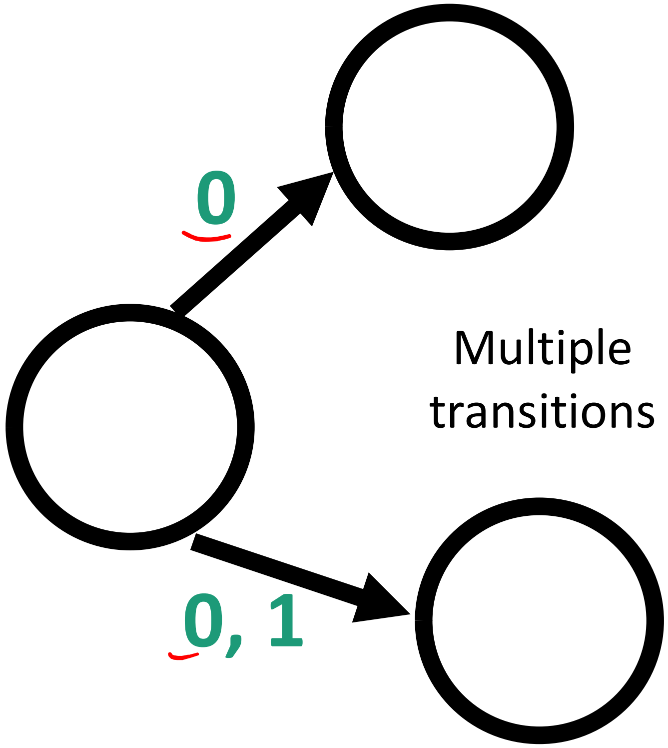
Nondeterminism



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

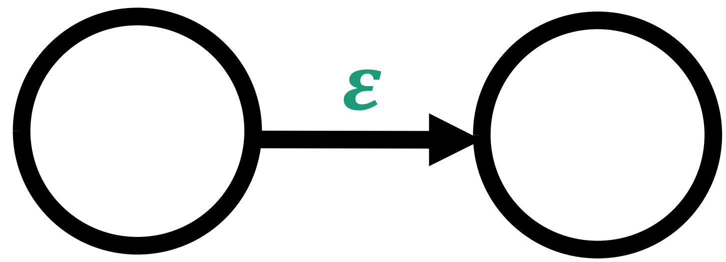
<u>Ex.</u> This NFA accepts input 1100, but does not accept input 11

Some special transitions

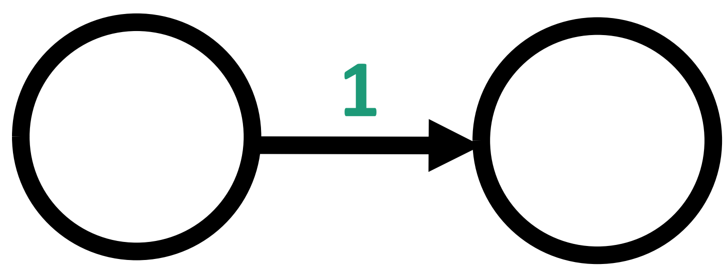


Multiple transitions

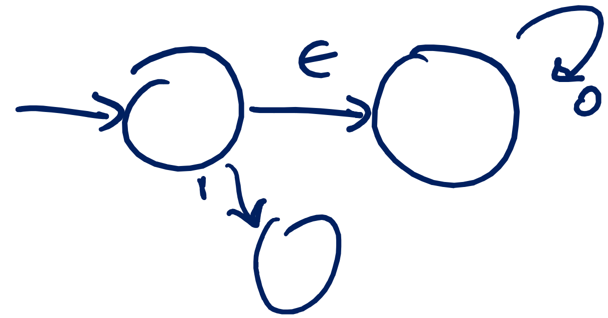
ε-transitions
(don’t consume a symbol)

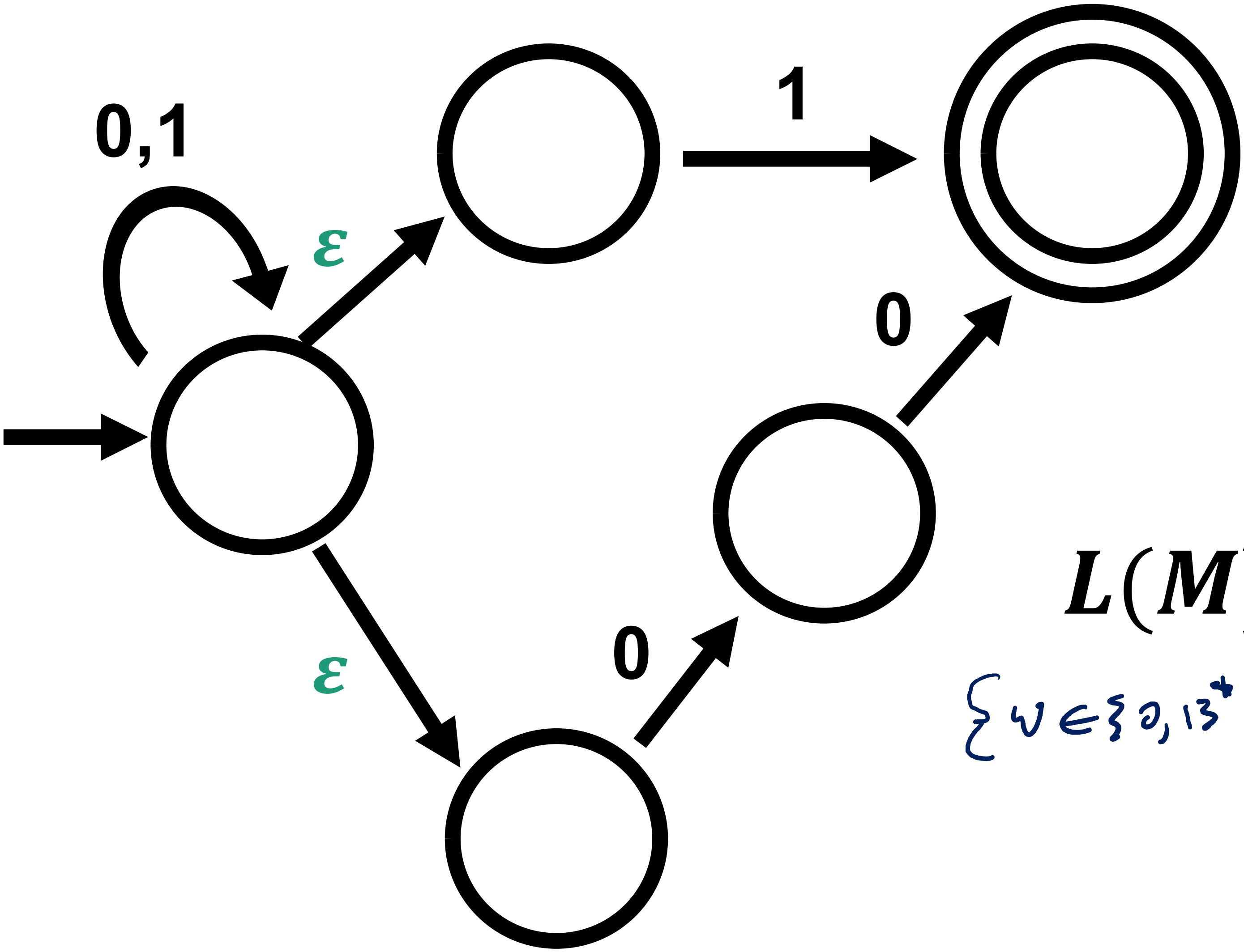


No transition



Example

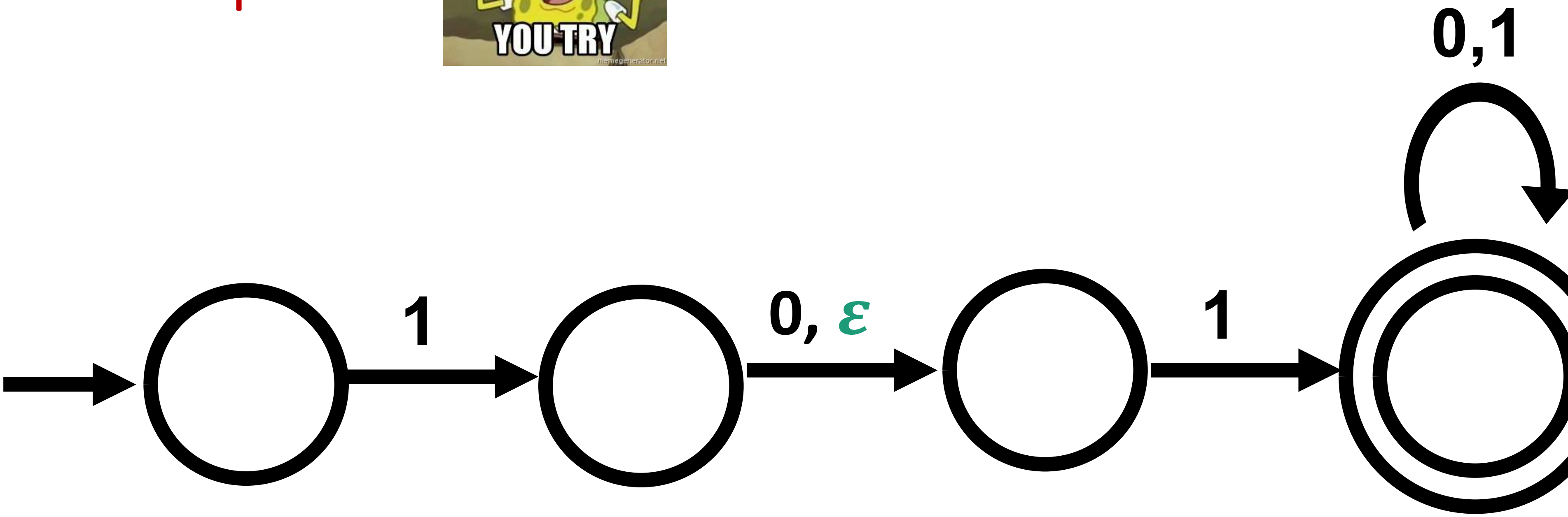




$L(M) =$

$\{ w \in \{0,1\}^* \mid w \text{ ends in } 1 \text{ or in } 00 \}$

Example



$\boldsymbol{L(N)} =$

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

$P(Q) =$ power set of Q
$= \{R \mid R \subseteq Q\}$

- Q is the set of states
- Σ is the alphabet

curr. state   next symbol   set of possible states

- $\boxed{\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)}$ is the transition function

$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

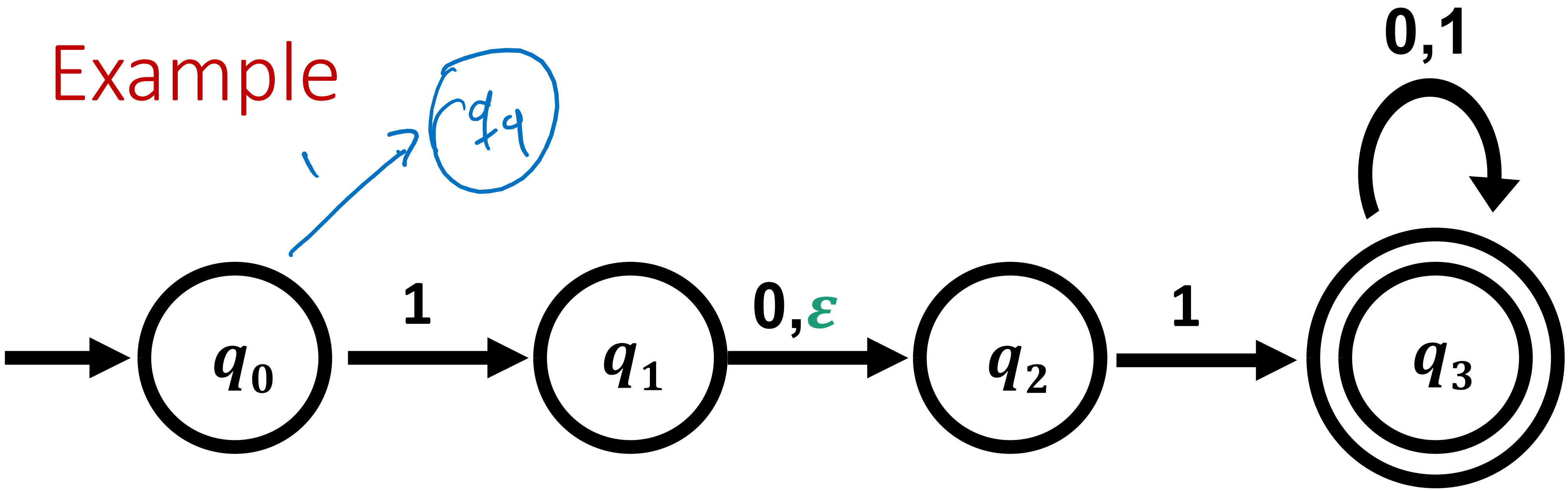
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

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

current state   next symbol   next state

cf: For DFAs $\delta : Q \times \Sigma \rightarrow Q$

Example



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

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

$\Sigma = \{0, 1\}$ $\quad \Sigma_\varepsilon = \{0, 1, \varepsilon\}$

$F = \{q_3\}$

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

$\delta(q_0, 1) = \{q_1, q_4\}$

$\delta(q_1, \varepsilon) = \{q_2\}$

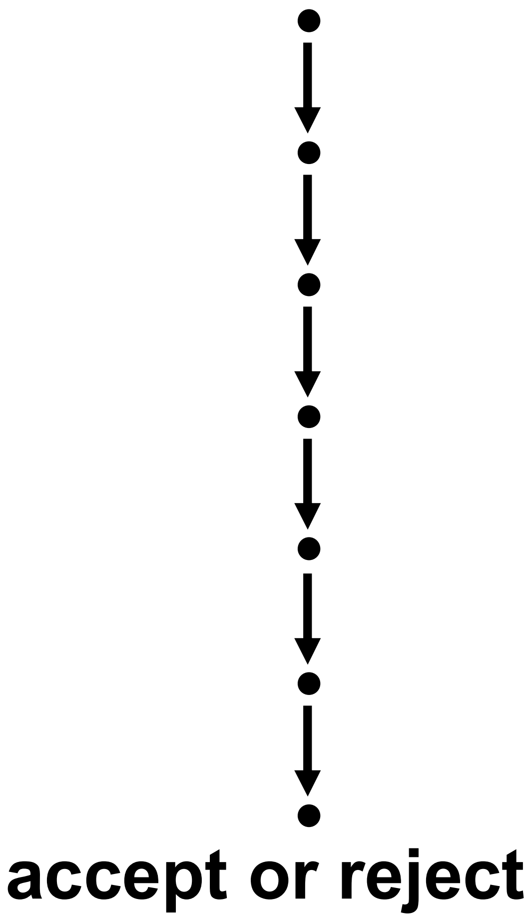
$\delta(q_2, 0) = \phi$

$\delta(q_3, 1) = \{q_3\}$

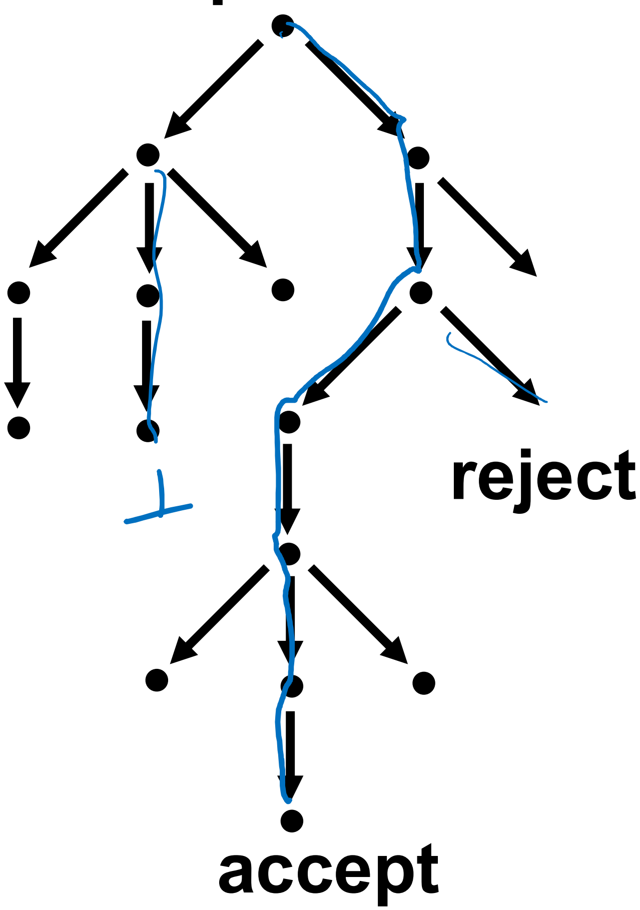
$\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$

Nondeterminism

Deterministic Computation



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:

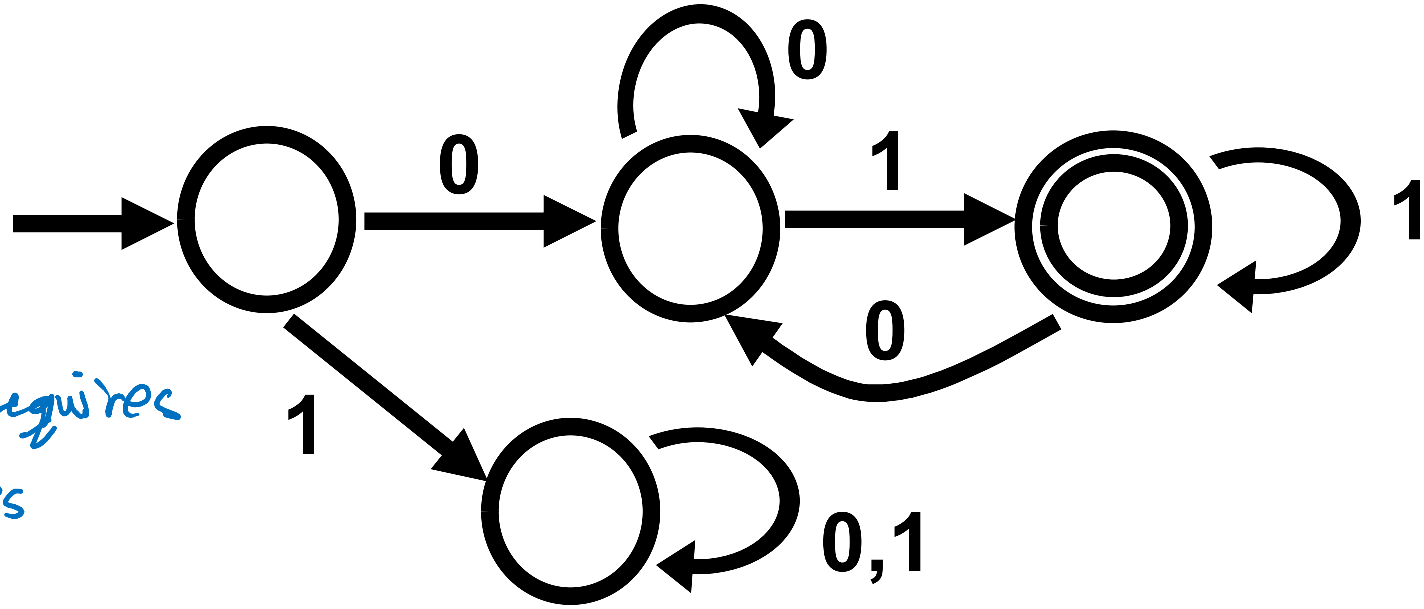
- NFAs can be simpler than DFAs
- Useful for understanding power of DFAs/regular languages
- Lets us study "nondeterminism" as a resource

  (cf. P vs. NP)

NFAs can be simpler than DFAs

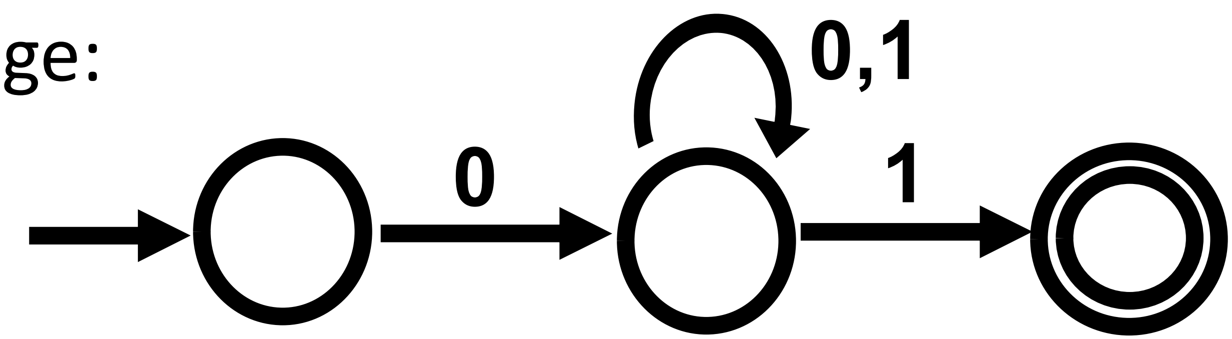
A DFA that recognizes the language

$\{w \mid w \text{ starts with } 0 \text{ and ends with } 1\}$:



Every DFA for this language requires ≥4 states

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

Intuitively true, but take care using formal definitions

Regular languages $\subseteq$ { languages recognizable by NFAs }

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

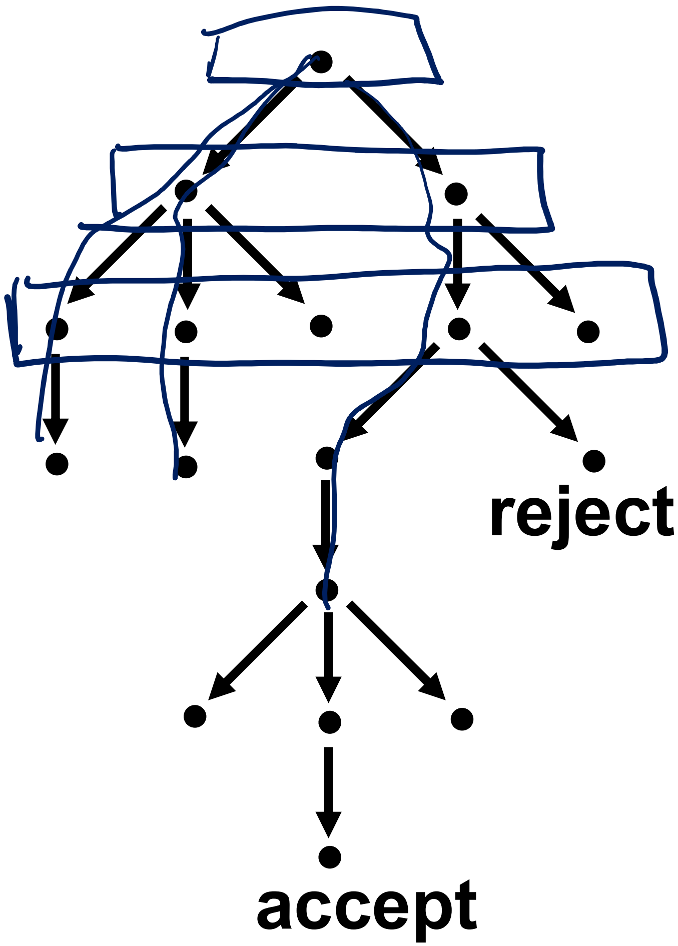
$\supseteq$

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

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



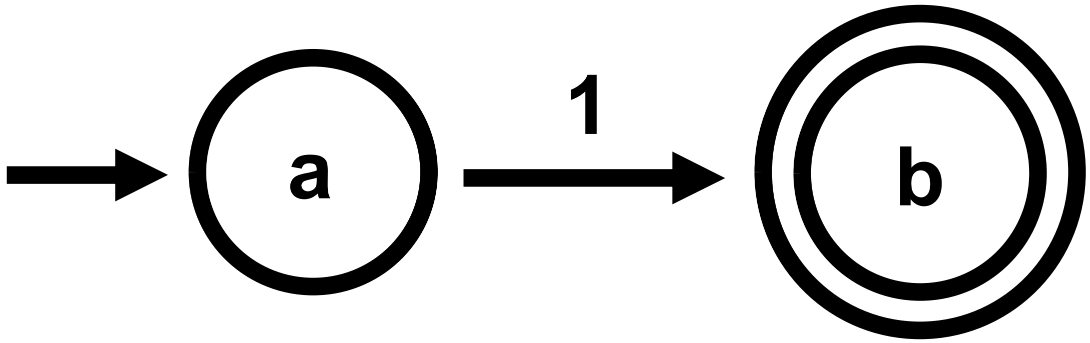
Intuition: Run all threads of N in parallel, maintaining the set of states where all threads are.

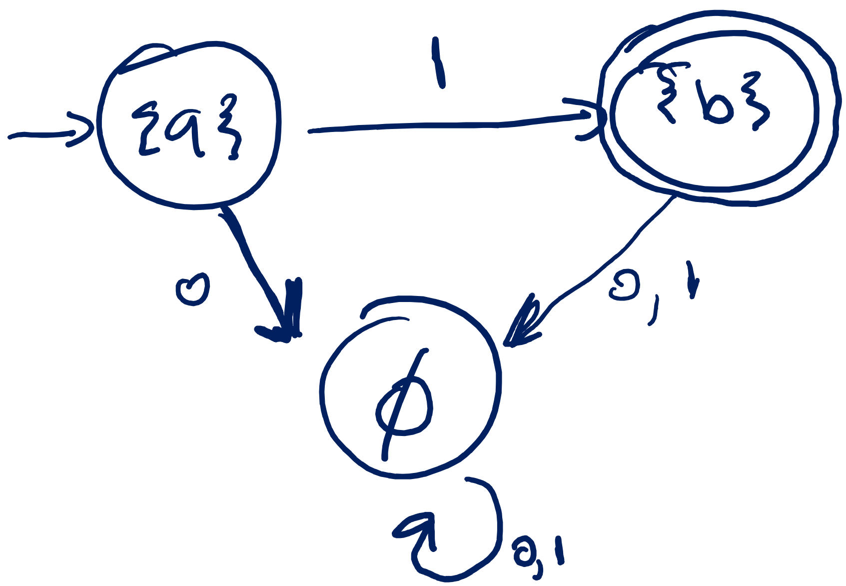
Formally: $Q' = P(Q)$

"The Subset Construction"

NFA → DFA Example

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





Subset Construction (Formally, first attempt)

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

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$ s.t. $L(M) = L(N)$

$Q' = P(Q) = \{R \mid R \subseteq Q\}$

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

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

$q_0' = \{q_0\}$

$F' = \{R \subseteq Q \mid R \cap F \neq \phi\}$

$= \{R \in P(Q) \mid \exists\, q \in F \text{ s.t. } q \in R\}$

Subset Construction (Formally, for real)

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

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

$\bigcup_{q \in T} \delta(q, \epsilon)$

$E(T) = \{ q \mid q$ is reachable from some state in T by following ε-moves $\}$

set of states of N

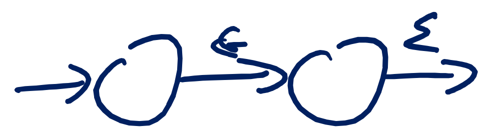
$T \subseteq Q$

$$Q' = P(Q)$$

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

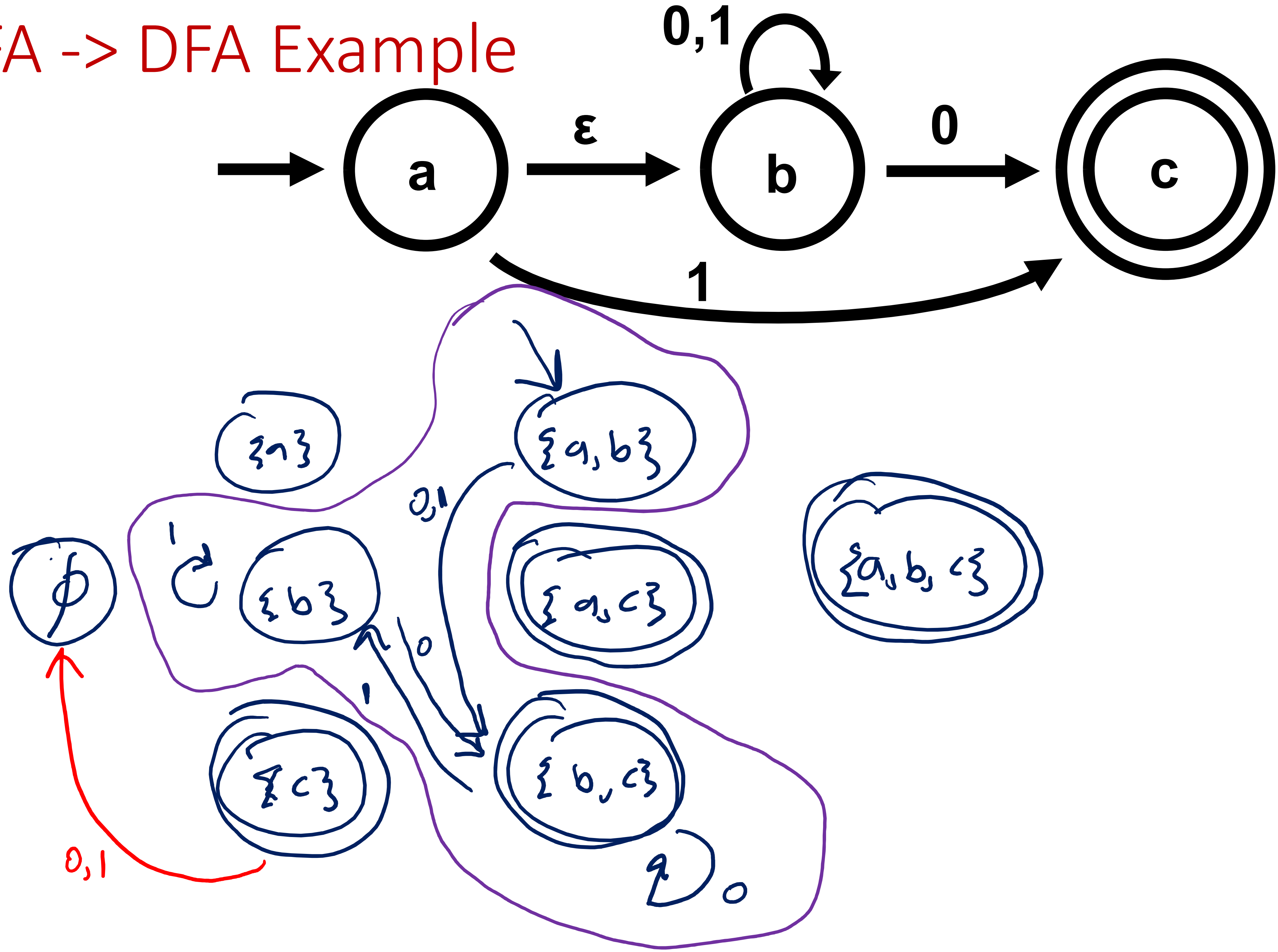
$\delta'(R, \sigma) = \bigcup_{r \in R} E(\delta(r, \sigma))$ for all $R \subseteq Q$ and $\sigma \in \Sigma$.

$$q_0' = E(\{q_0\})$$



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

NFA -> DFA Example



Proving the Construction Works

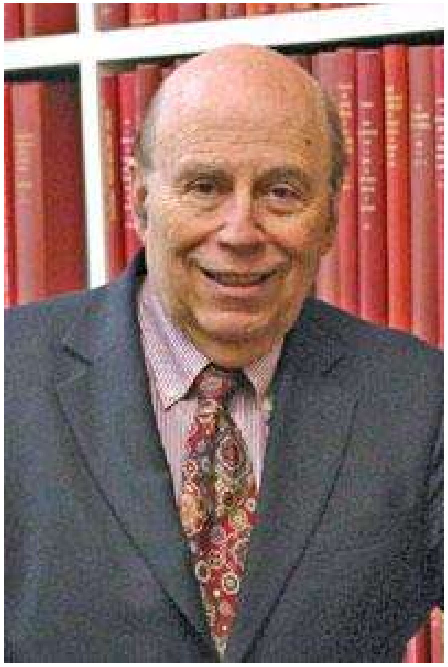
Claim: For every string w, running M on w leads to state

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

Proof idea: By induction on $|w|$

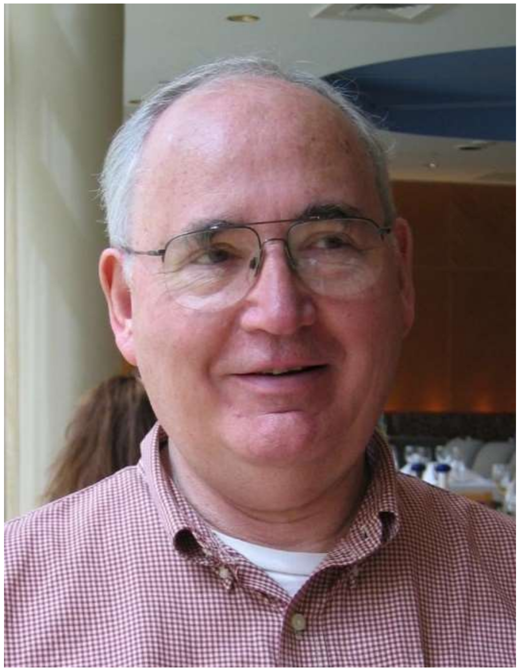
Historical Note

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 $= |P(Q)|$ where $|Q| = s.$

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).