Nondeterministic Finite Automata

Arjun Chandrasekhar

Nondeterministic Finite Automata

Nondeterministic Finite Automata

- A DFA is *deterministic*. For each state/symbol combination, there is exactly one transition defined.

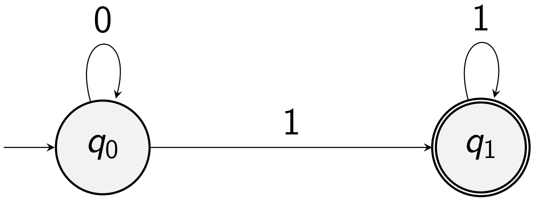
Nondeterministic Finite Automata

- A DFA is *deterministic*. For each state/symbol combination, there is exactly one transition defined.
- An **nondeterministic finite automaton (NFA)** is like a DFA, except a state/symbol pair may have any number of transitions defined for it (0, 1, 2, ...).

Nondeterministic Finite Automata

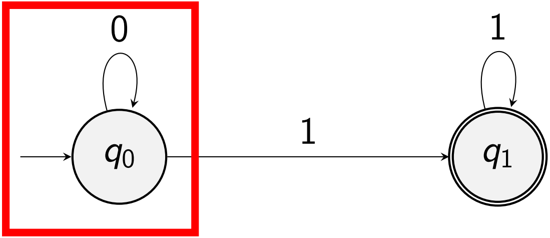
- A DFA is *deterministic*. For each state/symbol combination, there is exactly one transition defined.
- An **nondeterministic finite automaton (NFA)** is like a DFA, except a state/symbol pair may have any number of transitions defined for it (0, 1, 2, ...).
- Can also have ϵ transitions which let you change states without reading a symbol.

Nondeterministic Finite Automata



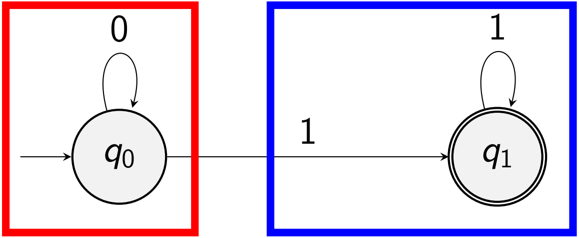
Nondeterministic Finite Automata

Can only read 0s here



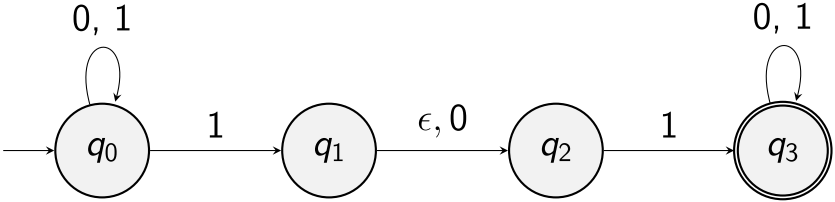
Nondeterministic Finite Automata

Can only read 0s here



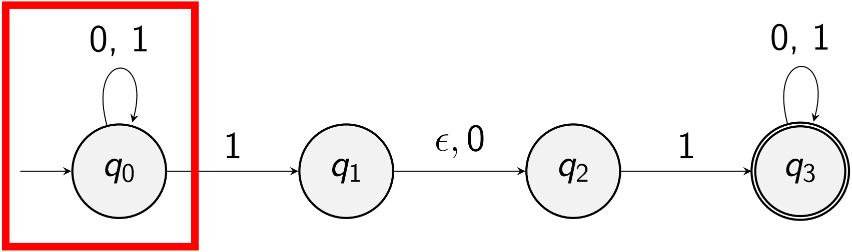
Can only read 1s here

Nondeterministic Finite Automata



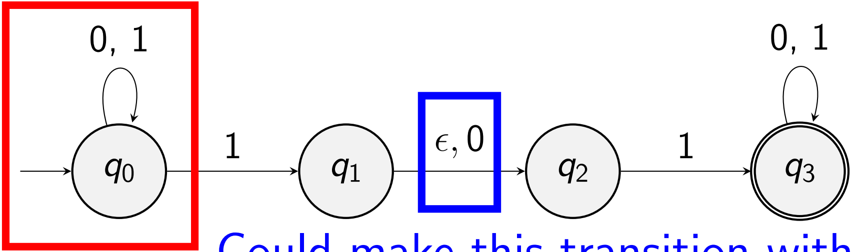
Nondeterministic Finite Automata

Multiple transitions for
the same symbol



Nondeterministic Finite Automata

Multiple transitions for
the same symbol



Could make this transition with
or without reading a symbol

Computation on an NFA

Computation on an NFA

- Start in the start state

Computation on an NFA

- Start in the start state
- Scan symbols one-by-one

Computation on an NFA

- Start in the start state
- Scan symbols one-by-one
- For each symbol σ scanned:

Computation on an NFA

- Start in the start state
- Scan symbols one-by-one
- For each symbol σ scanned:
  - Go to *one of* the possible arrows with the label σ

Computation on an NFA

- Start in the start state
- Scan symbols one-by-one
- For each symbol σ scanned:
  - Go to *one of* the possible arrows with the label σ
  - If no arrows have the label σ the computation *dies*

Computation on an NFA

- Start in the start state
- Scan symbols one-by-one
- For each symbol σ scanned:
  - Go to *one of* the possible arrows with the label σ
  - If no arrows have the label σ the computation *dies*
  - The NFA can behave in different ways on the same input string!

Computation on an NFA

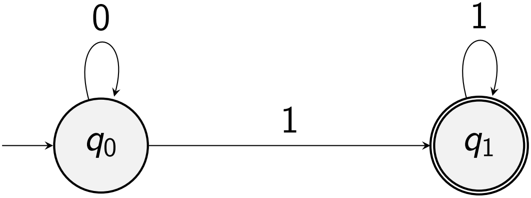
- Start in the start state
- Scan symbols one-by-one
- For each symbol σ scanned:
  - Go to *one of* the possible arrows with the label σ
  - If no arrows have the label σ the computation *dies*
  - The NFA can behave in different ways on the same input string!
- At any point the NFA may take an ϵ transition without consuming a character

Computation on an NFA

- Start in the start state
- Scan symbols one-by-one
- For each symbol σ scanned:
  - Go to *one of* the possible arrows with the label σ
  - If no arrows have the label σ the computation *dies*
  - The NFA can behave in different ways on the same input string!
- At any point the NFA may take an ϵ transition without consuming a character
- The NFA accepts if after reading all the characters, and taking any desired ϵ transitions, it is in an accept state

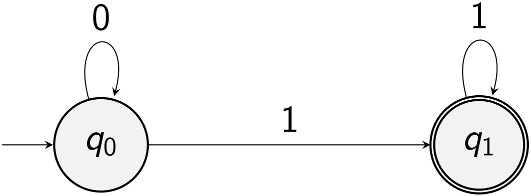
Computation on an NFA

What happens on inputs: 000, 010, 101, 011?



Computation on an NFA

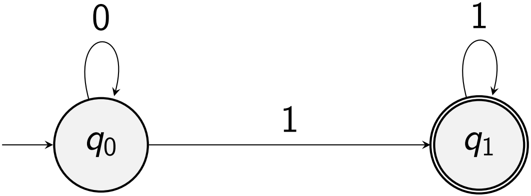
What happens on inputs: 000, 010, 101, 011?



$$q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \rightarrow \text{REJECT}$$

Computation on an NFA

What happens on inputs: 000, 010, 101, 011?

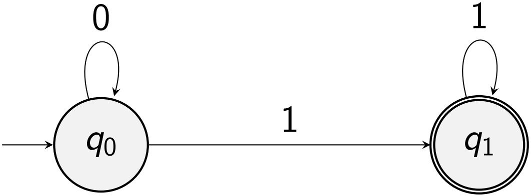


$$q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \rightarrow \text{REJECT}$$

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} \text{DIES}$$

Computation on an NFA

What happens on inputs: 000, 010, 101, 011?



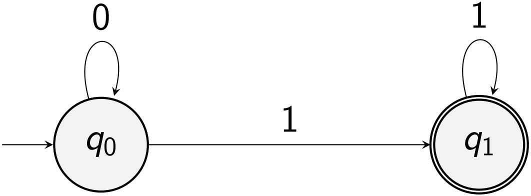
$$q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \rightarrow \text{REJECT}$$

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} \text{DIES}$$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} \text{DIES}$$

Computation on an NFA

What happens on inputs: 000, 010, 101, 011?



$$q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \rightarrow \text{REJECT}$$

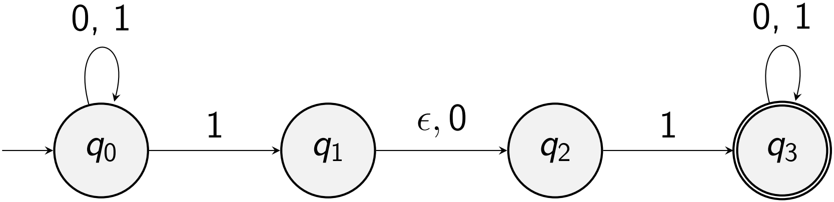
$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} \text{DIES}$$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{0} \text{DIES}$$

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \rightarrow \text{ACCEPT}$$

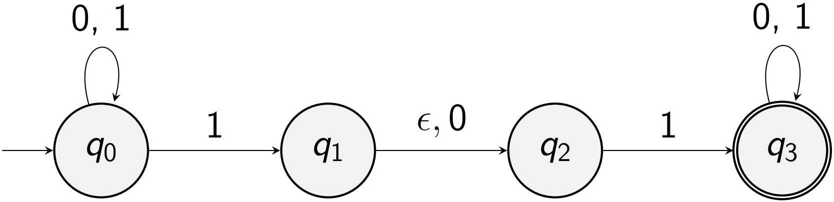
Computation on an NFA

What happens on input 111?



Computation on an NFA

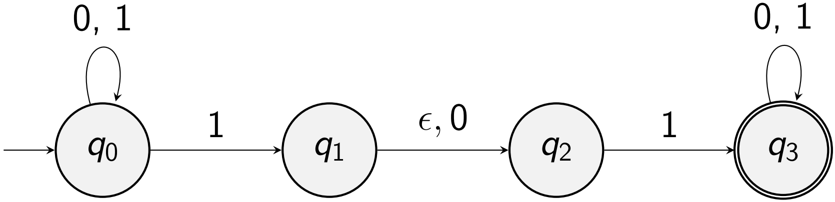
What happens on input 111?



$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \rightarrow \text{REJECT}$$

Computation on an NFA

What happens on input 111?

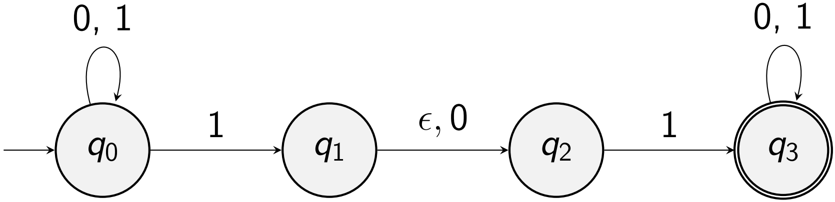


$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \rightarrow \text{REJECT}$$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} \text{DIES}$$

Computation on an NFA

What happens on input 111?



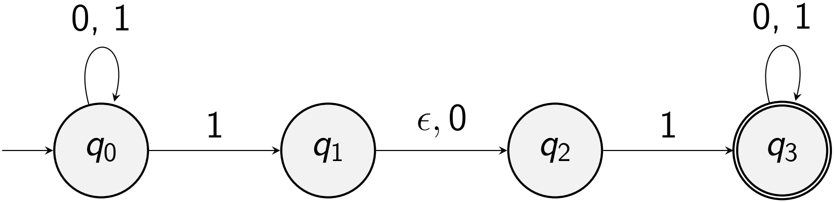
$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \rightarrow \text{REJECT}$$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} \text{DIES}$$

$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{\epsilon} q_2 \rightarrow \text{REJECT}$$

Computation on an NFA

What happens on input 111?



$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \rightarrow \text{REJECT}$$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} \text{DIES}$$

$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{\epsilon} q_2 \rightarrow \text{REJECT}$$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_3 \rightarrow \text{ACCEPT}$$

NFA Formal Definition

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA
- Σ the alphabet of (non-ϵ) characters that the NFA can read

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA
- Σ the alphabet of (non-ϵ) characters that the NFA can read
- q_s: the starting state

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA
- Σ the alphabet of (non-ϵ) characters that the NFA can read
- q_s: the starting state
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ - the transition function

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA
- Σ the alphabet of (non-ϵ) characters that the NFA can read
- q_s: the starting state
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ - the transition function
  - **Input:** Current state & next symbol (or ϵ)

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA
- Σ the alphabet of (non-ϵ) characters that the NFA can read
- q_s: the starting state
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$ - the transition function
  - **Input:** Current state & next symbol (or ϵ)
  - **Output:** *Set* of possible next states (could be empty)

NFA Formal Definition

Def: A **Nondeterministic finite automate (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$

- Q: The set of states in the NFA
- Σ the alphabet of (non-ϵ) characters that the NFA can read
- q_s: the starting state
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ - the transition function
  - **Input:** Current state & next symbol (or ϵ)
  - **Output:** *Set* of possible next states (could be empty)
- F - the set of accept states

NFA Accepting Computation

NFA Accepting Computation

- An NFA can do many different things on the same string

NFA Accepting Computation

- An NFA can do many different things on the same string
  - It may be capable of both accepting *and* rejecting the same string!

NFA Accepting Computation

- An NFA can do many different things on the same string
  - It may be capable of both accepting *and* rejecting the same string!
- What does it mean for an NFA to accept a string?

NFA Accepting Computation

- An NFA can do many different things on the same string
  - It may be capable of both accepting *and* rejecting the same string!
- What does it mean for an NFA to accept a string?
- Informally, an NFA accepts a string w if there *exists* a computation path that ends in an accept state

NFA Accepting Computation

- An NFA can do many different things on the same string
  - It may be capable of both accepting *and* rejecting the same string!
- What does it mean for an NFA to accept a string?
- Informally, an NFA accepts a string w if there *exists* a computation path that ends in an accept state
  - Even if every other path rejects and/or dies, just one accepting path is good enough

NFA Accepting Computation

NFA Accepting Computation

An NFA accepts a string $w = w_1 w_2 \ldots w_n$ if:

NFA Accepting Computation

An NFA accepts a string $w = w_1 w_2 \ldots w_n$ if:

1. We can re-write w as $y = y_1 y_2 \ldots y_n$ where each $y_i \in (\Sigma \cup \epsilon)$ (i.e. insert empty ϵ characters into w) and ...

NFA Accepting Computation

An NFA accepts a string $w = w_1 w_2 \ldots w_n$ if:

1. We can re-write w as $y = y_1 y_2 \ldots y_n$ where each $y_i \in (\Sigma \cup \epsilon)$ (i.e. insert empty ϵ characters into w) and ...
2. There *exists* a sequence of states $q_0 q_1 \ldots q_n$ such that...

NFA Accepting Computation

An NFA accepts a string $w = w_1 w_2 \dots w_n$ if:

1. We can re-write w as $y = y_1 y_2 \dots y_n$ where each $y_i \in (\Sigma \cup \epsilon)$ (i.e. insert empty ϵ characters into w) and ...
2. There *exists* a sequence of states $q_0 q_1 \dots q_n$ such that...
   2.1 $q_0 = q_s$ (start in the start state)

NFA Accepting Computation

An NFA accepts a string $w = w_1 w_2 \ldots w_n$ if:

1. We can re-write w as $y = y_1 y_2 \ldots y_n$ where each $y_i \in (\Sigma \cup \epsilon)$ (i.e. insert empty ϵ characters into w) and ...
2. There *exists* a sequence of states $q_0 q_1 \ldots q_n$ such that...
   2.1 $q_0 = q_s$ (start in the start state)
   2.2 $q_i \in \delta(q_{i-1}, y_i)$ for all i (all transitions are valid)

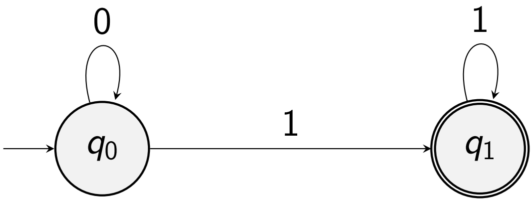
NFA Accepting Computation

An NFA accepts a string $w = w_1 w_2 \ldots w_n$ if:

1. We can re-write w as $y = y_1 y_2 \ldots y_n$ where each $y_i \in (\Sigma \cup \epsilon)$ (i.e. insert empty ϵ characters into w) and ...
2. There *exists* a sequence of states $q_0 q_1 \ldots q_n$ such that...
   2.1 $q_0 = q_s$ (start in the start state)
   2.2 $q_i \in \delta(q_{i-1}, y_i)$ for all i (all transitions are valid)
   2.3 $q_n \in F$ (end in an accept state)

NFA Accepting Computation

Which strings are accepted by this NFA?



A) ϵ (empty string)

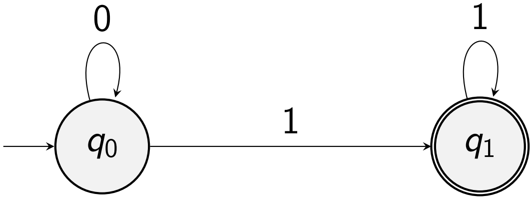
B) 1

C) 010

D) 101

NFA Accepting Computation

Which strings are accepted by this NFA?



A) ϵ (empty string)

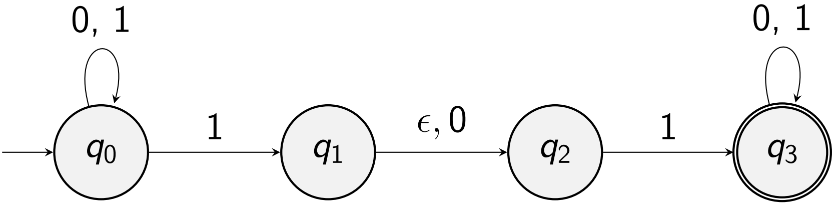
B) 1 ✓

C) 010

D) 101

NFA Accepting Computation

Which strings are accepted by this NFA?



A) ϵ (empty string)

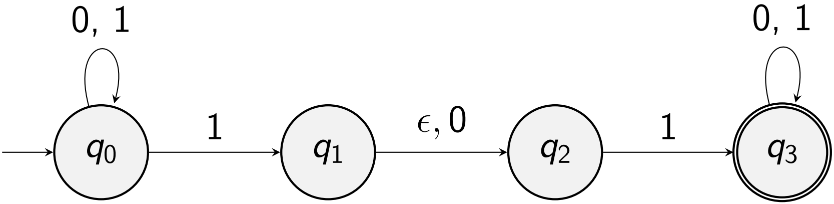
B) 111

C) 111101000

D) 0000

NFA Accepting Computation

Which strings are accepted by this NFA?



A) ϵ (empty string)

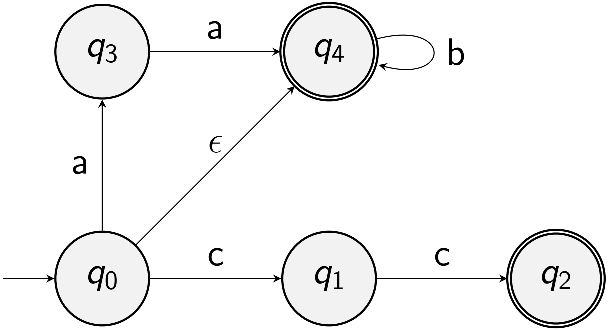
B) 111 ✓

C) 111101000 ✓

D) 0000

NFA Accepting Computation

Which strings are accepted by this NFA?



A) ϵ (empty string)

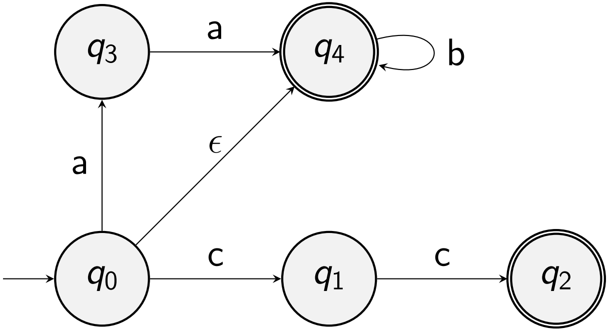
B) *abba*

C) *cc*

D) *ccccccccccccc*

NFA Accepting Computation

Which strings are accepted by this NFA?



A) ϵ (empty string) ✓

B) *abba*

C) *cc* ✓

D) *cccccccccccc*

The Language of an NFA

The Language of an NFA

- Let N be an NFA

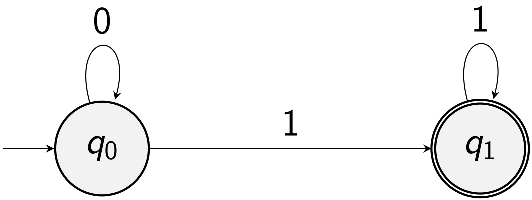
The Language of an NFA

- Let N be an NFA
- The *language of N* is the set of strings that N accepts i.e.

$$L(N) = \{w | N \text{ accepts } w\}$$

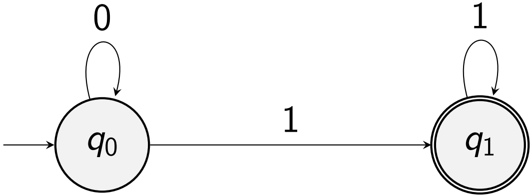
The Language of an NFA

What is the language of this NFA



The Language of an NFA

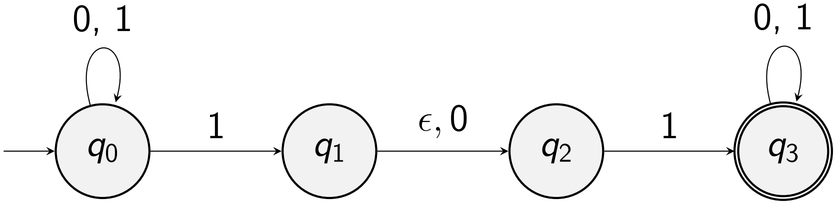
What is the language of this NFA



$L(N) = \{w \mid \text{0s precede 1s, at least one 1}\}$

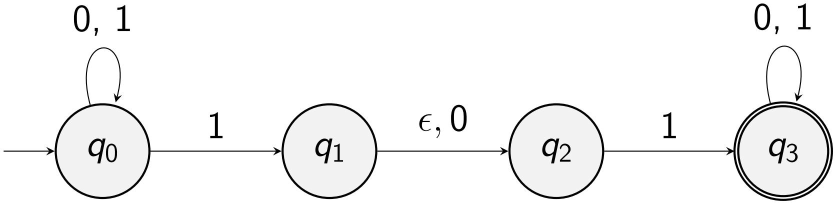
The Language of an NFA

What is the language of this NFA



The Language of an NFA

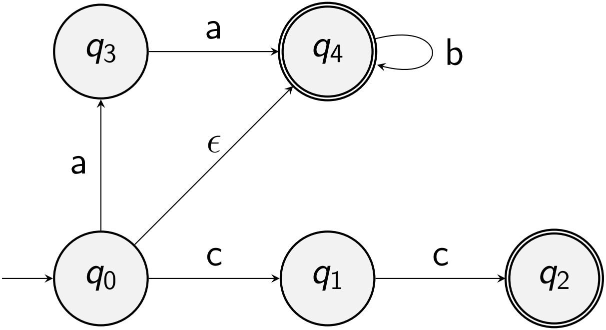
What is the language of this NFA



L(N) = {w | w contains 101 or 11 as a substring}

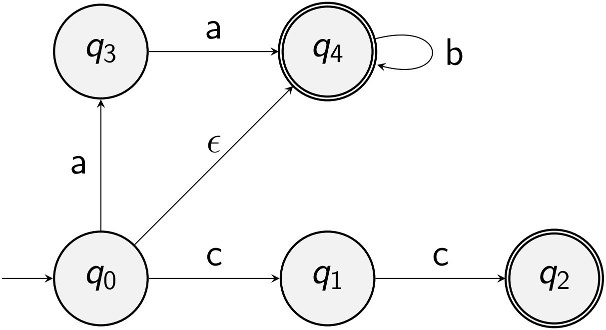
The Language of an NFA

What is the language of this NFA



The Language of an NFA

What is the language of this NFA



L(N) = {w | w has either zero or two a's followed by any number of b's, OR w = cc}

Nondeterminism

Nondeterminism

- As said earlier, an NFA can have many possible computation paths

Nondeterminism

- As said earlier, an NFA can have many possible computation paths
- We can think of nondeterminism in two ways:

Nondeterminism

- As said earlier, an NFA can have many possible computation paths
- We can think of nondeterminism in two ways:
  - The NFA "guesses" which choice will ultimately lead to an accepting state

Nondeterminism

- As said earlier, an NFA can have many possible computation paths
- We can think of nondeterminism in two ways:
  - The NFA "guesses" which choice will ultimately lead to an accepting state
  - The NFA branches/copies itself for each possible choice.

NFAs vs DFAs

NFAs vs DFAs

- Are NFAs more powerful than DFAs?

NFAs vs DFAs

- Are NFAs more powerful than DFAs?
  - That is, are there languages that an NFA can recognize, but a DFA cannot?

NFAs vs DFAs

- Are NFAs more powerful than DFAs?
  - That is, are there languages that an NFA can recognize, but a DFA cannot?
- As it turns out, no! So why study them?

NFAs vs DFAs

- Are NFAs more powerful than DFAs?
  - That is, are there languages that an NFA can recognize, but a DFA cannot?
- As it turns out, no! So why study them?
  - If we want to show a langauge is regular, It is often easier to describe an NFA than a DFA.

NFAs vs DFAs

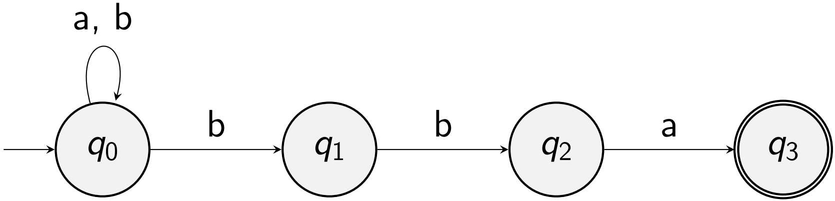
- Are NFAs more powerful than DFAs?
  - That is, are there languages that an NFA can recognize, but a DFA cannot?
- As it turns out, no! So why study them?
  - If we want to show a langauge is regular, It is often easier to describe an NFA than a DFA.
  - If we actually want to be able to recognize the language, then we can automate the conversion of an NFA to a DFA.

Designing an NFA

Design a 4-state NFA to recognize the following language: L $= \{$w $|$ w ends with bba$\}$

Designing an NFA

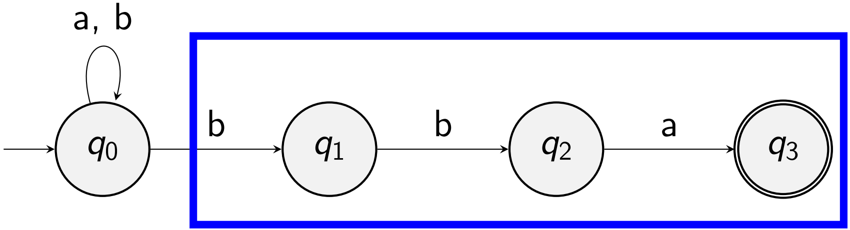
Design a 4-state NFA to recognize the following language: L = $\{w \mid w \text{ ends with bba}\}$



Designing an NFA

Design a 4-state NFA to recognize the following language: $L = \{w \mid w \text{ ends with bba}\}$

"Guess" when we've reached the end

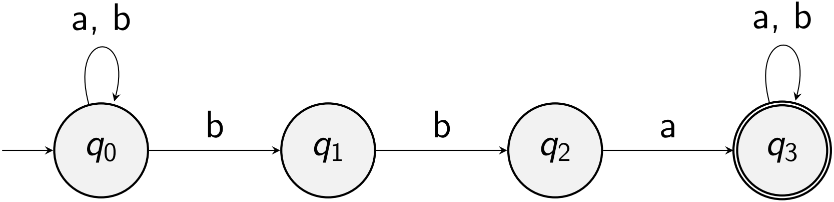


Designing an NFA

Design a 4-state NFA to recognize the following language: $L = \{w \mid w \text{ contains bba}\}$

Designing an NFA

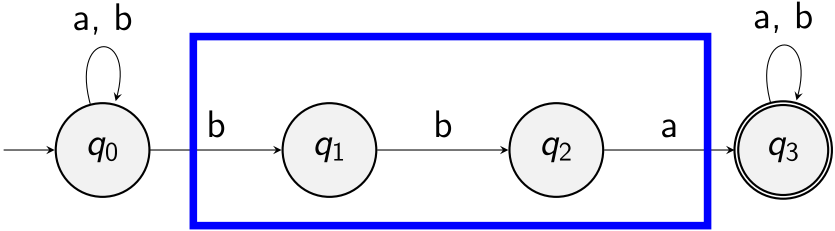
Design a 4-state NFA to recognize the following language: L = {w | w contains bba}



Designing an NFA

Design a 4-state NFA to recognize the following language: L = {w | w contains bba}

"Guess" where bba occurs



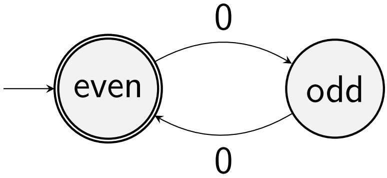
Combining NFAs

Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings with an even number of 0s

Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings with an even number of 0s

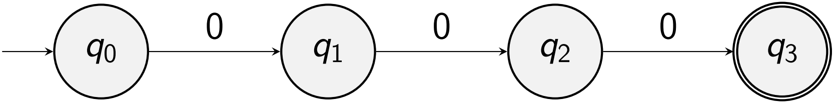


Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings with an exactly three 0s

Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings with an exactly three 0s

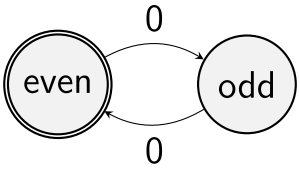


Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings where the number of 0s is even or exactly 3

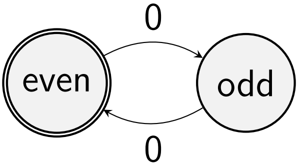
Combining NFAs

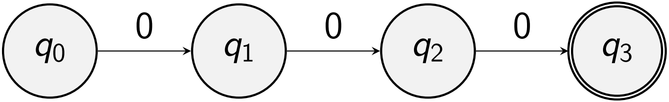
Let $\Sigma = \{0\}$. Design an NFA to recognize strings where the number of 0s is even or exactly 3



Combining NFAs

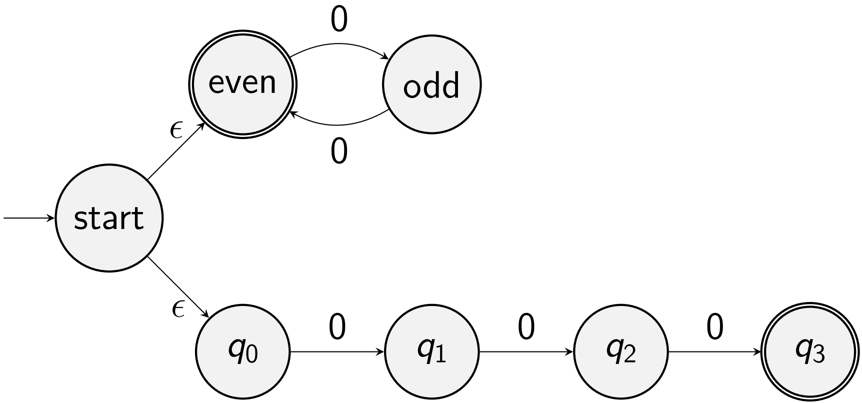
Let $\Sigma = \{0\}$. Design an NFA to recognize strings where the number of 0s is even or exactly 3





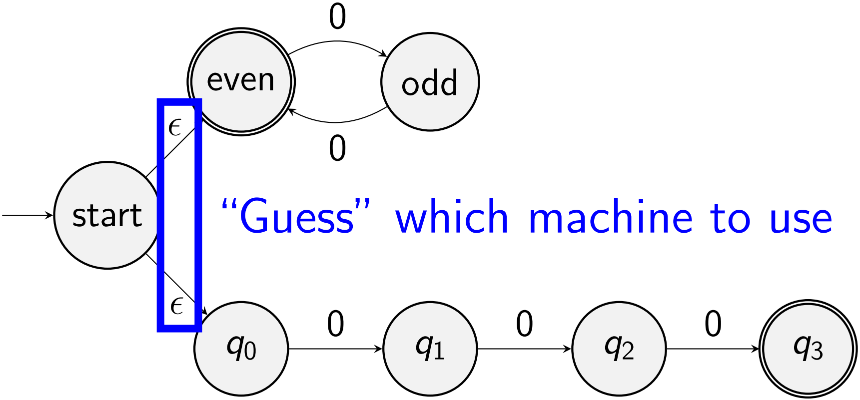
Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings where the number of 0s is even or exactly 3



Combining NFAs

Let $\Sigma = \{0\}$. Design an NFA to recognize strings where the number of 0s is even or exactly 3



"Guess" which machine to use

Equivalence of NFAs and DFAs

Equivalence of NFAs and DFAs

Theorem: A language is recognized by an NFA if and only if it is recognized by a DFA

Equivalence of NFAs and DFAs

Theorem: A language is recognized by an NFA if and only if it is recognized by a DFA

- **Proof idea:** We will show that every NFA N can be converted to an equivalent DFA D that recognizes all the same strings

Equivalence of NFAs and DFAs

Theorem: A language is recognized by an NFA if and only if it is recognized by a DFA

- **Proof idea:** We will show that every NFA N can be converted to an equivalent DFA D that recognizes all the same strings
- **Technique:** Simulate nondeterminism using the power set construction

Equivalence of NFAs and DFAs

Theorem: A language is recognized by an NFA if and only if it is recognized by a DFA

- **Proof idea:** We will show that every NFA N can be converted to an equivalent DFA D that recognizes all the same strings
- **Technique:** Simulate nondeterminism using the power set construction
  - Every state in the D will correspond to a *subset of states* in N, i.e. set of possible states where N *could be* at some point in the computation

Equivalence of NFAs and DFAs

Theorem: A language is recognized by an NFA if and only if it is recognized by a DFA

- **Proof idea:** We will show that every NFA N can be converted to an equivalent DFA D that recognizes all the same strings
- **Technique:** Simulate nondeterminism using the power set construction
  - Every state in the D will correspond to a *subset of states* in N, i.e. set of possible states where N *could be* at some point in the computation
  - Every transition in D will correspond to *all* of the possible states N could reach from *any* of the states in the previous step

Equivalence of NFAs and DFAs

Theorem: A language is recognized by an NFA if and only if it is recognized by a DFA

- **Proof idea:** We will show that every NFA N can be converted to an equivalent DFA D that recognizes all the same strings
- **Technique:** Simulate nondeterminism using the power set construction
  - Every state in the D will correspond to a *subset of states* in N, i.e. set of possible states where N *could be* at some point in the computation
  - Every transition in D will correspond to *all* of the possible states N could reach from *any* of the states in the previous step
  - Accept if the NFA *could be* in an accept state

Equivalence between NFAs and DFAs

Equivalence between NFAs and DFAs

($\Rightarrow$) If a language L is recognized by a DFA, then there exists an NFA to recognize it

Equivalence between NFAs and DFAs

($\Rightarrow$) If a language L is recognized by a DFA, then there exists an NFA to recognize it

- Suppose there is a DFA D that recognizes L

Equivalence between NFAs and DFAs

($\Rightarrow$) If a language L is recognized by a DFA, then there exists an NFA to recognize it

- Suppose there is a DFA D that recognizes L
- Then D *is* an NFA!

Equivalence between NFAs and DFAs

($\Rightarrow$) If a language L is recognized by a DFA, then there exists an NFA to recognize it

- Suppose there is a DFA D that recognizes L
- Then D *is* an NFA!
  - It's an NFA that simply chooses not to have any nondeterminism, missing transitions, or ϵ transition

Equivalence between NFAs and DFAs

($\Rightarrow$) If a language L is recognized by a DFA, then there exists an NFA to recognize it

- Suppose there is a DFA D that recognizes L
- Then D *is* an NFA!
  - It's an NFA that simply chooses not to have any nondeterminism, missing transitions, or ϵ transition
- Thus, there exists an NFA that recognizes L

Equivalence between NFAs and DFAs

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L
- For now, assume N has no ϵ transitions

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L
- For now, assume N has no ϵ transitions
- We will construct a DFA $D = (Q_D, \Sigma, q_{S_D}, \delta_D, F_D)$ to recognize L

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L
- For now, assume N has no ϵ transitions
- We will construct a DFA $D = (Q_D, \Sigma, q_{S_D}, \delta_D, F_D)$ to recognize L
  - $Q_D = \mathcal{P}(Q_N)$

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L
- For now, assume N has no ϵ transitions
- We will construct a DFA $D = (Q_D, \Sigma, q_{S_D}, \delta_D, F_D)$ to recognize L
  - $Q_D = \mathcal{P}(Q_N)$
  - $\delta_D(R, \sigma) = \bigcup_{r \in R} \delta_N(r, \sigma)$

Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L
- For now, assume N has no ϵ transitions
- We will construct a DFA $D = (Q_D, \Sigma, q_{S_D}, \delta_D, F_D)$ to recognize L
  - $Q_D = \mathcal{P}(Q_N)$
  - $\delta_D(R, \sigma) = \bigcup_{r \in R} \delta_N(r, \sigma)$
  - $q_{S_D} = \{q_{s_N}\}$

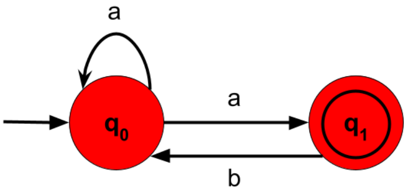
Equivalence between NFAs and DFAs

($\Leftarrow$) If a language L is recognized by an NFA, then there exists a DFA to recognize it

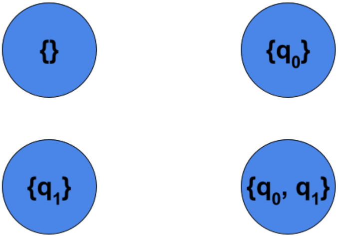
- Suppose there is an NFA $N = (Q_N, \Sigma, q_{s_N}, \delta_N, F_N)$ that recognizes L
- For now, assume N has no ϵ transitions
- We will construct a DFA $D = (Q_D, \Sigma, q_{S_D}, \delta_D, F_D)$ to recognize L
  - $Q_D = \mathcal{P}(Q_N)$
  - $\delta_D(R, \sigma) = \bigcup_{r \in R} \delta_N(r, \sigma)$
  - $q_{S_D} = \{q_{s_N}\}$
  - $F_D = \{R \subseteq Q_N | R \cap F_N \neq \emptyset\}$ (i.e., all subsets that include at least one accept state)

NFA to DFA conversion

Original NFA



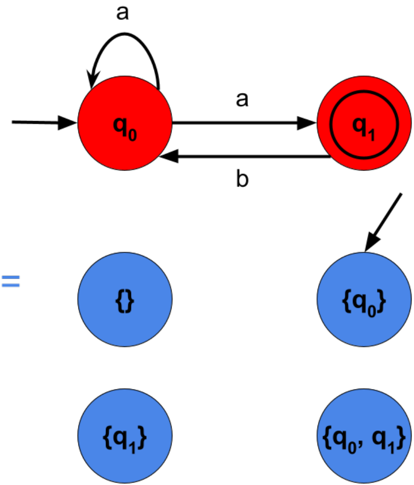
DFA States



NFA to DFA conversion

Original NFA
start state = q_0

DFA start state =
$\{q_0\}$



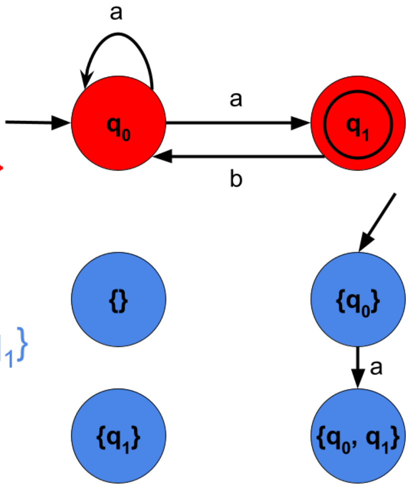
NFA to DFA conversion

Original NFA

$\delta(q_0, a) = \{q_0, q_1\}$

DFA

$\delta(\{q_0\}, a) = \{q_0, q_1\}$



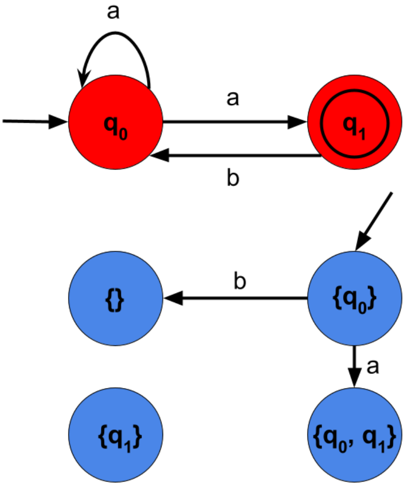
NFA to DFA conversion

Original NFA

$\delta(q_0, b) = \{\}$

DFA

$\delta(\{q_0\}, b) = \{\}$



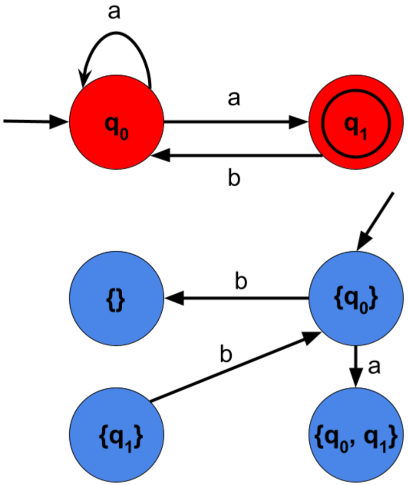
NFA to DFA conversion

Original NFA

$\delta(q_1, b) = \{q_0\}$

DFA

$\delta(\{q_1\}, b) = \{q_0\}$



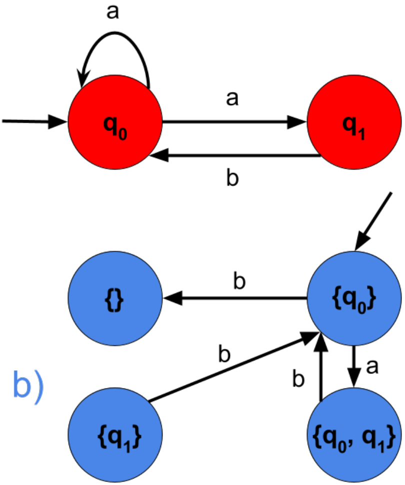
NFA to DFA conversion

Original NFA

$\delta(q_0, b) = \{\}$
$\delta(q_1, b) = \{q_0\}$

DFA

$\delta(\{q_0, q_1\}, b)$
$= \delta(q_0, b) \cup \delta(q_1, b)$
$= \{\} \cup \{q_0\}$
$= \{q_0\}$



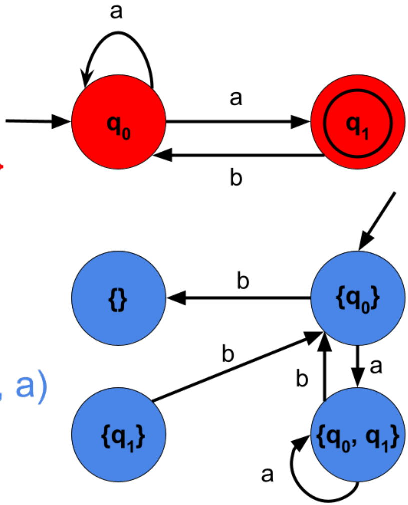
NFA to DFA conversion

Original NFA

$\delta(q_0, a) = \{q_0, q_1\}$
$\delta(q_1, a) = \{\}$

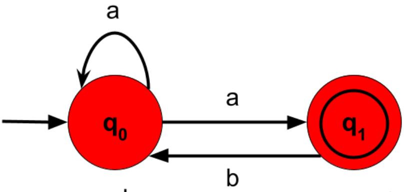
DFA

$\delta(\{q_0, q_1\}, a)$
$= \delta(q_0, a) \cup \delta(q_1, a)$
$= \{q_0, q_1\} \cup \{\}$
$= \{q_0, q_1\}$

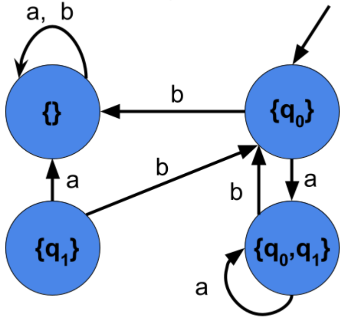


NFA to DFA conversion

Original NFA

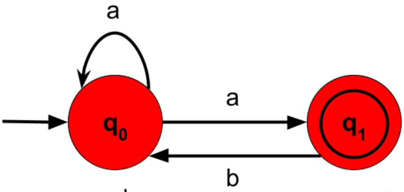


DFA with all transitions

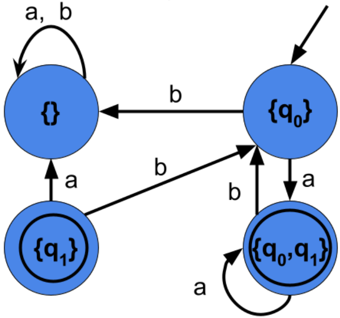


NFA to DFA conversion

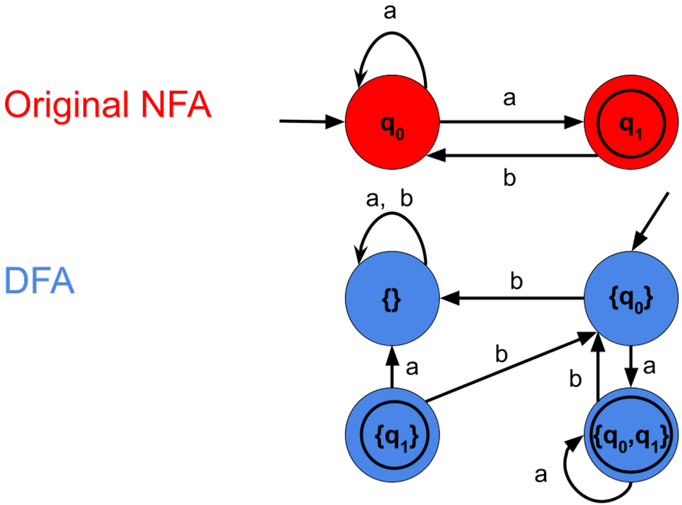
Original NFA
F = $\{q_1\}$



DFA
F = anything containing an NFA accept state
= $\{\{q_1\}, \{q_0, q_1\}\}$



NFA to DFA conversion



Epsilon Closure

Epsilon Closure

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

Epsilon Closure

- Let $N = (Q, \Sigma, q_s, \delta, F)$ be an NFA
- Let $S \subseteq Q$ be a set of states

Epsilon Closure

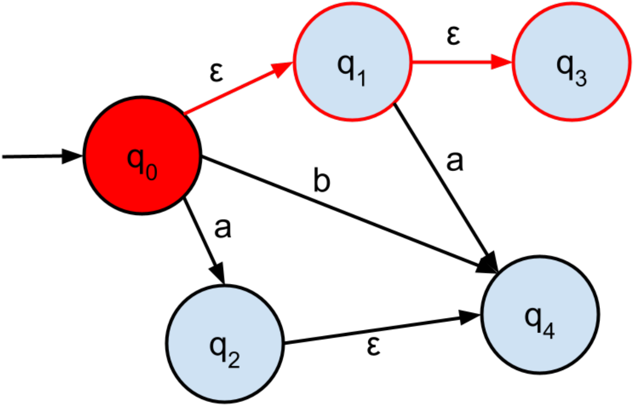
- Let $N = (Q, \Sigma, q_s, \delta, F)$ be an NFA
- Let $S \subseteq Q$ be a set of states
- **Def:** the **epsilon closure** $E(S)$ is the set of states that can be reached from S using only ϵ arrows

Epsilon Closure

- Let $N = (Q, \Sigma, q_s, \delta, F)$ be an NFA
- Let $S \subseteq Q$ be a set of states
- **Def:** the **epsilon closure** $E(S)$ is the set of states that can be reached from S using only ϵ arrows
  - This includes members of S

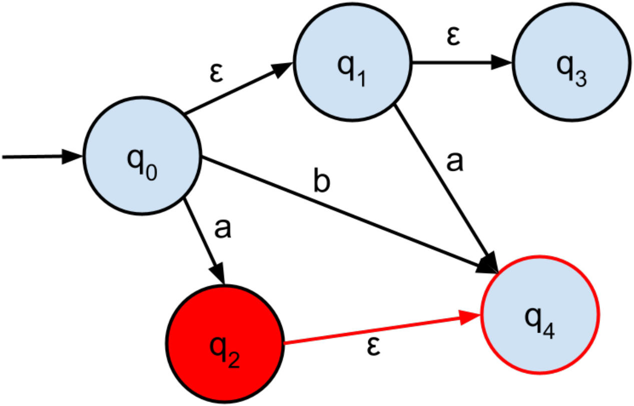
Epsilon Closure Example

$E(\{q_0\}) = \{q_0, q_1, q_3\}$



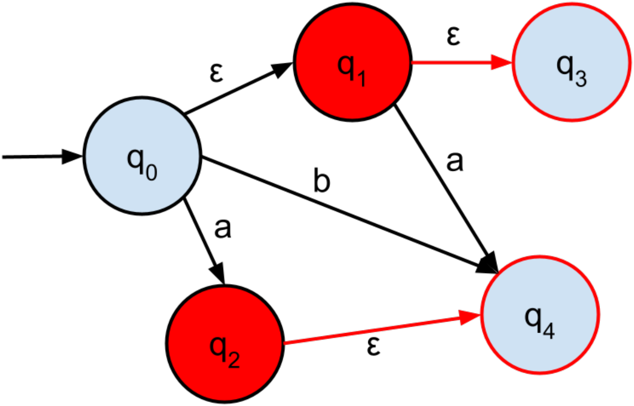
Epsilon Closure Example

$E(\{q_2\}) = \{q_2, q_4\}$



Epsilon Closure Example

$$E(\{q_1, q_2\}) = \{q_1, q_2, q_3, q_4\}$$



NFA to DFA conversion

How do we extend our conversion to account for ϵ transitions?

- $Q = \mathcal{P}(Q_N)$
- $\delta_D(R, \sigma) = \bigcup_{r \in R} \delta_N(r, \sigma)$
- $q_{S_D} = \{q_{s_N}\}$
- $F_D = \{R \subseteq Q_N | R \cap F_N \neq \emptyset\}$

NFA to DFA conversion

How do we extend our conversion to account for ϵ transitions?

- $Q = \mathcal{P}(Q_N)$
- $\delta_D(R, \sigma) = \bigcup_{r \in R} \delta_N(r, \sigma)$
- $q_{S_D} = \{q_{s_N}\}$
- $F_D = \{R \subseteq Q_N | R \cap F_N \neq \emptyset\}$

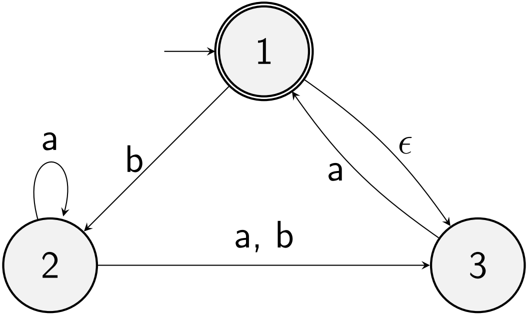
NFA to DFA conversion

How do we extend our conversion to account for ϵ transitions?

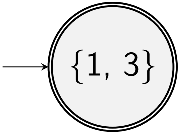
- $Q = \mathcal{P}(Q_N)$
- $\delta_D(R, \sigma) = E\left(\bigcup_{r \in R} \delta_N(r, \sigma)\right)$
- $q_{S_D} = E(\{q_{s_N}\})$
- $F_D = \{R \subseteq Q_N | R \cap F_N \neq \emptyset\}$

NFA to DFA Conversion Example

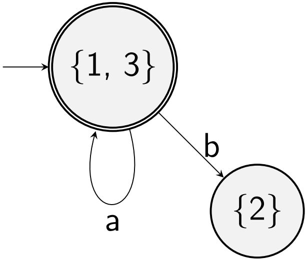
Let's convert the following NFA to a DFA



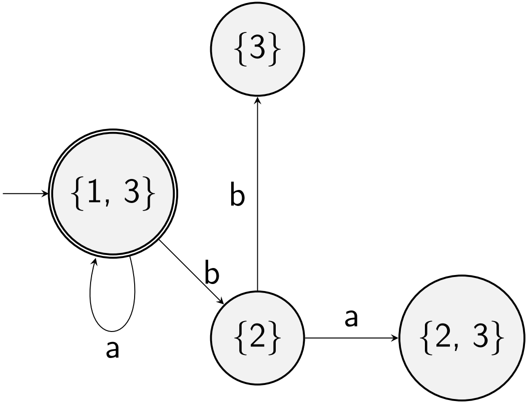
NFA to DFA Conversion Example



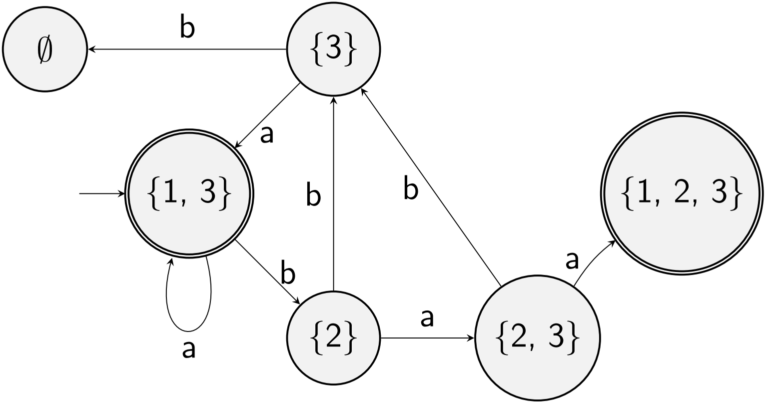
NFA to DFA Conversion Example



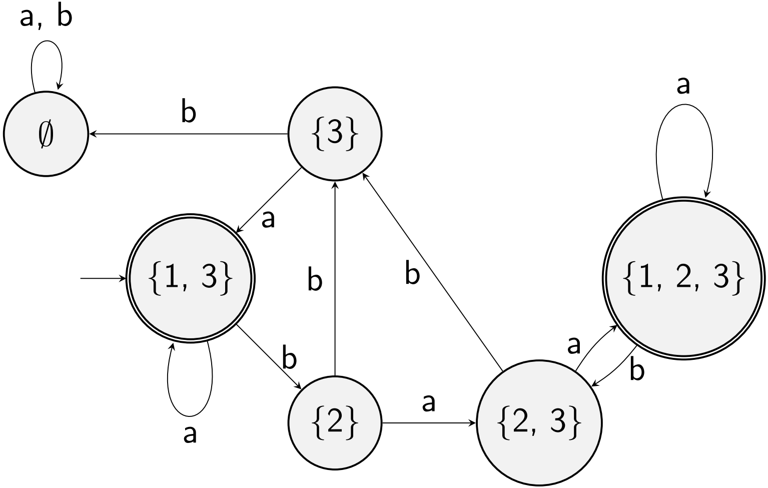
NFA to DFA Conversion Example



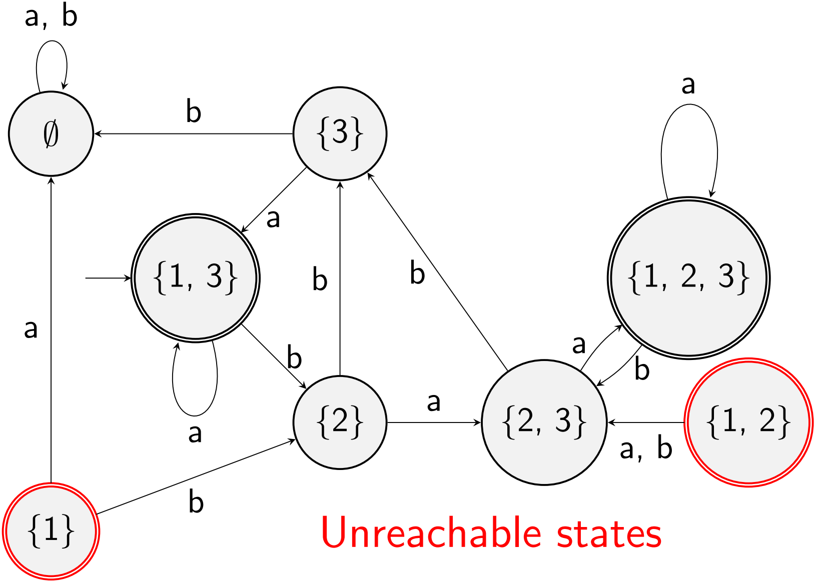
NFA to DFA Conversion Example



NFA to DFA Conversion Example



NFA to DFA Conversion Example



Unreachable states

NFAs and regular languages

NFAs and regular languages

- Recall that the regular languages are the languages recognized by DFAs

NFAs and regular languages

- Recall that the regular languages are the languages recognized by DFAs
- We have proven that DFAs and NFAs are equivalent

NFAs and regular languages

NFAs and regular languages

- Recall that the regular languages are the languages recognized by DFAs
- We have proven that DFAs and NFAs are equivalent
- **Corollary:** a language is regular if and only if it is recognized by an NFA

NFAs and regular languages

- Recall that the regular languages are the languages recognized by DFAs
- We have proven that DFAs and NFAs are equivalent
- **Corollary:** a language is regular if and only if it is recognized by an NFA
- It will often be more convenient use NFAs when we want to show that a langauge is regular!

Regular operations

Regular operations

Recall the regular operations:

Regular operations

Recall the regular operations:

- **Union:**
  $A \cup B = \{w | w \in A \text{ or } w \in B\}$

Regular operations

Recall the regular operations:

- **Union:**
  $A \cup B = \{w | w \in A \text{ or } w \in B\}$
- **Concatenation:**
  $A \circ B = \{w = w_1 w_2 | w_1 \in A, w_2 \in B\}$

Regular operations

Recall the regular operations:

- **Union:**
  $A \cup B = \{w | w \in A \text{ or } w \in B\}$
- **Concatenation:**
  $A \circ B = \{w = w_1 w_2 | w_1 \in A, w_2 \in B\}$
- **(Kleene) Star:**
  $A^* = \{\epsilon\} \cup \{w = w_1 w_2 \ldots w_n | w_i \in A\}$

Kleene's Theorem

Kleene's Theorem

Theorem: The regular languages are closed under the regular operations

Kleene's Theorem

Theorem: The regular languages are closed under the regular operations

- Want to show that if L_1 and L_2 are regular, then $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^* are regular

Kleene's Theorem

Theorem: The regular languages are closed under the regular operations

- Want to show that if L_1 and L_2 are regular, then $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^* are regular
- With DFAs, it was messy

Kleene's Theorem

Theorem: The regular languages are closed under the regular operations

- Want to show that if L_1 and L_2 are regular, then $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^* are regular
- With DFAs, it was messy
- With NFAs, this will be easy!

Kleene's Theorem

Theorem: The regular languages are closed under the regular operations

- Want to show that if L_1 and L_2 are regular, then $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^* are regular
- With DFAs, it was messy
- With NFAs, this will be easy!
- **Proof idea:** We will combine the DFAs for L_1 and L_2 into an NFA that simulates the regular operation.

Kleene's Theorem

Theorem: The regular languages are closed under the regular operations

- Want to show that if L_1 and L_2 are regular, then $L_1 \cup L_2$, $L_1 \circ L_2$, and L_1^* are regular
- With DFAs, it was messy
- With NFAs, this will be easy!
- **Proof idea:** We will combine the DFAs for L_1 and L_2 into an NFA that simulates the regular operation.
  - For Kleene star we only modify the DFA for L_1

Closure under union

Closure under union

- Let N_1 recognize L_1 and let N_2 recognize L_2

Closure under union

- Let N_1 recognize L_1 and let N_2 recognize L_2
- Start with the two smaller NFAs

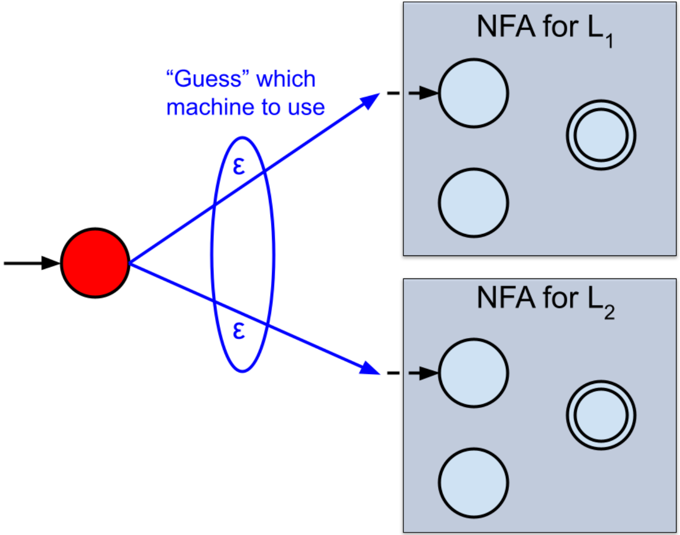
Closure under union

- Let N_1 recognize L_1 and let N_2 recognize L_2
- Start with the two smaller NFAs
- Add a new start state

Closure under union

- Let N_1 recognize L_1 and let N_2 recognize L_2
- Start with the two smaller NFAs
- Add a new start state
- Add ϵ transitions to the two original start states

Closure under union



Closure under concatenation

Closure under concatenation

- Let N_1 recognize L_1 and let N_2 recognize L_2

Closure under concatenation

- Let N_1 recognize L_1 and let N_2 recognize L_2
- Start with the two smaller NFAs

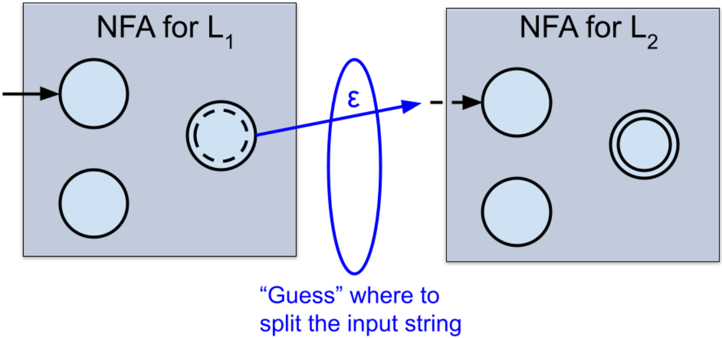
Closure under concatenation

- Let N_1 recognize L_1 and let N_2 recognize L_2
- Start with the two smaller NFAs
- Add an ϵ transition between N_1's accept state(s) and N_2's start state

Closure under concatenation

- Let N_1 recognize L_1 and let N_2 recognize L_2
- Start with the two smaller NFAs
- Add an ϵ transition between N_1's accept state(s) and N_2's start state
- Accept states in N_1 are no longer accept states (we have to accept in N_2)

Closure under concatenation



Closure under Kleene star

Closure under Kleene star

- Let N_1 recognize L_1

Closure under Kleene star

- Let N_1 recognize L_1
- Start with the smaller NFA

Closure under Kleene star

- Let N_1 recognize L_1
- Start with the smaller NFA
- Add ϵ transitions from each accept state back to the start state

Closure under Kleene star

- Let N_1 recognize L_1
- Start with the smaller NFA
- Add ϵ transitions from each accept state back to the start state
- Add an new start state with an ϵ transition to the original start state

Closure under Kleene star

- Let N_1 recognize L_1
- Start with the smaller NFA
- Add ϵ transitions from each accept state back to the start state
- Add an new start state with an ϵ transition to the original start state
  - This new start state will also be an accept state

Closure under Kleene star