

# Remarkable Fact (that we want to prove)

Thm.: A language,  $L$ , is described by a **regular expression,  $R$** , if and only if  $L$  is regular.

# Remarkable Fact (that we want to prove)

Thm.: A language,  $L$ , is described by a **regular expression,  $R$** , if and only if  $L$  is regular.

$\implies$  construct an NFA accepting  $R$ .

## Remarkable Fact (that we want to prove)

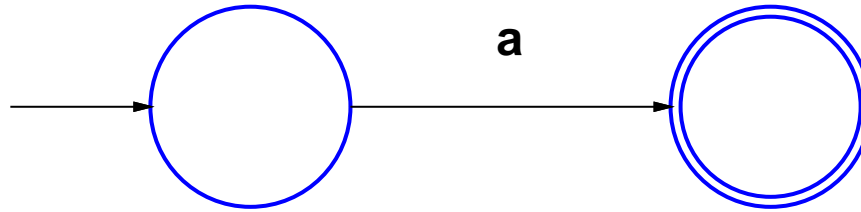
Thm.: A language,  $L$ , is described by a **regular expression,  $R$** , if and only if  $L$  is regular.

$\implies$  construct an NFA accepting  $R$ .

$\impliedby$  Given a **regular language,  $L$** , construct an equivalent **regular expression**.

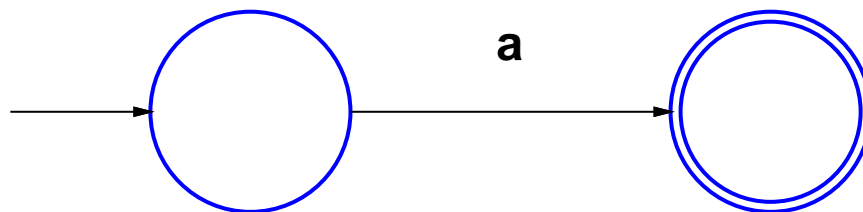
# $(\implies)$ NFA Accepting Reg Expression, $R$

1.  $R = a$ , for some  $a \in \Sigma$

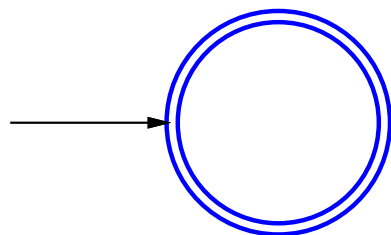


# $(\implies)$ NFA Accepting Reg Expression, $R$

1.  $R = a$ , for some  $a \in \Sigma$

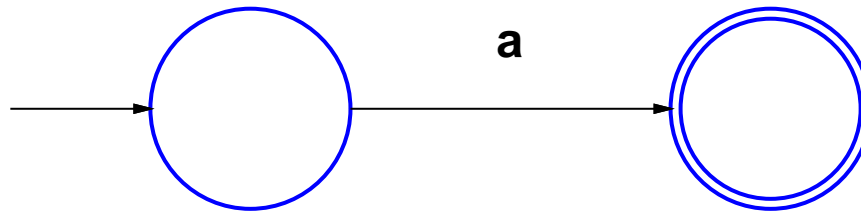


2.  $R = \varepsilon$

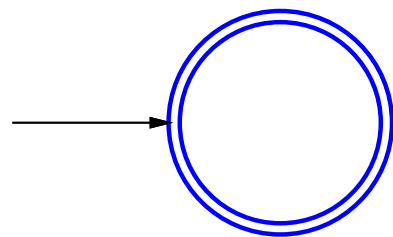


# $(\implies)$ NFA Accepting Reg Expression, $R$

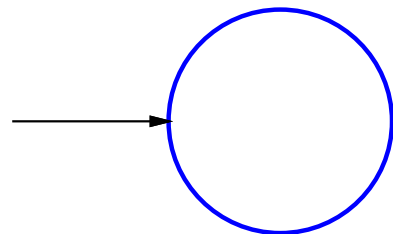
1.  $R = a$ , for some  $a \in \Sigma$



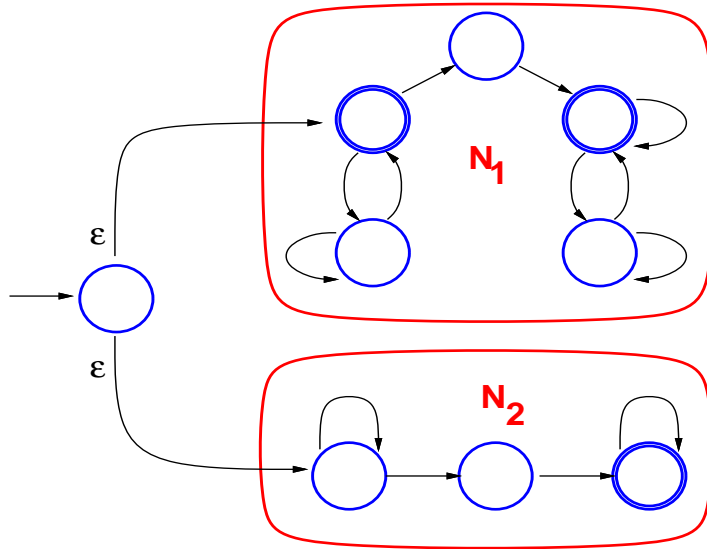
2.  $R = \varepsilon$



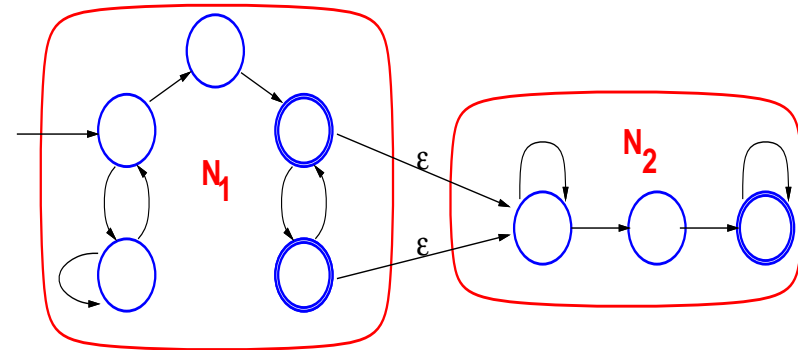
3.  $R = \emptyset$



# ( $\implies$ ) NFA Accepting Reg Expression, $R$

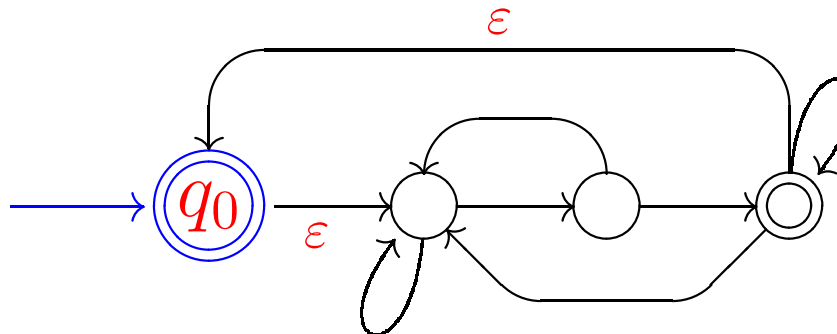


$$R = (R_1 \cup R_2)$$

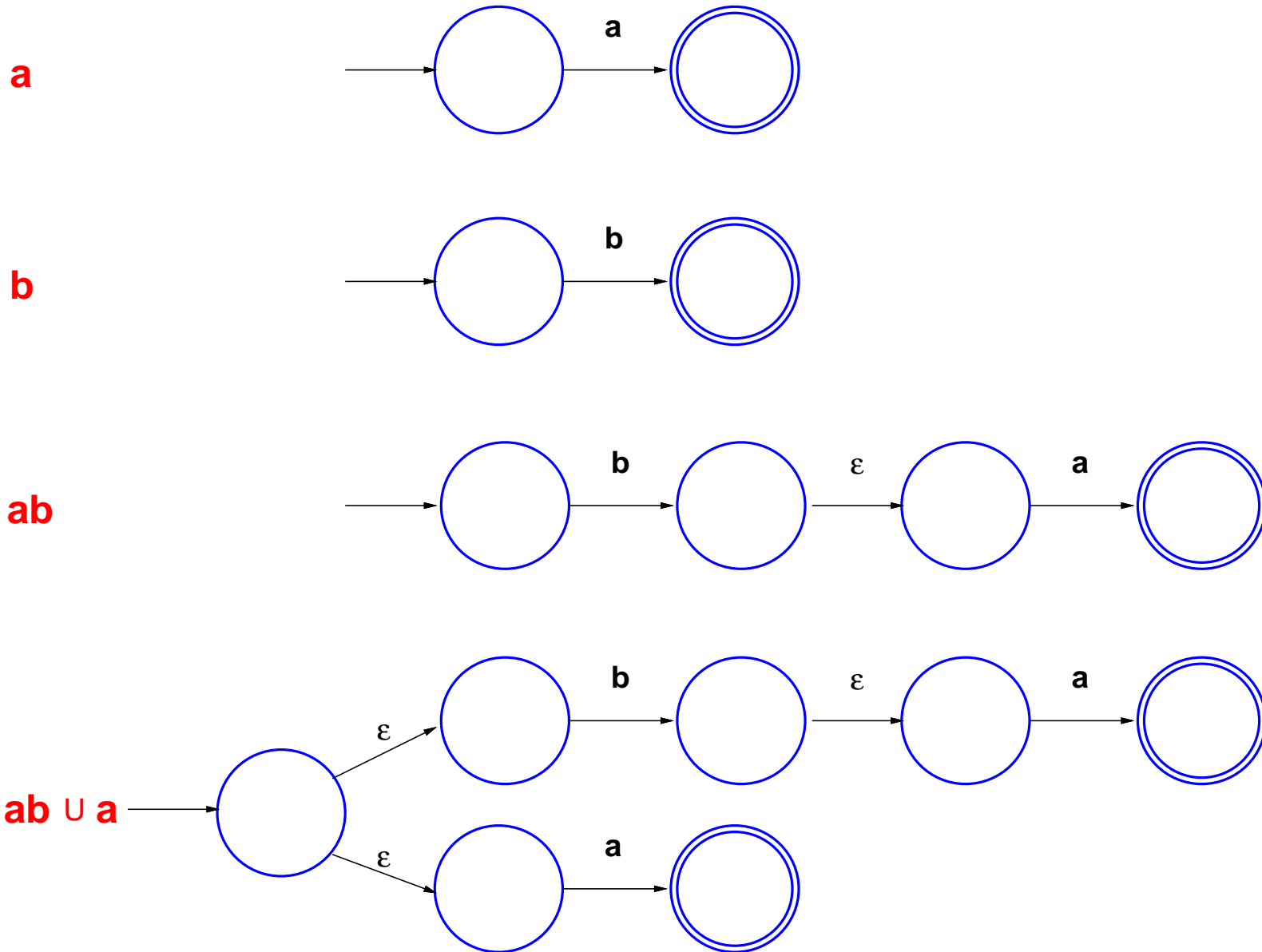


$$R = (R_1 \circ R_2)$$

$$R = (R_1)^*$$



# Example



## ( $\Leftarrow$ ) Regular Expression from an NFA

We now define **generalized** non-deterministic finite automata (**GNFA**).

## ( $\Leftarrow$ ) Regular Expression from an NFA

We now define **generalized** non-deterministic finite automata (**GNFA**).

An **NFA**:

- Each transition labeled with a symbol or  $\varepsilon$ ,
- reads **zero or one** symbols,
- takes matching transition, if any.

## ( $\Leftarrow$ ) Regular Expression from an NFA

We now define **generalized** non-deterministic finite automata (**GNFA**).

An **NFA**:

- Each transition labeled with a symbol or  $\varepsilon$ ,
- reads **zero or one** symbols,
- takes matching transition, if any.

A **GNFA**:

- Each transition labeled with a **regular expression**,
- reads **zero or more** symbols,
- takes transition whose **regular expression** matches string, if any.

## ( $\Leftarrow$ ) Regular Expression from an NFA

We now define **generalized** non-deterministic finite automata (**GNFA**).

An **NFA**:

- Each transition labeled with a symbol or  $\varepsilon$ ,
- reads **zero or one** symbols,
- takes matching transition, if any.

A **GNFA**:

- Each transition labeled with a **regular expression**,
- reads **zero or more** symbols,
- takes transition whose **regular expression** matches string, if any.

**GNFAs** are natural generalization of NFAs.

# GNFA Special Form

- Start state has outgoing arrows to **every** other state, but no incoming arrows.

# GNFA Special Form

- Start state has outgoing arrows to **every** other state, but no incoming arrows.
- Unique **accept state** has incoming arrows from **every** other state, but no outgoing arrows

# GNFA Special Form

- Start state has outgoing arrows to **every** other state, but no incoming arrows.
- Unique **accept state** has incoming arrows from **every** other state, but no outgoing arrows
- Except for start and accept states, arrows goes **from every state to every other state**, including itself.

# GNFA Special Form

- Start state has outgoing arrows to **every** other state, but no incoming arrows.
- Unique **accept state** has incoming arrows from **every** other state, but no outgoing arrows
- Except for start and accept states, arrows goes **from every state to every other state**, including itself.

Easy to transform any GNFA into special form.

# GNFA Special Form

- Start state has outgoing arrows to **every** other state, but no incoming arrows.
- Unique **accept state** has incoming arrows from **every** other state, but no outgoing arrows
- Except for start and accept states, arrows goes **from every state to every other state**, including itself.

Easy to transform any GNFA into special form.

Really? How? ...

# Converting DFA to Regular Expression

( $\Leftarrow$ )

Strategy – sequence of **equivalent** transformations

- given a  $k$ -state DFA

# Converting DFA to Regular Expression

( $\Leftarrow$ )

Strategy – sequence of **equivalent** transformations

- given a  $k$ -state DFA
- transform into  $(k + 2)$ -state GNFA

# Converting DFA to Regular Expression

( $\Leftarrow$ )

Strategy – sequence of **equivalent** transformations

- given a  $k$ -state DFA
- transform into  $(k + 2)$ -state GNFA
- while GNFA has **more than 2 states**, transform it into equivalent GNFA with **one fewer** state

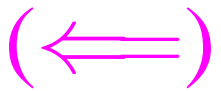
# Converting DFA to Regular Expression

( $\Leftarrow$ )

Strategy – sequence of **equivalent** transformations

- given a  $k$ -state DFA
- transform into  $(k + 2)$ -state GNFA
- while GNFA has **more than 2 states**, transform it into equivalent GNFA with **one fewer** state
- eventually reach **2**-state GNFA (states are just **start** and **accept**).

# Converting DFA to Regular Expression

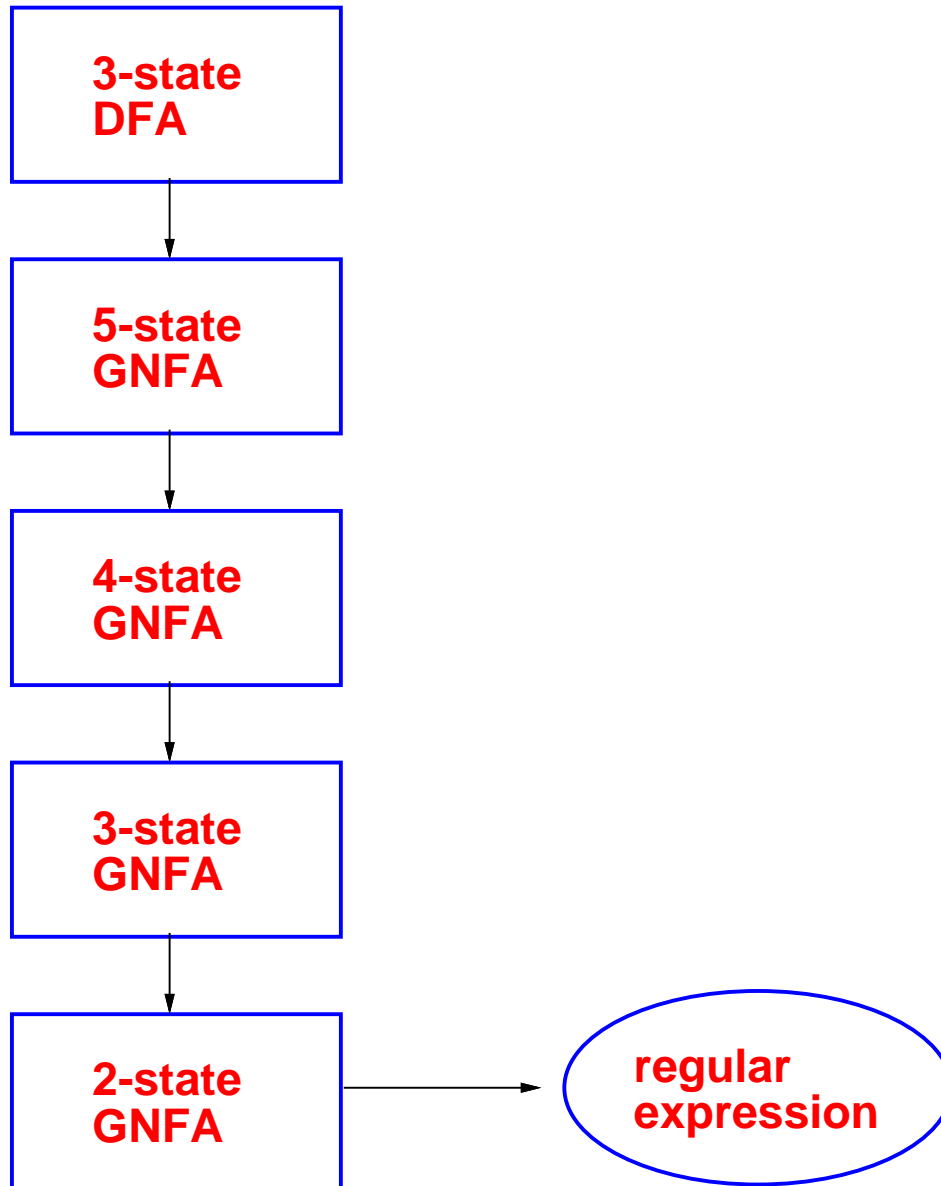


Strategy – sequence of **equivalent** transformations

- given a  $k$ -state DFA
- transform into  $(k + 2)$ -state GNFA
- while GNFA has **more than 2 states**, transform it into equivalent GNFA with **one fewer** state
- eventually reach **2**-state GNFA (states are just **start** and **accept**).
- label on single transition is the **desired regular expression**.

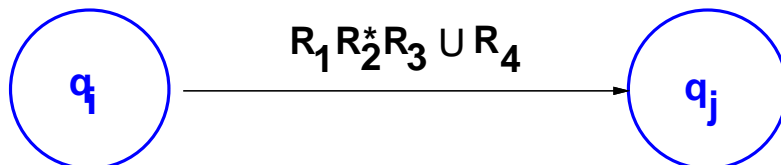
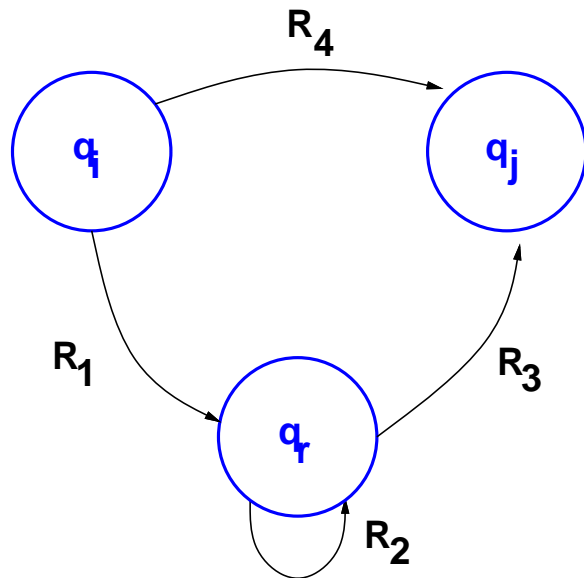


# Converting Strategy ( $\Leftarrow$ )



# Removing One State

We **remove** one state  $q_r$ , and then **repair** the machine by **altering** regular expression of other transitions.



# Formal Treatment – GNDA Definition

- $q_s$  is start state.
- $q_a$  is accept state.
- $\mathcal{R}$  is collection of regular expressions over  $\Sigma$ .

# Formal Treatment – GNDA Definition

- $q_s$  is start state.
- $q_a$  is accept state.
- $\mathcal{R}$  is collection of regular expressions over  $\Sigma$ .

The transition function is

$$\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$$

# Formal Treatment – GNDA Definition

- $q_s$  is start state.
- $q_a$  is accept state.
- $\mathcal{R}$  is collection of regular expressions over  $\Sigma$ .

The transition function is

$$\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$$

Arrows connect every state to every other state except:

- no arrow from  $q_a$
- no arrow to  $q_s$

# Formal Treatment – GNDA Definition

- $q_s$  is start state.
- $q_a$  is accept state.
- $\mathcal{R}$  is collection of regular expressions over  $\Sigma$ .

The transition function is

$$\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$$

Arrows connect every state to every other state except:

- no arrow from  $q_a$
- no arrow to  $q_s$

If  $\delta(q_i, q_j) = R$ , then **arrow** from  $q_i$  to  $q_j$  has **label**  $R$ .

# Formal Definition

A **generalized** deterministic finite automaton (**GDFA**) is  $(Q, \Sigma, \delta, q_s, q_a)$ , where

- $Q$  is a finite set of **states**,
- $\Sigma$  is the **alphabet**,
- $\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$  is the **transition function**.
- $q_s \in Q$  is the **start state**, and
- $q_a \in Q$  is the unique **accept state**.

# A Formal Model of GNFA Computation

A GNFA accepts a string  $w \in \Sigma^*$  if **there exists** a *parsing* of  $w$ ,  $w = w_1 w_2 \cdots w_k$ , where each  $w_i \in \Sigma^*$ , and **there exists** a sequence of states  $q_0, \dots, q_k$  such that

- $q_0 = q_s$ , the start state,
- $q_k = q_a$ , the accept state, and
- for each  $i$ ,  $w_i \in L(R_i)$ , where  $R_i = \delta(q_{i-1}, q_i)$ .
- (namely  $w_i$  is an element of the language described by the regular expression  $R_i$ .)

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .
- Let  $Q' = Q - \{q_r\}$ .

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .
- Let  $Q' = Q - \{q_r\}$ .
- For any  $q_i \in Q' - \{q_a\}$  and  $q_j \in Q' - \{q_s\}$ , let

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .
- Let  $Q' = Q - \{q_r\}$ .
- For any  $q_i \in Q' - \{q_a\}$  and  $q_j \in Q' - \{q_s\}$ , let
  - $R_1 = \delta(q_i, q_r)$ ,  $R_2 = \delta(q_r, q_j)$ ,

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .
- Let  $Q' = Q - \{q_r\}$ .
- For any  $q_i \in Q' - \{q_a\}$  and  $q_j \in Q' - \{q_s\}$ , let
  - $R_1 = \delta(q_i, q_r)$ ,  $R_2 = \delta(q_r, q_r)$ ,
  - $R_3 = \delta(q_r, q_j)$ , and  $R_4 = \delta(q_i, q_j)$ .

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .
- Let  $Q' = Q - \{q_r\}$ .
- For any  $q_i \in Q' - \{q_a\}$  and  $q_j \in Q' - \{q_s\}$ , let
  - $R_1 = \delta(q_i, q_r)$ ,  $R_2 = \delta(q_r, q_r)$ ,
  - $R_3 = \delta(q_r, q_j)$ , and  $R_4 = \delta(q_i, q_j)$ .
- Define  $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$ .

# The CONVERT Algorithm

Given GDFA  $G$ , convert it to equivalent GNFA  $G'$ .

- let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow.
- If  $k > 2$ , select any  $q_r$  distinct from  $q_s$  and  $q_a$ .
- Let  $Q' = Q - \{q_r\}$ .
- For any  $q_i \in Q' - \{q_a\}$  and  $q_j \in Q' - \{q_s\}$ , let
  - $R_1 = \delta(q_i, q_r)$ ,  $R_2 = \delta(q_r, q_r)$ ,
  - $R_3 = \delta(q_r, q_j)$ , and  $R_4 = \delta(q_i, q_j)$ .
- Define  $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$ .
- Denote the resulting  $k - 1$  states GNFA by  $G'$ .

# The CONVERT Procedure

We define the recursive procedure **CONVERT**( $\cdot$ ):

Given GDFA  $G$ .

- Let  $k$  be the number of states of  $G$ .

# The CONVERT Procedure

We define the recursive procedure **CONVERT**( $\cdot$ ):

Given GDFA  $G$ .

- Let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow of  $G$ .

# The CONVERT Procedure

We define the recursive procedure **CONVERT**( $\cdot$ ):

Given GDFA  $G$ .

- Let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow of  $G$ .
- If  $k > 2$ , let  $G'$  be the  $k - 1$  states GNFA produced by the algorithm.

# The CONVERT Procedure

We define the recursive procedure **CONVERT**( $\cdot$ ):

Given GDFA  $G$ .

- Let  $k$  be the number of states of  $G$ .
- If  $k = 2$ , return the regular expression labeling the only arrow of  $G$ .
- If  $k > 2$ , let  $G'$  be the  $k - 1$  states GNFA produced by the algorithm.
- Return **CONVERT**( $G'$ ).

# Correctness Proof of Construction

**Theorem:**  $G$  and  $\text{CONVERT}(G)$  accept the same language.

**Proof:** By induction on number of states of  $G$

# Correctness Proof of Construction

**Theorem:**  $G$  and  $\text{CONVERT}(G)$  accept the same language.

**Proof:** By induction on number of states of  $G$

**Basis:** When there are only 2 states, there is a single label, which characterizes the strings accepted by  $G$ .

# Correctness Proof of Construction

**Theorem:**  $G$  and  $\text{CONVERT}(G)$  accept the same language.

**Proof:** By induction on number of states of  $G$

**Basis:** When there are only 2 states, there is a single label, which characterizes the strings accepted by  $G$ .

**Induction Step:** Assume claim for  $k - 1$  states, prove for  $k$ .

# Correctness Proof of Construction

**Theorem:**  $G$  and  $\text{CONVERT}(G)$  accept the same language.

**Proof:** By induction on number of states of  $G$

**Basis:** When there are only 2 states, there is a single label, which characterizes the strings accepted by  $G$ .

**Induction Step:** Assume claim for  $k - 1$  states, prove for  $k$ .

Let  $G'$  be the  $k - 1$  states GNFA produced from  $G$  by the algorithm.

$G$  and  $G'$  accept the same language

By the induction hypothesis,  $G'$  and  $\text{CONVERT}(G')$  accept the same language.

$G$  and  $G'$  accept the same language

By the induction hypothesis,  $G'$  and  $\text{CONVERT}(G')$  accept the same language.

On input  $G$ , the procedure returns  $\text{CONVERT}(G')$ .

$G$  and  $G'$  accept the same language

By the induction hypothesis,  $G'$  and  $\text{CONVERT}(G')$  accept the same language.

On input  $G$ , the procedure returns  $\text{CONVERT}(G')$ .

So to complete the proof, it suffices to show that  $G$  and  $G'$  accept the same language.

# $G$ and $G'$ accept the same language

By the induction hypothesis,  $G'$  and  $\text{CONVERT}(G')$  accept the same language.

On input  $G$ , the procedure returns  $\text{CONVERT}(G')$ .

So to complete the proof, it suffices to show that  $G$  and  $G'$  accept the same language.

Three steps:

1. If  $G$  accepts  $w$ , then so does  $G'$ .
2. If  $G'$  accepts  $w$ , then so does  $G$ .
3. Therefore  $G$  and  $G'$  are equivalent.

# Step One

**Claim:** If  $G$  accepts  $w$ , then so does  $G'$ :

- If  $G$  accepts  $w$ , then there exists a “path of states”  $q_s, q_1, q_2, \dots, q_a$  traversed by  $G$  on  $w$ , leading to the accept state  $q_a$ .

# Step One

**Claim:** If  $G$  accepts  $w$ , then so does  $G'$ :

- If  $G$  accepts  $w$ , then there exists a “path of states”  $q_s, q_1, q_2, \dots, q_a$  traversed by  $G$  on  $w$ , leading to the accept state  $q_a$ .
- If  $q_r$  does not appear on path, then  $G'$  accepts  $w$  because the the new regular expression on each edge of  $G'$  contains the old regular expression in the “union part”.

# Step One

**Claim:** If  $G$  accepts  $w$ , then so does  $G'$ :

- If  $G$  accepts  $w$ , then there exists a “path of states”  $q_s, q_1, q_2, \dots, q_a$  traversed by  $G$  on  $w$ , leading to the accept state  $q_a$ .
- If  $q_r$  does not appear on path, then  $G'$  accepts  $w$  because the the new regular expression on each edge of  $G'$  contains the old regular expression in the “union part”.
- If  $q_r$  does appear, consider the regular expression corresponding to  $\dots q_i, q_r, \dots, q_r, q_j \dots$ .  
The new regular expression  $(R_{i,r})(R_{r,r})^*(R_{r,j})$  linking  $q_i$  and  $q_j$  encompasses any such string.

# Step One

**Claim:** If  $G$  accepts  $w$ , then so does  $G'$ :

- If  $G$  accepts  $w$ , then there exists a “path of states”  $q_s, q_1, q_2, \dots, q_a$  traversed by  $G$  on  $w$ , leading to the accept state  $q_a$ .
- If  $q_r$  does not appear on path, then  $G'$  accepts  $w$  because the the new regular expression on each edge of  $G'$  contains the old regular expression in the “union part”.
- If  $q_r$  does appear, consider the regular expression corresponding to  $\dots q_i, q_r, \dots, q_r, q_j \dots$ .  
The new regular expression  $(R_{i,r})(R_{r,r})^*(R_{r,j})$  linking  $q_i$  and  $q_j$  encompasses any such string.
- Either way, the claim holds.

## Steps Two and Three

**Claim:** If  $G'$  accepts  $w$ , then so does  $G$ .

**Proof:** Each transition from  $q_i$  to  $q_j$  in  $G'$  corresponds to a transition in  $G$ , either directly or through  $q_r$ . Thus if  $G'$  accepts  $w$ , then so does  $G$ .

- This completes the proof of the claim that  $L(G) = L(G')$ .

## Steps Two and Three

**Claim:** If  $G'$  accepts  $w$ , then so does  $G$ .

**Proof:** Each transition from  $q_i$  to  $q_j$  in  $G'$  corresponds to a transition in  $G$ , either directly or through  $q_r$ . Thus if  $G'$  accepts  $w$ , then so does  $G$ .

- This completes the proof of the claim that  $L(G) = L(G')$ .
- Combined with the induction hypothesis, this shows that  $G$  and the regular expression  $\text{CONVERT}(G)$  accept the same language.

## Steps Two and Three

**Claim:** If  $G'$  accepts  $w$ , then so does  $G$ .

**Proof:** Each transition from  $q_i$  to  $q_j$  in  $G'$  corresponds to a transition in  $G$ , either directly or through  $q_r$ . Thus if  $G'$  accepts  $w$ , then so does  $G$ .

- This completes the proof of the claim that  $L(G) = L(G')$ .
- Combined with the induction hypothesis, this shows that  $G$  and the regular expression  $\text{CONVERT}(G)$  accept the same language.
- This, in turn, proves our *remarkable claim*: A language,  $L$ , is described by a **regular expression**,  $R$ , if and only if  $L$  is regular.



# Negative Results

We have made a lot of progress understanding what finite automata **can** do. But what **can't** they do?

# Negative Results

We have made a lot of progress understanding what finite automata **can** do. But what **can't** they do?

Is there a DFA that accepts

- $B = \{0^n 1^n \mid n \geq 0\}$
- $C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$
- $D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 substrings}\}$

Consider  $B$ :

- DFA must “remember” how many **0**'s it has seen
- impossible with finite state.

The others are exactly the same.

# Negative Results

Is there a DFA that accepts

- $B = \{0^n 1^n \mid n \geq 0\}$
- $C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$
- $D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 substrings}\}$

# Negative Results

Is there a DFA that accepts

- $B = \{0^n 1^n \mid n \geq 0\}$
- $C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$
- $D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 substrings}\}$

Consider  $B$ :

- DFA must “remember” how many 0's it has seen
- impossible with finite state.

# Negative Results

Is there a DFA that accepts

- $B = \{0^n 1^n \mid n \geq 0\}$
- $C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$
- $D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 substrings}\}$

Consider  $B$ :

- DFA must “remember” how many 0's it has seen
- impossible with finite state.

The others are exactly the same...

# Negative Results

Is there a DFA that accepts

- $B = \{0^n 1^n \mid n \geq 0\}$
- $C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$
- $D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 substrings}\}$

Consider  $B$ :

- DFA must “remember” how many 0's it has seen
- impossible with finite state.

The others are exactly the same...

**Question:** Is this a proof?

# Negative Results

Is there a DFA that accepts

- $B = \{0^n 1^n \mid n \geq 0\}$
- $C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$
- $D = \{w \mid w \text{ has an equal number of occurrences of 01 and 10 substrings}\}$

Consider  $B$ :

- DFA must “remember” how many 0's it has seen
- impossible with finite state.

The others are exactly the same...

**Question:** Is this a proof?

**Answer:** No,  $D$  is regular!??? (see problem set 1)

# Pumping Lemma

We will show that all regular languages have a special property.

- Suppose  $L$  is regular.
- If a string in  $L$  is longer than a certain critical length  $\ell$  (the **pumping length**),
- then it can be “**pumped**” to a longer string by **repeating** an internal substring any number of times.
- The longer string must be in  $L$  too.
- This is a powerful technique for showing that a language is **not regular**.

# Pumping Lemma

**Theorem:** If  $L$  is a regular language, then there is an  $\ell > 0$  (the pumping length), where if  $s$  is any string in  $L$  of length  $|s| > \ell$ , then  $s$  may be divided into three pieces  $s = xyz$  such that

# Pumping Lemma

**Theorem:** If  $L$  is a regular language, then there is an  $\ell > 0$  (the **pumping length**), where if  $s$  is any string in  $L$  of length  $|s| > \ell$ , then  $s$  may be divided into three pieces  $s = xyz$  such that

- for every  $i > 0$ ,  $xy^iz \in L$ ,
- $|y| > 0$ , and
- $|xy| \leq \ell$ .

# Pumping Lemma

**Theorem:** If  $L$  is a regular language, then there is an  $\ell > 0$  (the **pumping length**), where if  $s$  is any string in  $L$  of length  $|s| > \ell$ , then  $s$  may be divided into three pieces  $s = xyz$  such that

- for every  $i > 0$ ,  $xy^iz \in L$ ,
- $|y| > 0$ , and
- $|xy| \leq \ell$ .

**Remarks:** Without the second condition, the theorem would be trivial.

# Pumping Lemma

**Theorem:** If  $L$  is a regular language, then there is an  $\ell > 0$  (the **pumping length**), where if  $s$  is any string in  $L$  of length  $|s| > \ell$ , then  $s$  may be divided into three pieces  $s = xyz$  such that

- for every  $i > 0$ ,  $xy^iz \in L$ ,
- $|y| > 0$ , and
- $|xy| \leq \ell$ .

**Remarks:** Without the second condition, the theorem would be trivial.

The third condition is technical and useful occasionally.

# Pumping Lemma – Proof

Let  $M = (Q, \Sigma, \delta, q_1, F)$  be a DFA that accepts  $L$ .

Let  $\ell$  be  $|Q|$ , the number of states of  $M$ .

# Pumping Lemma – Proof

Let  $M = (Q, \Sigma, \delta, q_1, F)$  be a DFA that accepts  $L$ .

Let  $\ell$  be  $|Q|$ , the number of states of  $M$ .

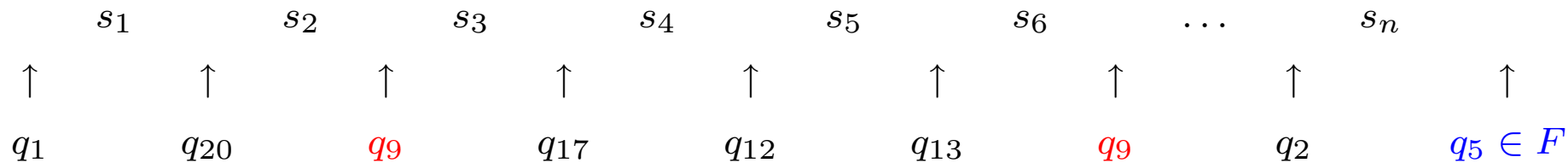
If  $s \in L$  has length at least  $\ell$ , consider the sequence of states  $M$  goes through as it reads  $s$ :

# Pumping Lemma – Proof

Let  $M = (Q, \Sigma, \delta, q_1, F)$  be a DFA that accepts  $L$ .

Let  $\ell$  be  $|Q|$ , the number of states of  $M$ .

If  $s \in L$  has length at least  $\ell$ , consider the sequence of states  $M$  goes through as it reads  $s$ :

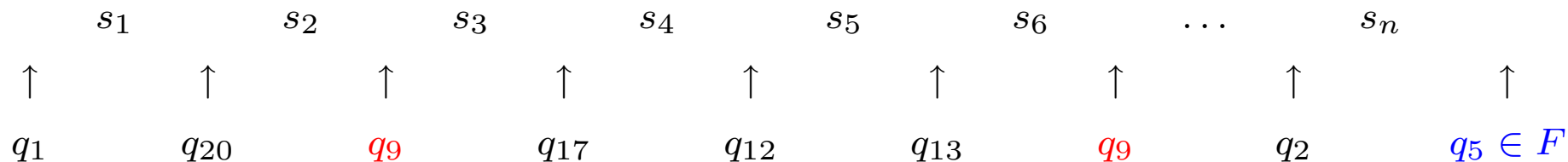


# Pumping Lemma – Proof

Let  $M = (Q, \Sigma, \delta, q_1, F)$  be a DFA that accepts  $L$ .

Let  $\ell$  be  $|Q|$ , the number of states of  $M$ .

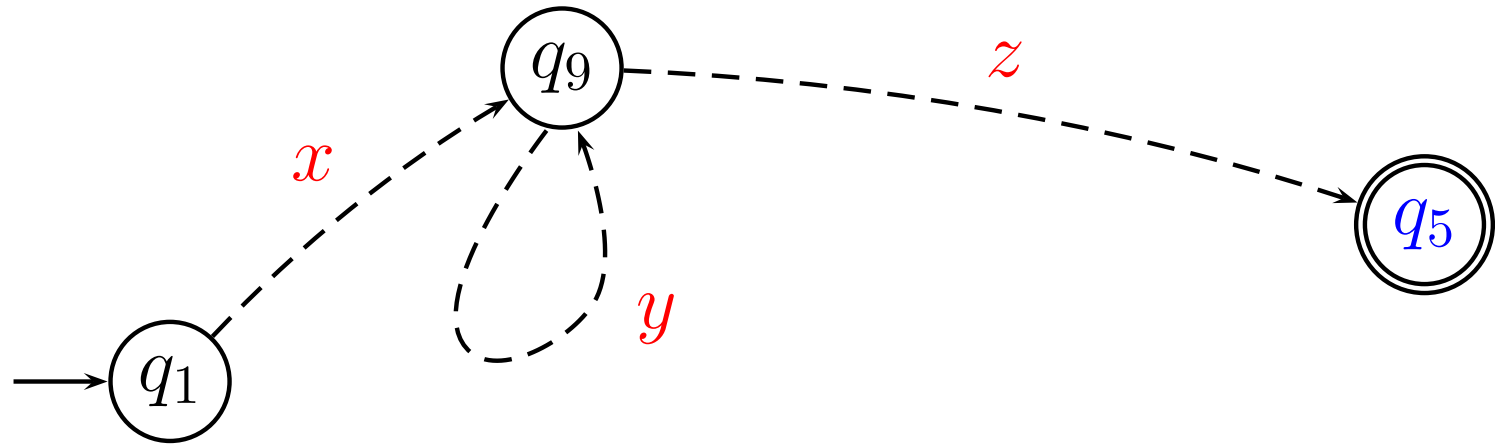
If  $s \in L$  has length at least  $\ell$ , consider the sequence of states  $M$  goes through as it reads  $s$ :



Since the sequence of states is of length  $|s| + 1 > \ell$ , and there are only  $\ell$  different states in  $Q$ , at least one state is repeated (by the **pigeonhole principle**).

# Pumping Lemma – Proof (cont.)

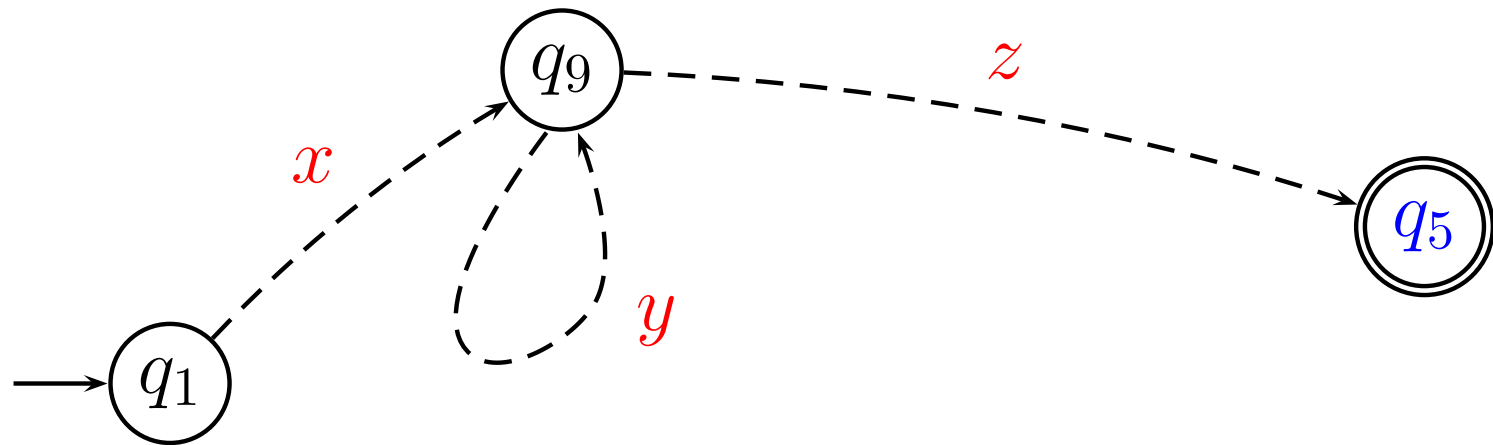
Write down  $s = xyz$



- By inspection,  $M$  accepts  $xy^kz$  for every  $k \geq 0$ .

# Pumping Lemma – Proof (cont.)

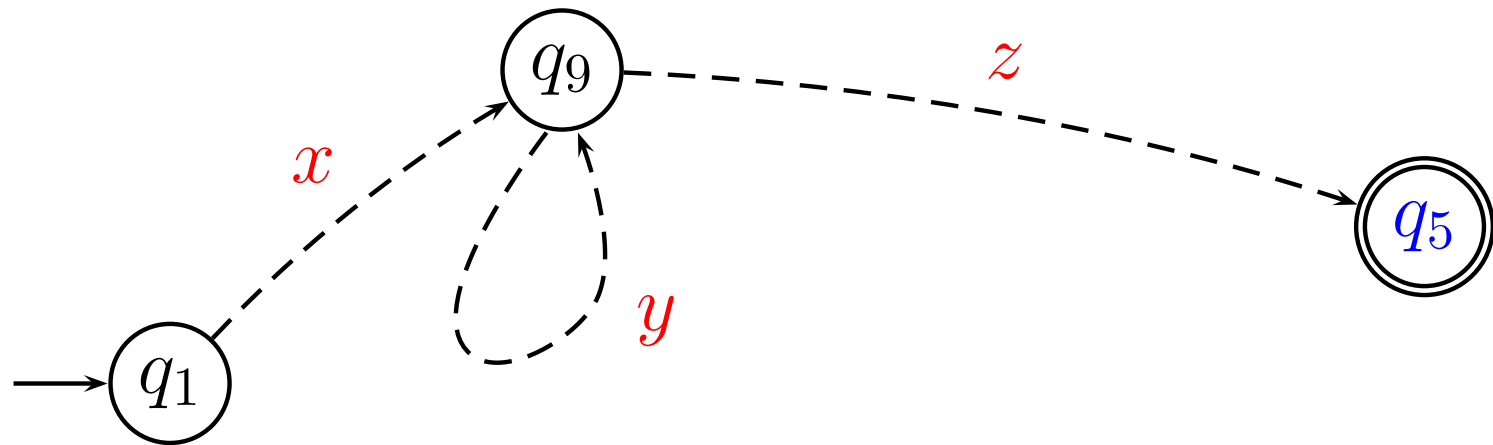
Write down  $s = xyz$



- By inspection,  $M$  accepts  $xy^kz$  for every  $k \geq 0$ .
- $|y| > 0$  because the state ( $q_9$  in figure) is repeated.

# Pumping Lemma – Proof (cont.)

Write down  $s = xyz$



- By inspection,  $M$  accepts  $xy^kz$  for every  $k \geq 0$ .
- $|y| > 0$  because the state ( $q_9$  in figure) is repeated.
- To ensure that  $|xy| \leq \ell$ , pick first state repetition, which must occur no later than  $\ell + 1$  states in sequence.

# An Application

**Theorem:** The language  $B = \{0^n 1^n \mid n > 0\}$  is not regular.

**Proof:** By contradiction. Suppose  $B$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .

# An Application

**Theorem:** The language  $B = \{0^n 1^n \mid n > 0\}$  is not regular.

**Proof:** By contradiction. Suppose  $B$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^k z \in B$  for every  $k$ .

# An Application

**Theorem:** The language  $B = \{0^n 1^n \mid n > 0\}$  is not regular.

**Proof:** By contradiction. Suppose  $B$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^k z \in B$  for every  $k$ .
- If  $y$  is all 0, then  $xy^k z$  has too many 0's.

# An Application

**Theorem:** The language  $B = \{0^n 1^n \mid n > 0\}$  is not regular.

**Proof:** By contradiction. Suppose  $B$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^k z \in B$  for every  $k$ .
- If  $y$  is all 0, then  $xy^k z$  has too many 0's.
- If  $y$  is all 1, then  $xy^k z$  has too many 1's.

# An Application

**Theorem:** The language  $B = \{0^n 1^n \mid n > 0\}$  is not regular.

**Proof:** By contradiction. Suppose  $B$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^k z \in B$  for every  $k$ .
- If  $y$  is all 0, then  $xy^k z$  has too many 0's.
- If  $y$  is all 1, then  $xy^k z$  has too many 1's.
- If  $y$  is mixed, then  $xy^k z$  is not of right form. ♣

# Another Application

**Theorem:** The language

$C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$  is not regular.

**Proof:** By contradiction. Suppose  $C$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .

# Another Application

**Theorem:** The language

$C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$  is not regular.

**Proof:** By contradiction. Suppose  $C$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^kz \in C$  for every  $k$ .

# Another Application

**Theorem:** The language

$C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$  is not regular.

**Proof:** By contradiction. Suppose  $C$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^kz \in C$  for every  $k$ .
- If  $y$  is all 0, then  $xy^kz$  has too many 0's.

# Another Application

**Theorem:** The language

$C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$  is not regular.

**Proof:** By contradiction. Suppose  $C$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^kz \in C$  for every  $k$ .
- If  $y$  is all 0, then  $xy^kz$  has too many 0's.
- If  $y$  is all 1, then  $xy^kz$  has too many 1's.

# Another Application

**Theorem:** The language

$C = \{w \mid w \text{ has an equal number of 0's and 1's}\}$  is not regular.

**Proof:** By contradiction. Suppose  $C$  is regular, accepted by DFA  $M$ . Let  $\ell$  be the pumping length.

- Consider the string  $s = 0^\ell 1^\ell$ .
- By pumping lemma  $s = xyz$ , where  $xy^kz \in C$  for every  $k$ .
- If  $y$  is all 0, then  $xy^kz$  has too many 0's.
- If  $y$  is all 1, then  $xy^kz$  has too many 1's.
- If  $y$  is mixed, then since  $|xy| \leq \ell$ ,  $y$  must be all 0's, contradiction.



# Algorithms for NDA's

Given an NDA,  $N$ , and a string  $s$ , is  $s \in L(N)$ ?

**Answer:** Construct the DFA equivalent to  $N$  and run it on  $w$ .

# Algorithms for NDA's

Given an NDA,  $N$ , and a string  $s$ , is  $s \in L(N)$ ?

**Answer:** Construct the DFA equivalent to  $N$  and run it on  $w$ .

Is  $L(N) = \emptyset$ ?

**Answer:** This is a **reachability** question in graphs: Is there a path in the states' graph of  $N$  from the start state to some accepting state. There are simple, efficient algorithms for this task.

# More Algorithms for NDA's

Is  $L(N) = \Sigma^*$ ?

Answer: Check if  $\overline{L(N)} = \emptyset$ .

# More Algorithms for NDA's

Is  $L(N) = \Sigma^*$ ?

Answer: Check if  $\overline{L(N)} = \emptyset$ .

Given  $N_1$  and  $N_2$ , is  $L(N_1) \subseteq L(N_2)$ ?

Answer: Check if  $\overline{L(N_2)} \cap L(N_1) = \emptyset$ .

# More Algorithms for NDA's

Is  $L(N) = \Sigma^*$ ?

Answer: Check if  $\overline{L(N)} = \emptyset$ .

Given  $N_1$  and  $N_2$ , is  $L(N_1) \subseteq L(N_2)$ ?

Answer: Check if  $\overline{L(N_2)} \cap L(N_1) = \emptyset$ .

Given  $N_1$  and  $N_2$ , is  $L(N_1) = L(N_2)$ ?

Answer: Check if  $L(N_1) \subseteq L(N_2)$  and  $L(N_2) \subseteq L(N_1)$ .