Chapter 1. Regular Languages

- 1.0 Why languages?
- 1.1 Finite automata
- 1.2 Nondeterminism
- 1.3 Regular expressions
- 1.4 Non-regular languages
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems.
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems.
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems

(1) computational problems
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems

(1) computational problems \iff \text{decision problems} \iff \text{languages}
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems

(1) computational problems $\iff$ decision problems
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems

(1) computational problems \iff decision problems

(2) decision problems
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems.

(1) computational problems ⇔ decision problems

(2) decision problems ⇔ languages
1.0 Why languages

Studying languages is a simple and rigorous way to investigate computational problems

(1) computational problems $\iff$ decision problems

(2) decision problems $\iff$ languages
1.0 Why languages

Two types of problems
(1) decision problems whose output is TRUE/FALSE (one bit);
(2) search problems whose output can be much longer

Is Sorting a decision or search problem?
1.0 Why languages

Two types of problems

(1) decision problems whose output is TRUE/FALSE (one bit);
(2) search problems whose output can be much longer

Is Sorting a decision or search problem?
1.0 Why languages

Two types of problems

(1) decision problems whose output is TRUE/FALSE (one bit);
1.0 Why languages

Two types of problems

(1) decision problems whose output is TRUE/FALSE (one bit);
(2) search problems whose output can be much longer
1.0 Why languages

Two types of problems

(1) decision problems whose output is TRUE/FALSE (one bit);
(2) search problems whose output can be much longer

Is Sorting a decision or search problem?
Two types of problems

(1) decision problems whose output is TRUE/FALSE (one bit);
(2) search problems whose output can be much longer

Is Sorting a decision or search problem?
1.0 Why languages
The following **Shortest Path** problem is a search problem.
1.0 Why languages

The following **Shortest Path** problem is a search problem

**Shortest Path**

Input: graph $G = (V, E)$ and vertices $s, t \in V$

Output: a path connecting $s$ and $t$ that contains the fewest edges
The following **Shortest Path** problem is a search problem.

**Shortest Path**

*Input*: graph $G = (V, E)$ and vertices $s, t \in V$

*Output*: a path connecting $s$ and $t$ that contains the fewest edges

For input $G$: and $s = a, t = f,$
1.0 Why languages

The following Shortest Path problem is a search problem.

**Shortest Path**

Input: graph $G = (V, E)$ and vertices $s, t \in V$
Output: a path connecting $s$ and $t$ that contains the fewest edges

For input $G$: and $s = a, t = f$,
the output should be

either $(a, b), (b, f)$,
The following **Shortest Path** problem is a search problem.

**Shortest Path**
- **Input:** graph $G = (V, E)$ and vertices $s, t \in V$
- **Output:** a path connecting $s$ and $t$ that contains the fewest edges.

For input $G$: and $s = a$, $t = f$, the output should be

either $(a, b), (b, f)$, or $(a, e), (e, f)$. 

1.0 Why languages

Consider the following decision problem:

Shortest Path Length

Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;

Output: TRUE if and only if the shortest path from $s$ to $t$ contains $k$ edges.

For input $G$, $s = a$, $t = f$, and $k = 2$, the output should be TRUE; but for input $G$, $s = a$, $t = c$, and $k = 3$, the output should be FALSE.

Can an algorithms for Shortest Path Length and for Shortest Path help each other? YES!
1.0 Why languages

Consider the following decision problem:

**Shortest Path Length**

Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
Output: **TRUE if and only if** the shortest path from $s$ to $t$ contains $k$ edges.
1.0 Why languages

Consider the following decision problem:

**Shortest Path Length**

- **Input:** graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
- **Output:** **TRUE** if and only if the shortest path from $s$ to $t$ contains $k$ edges.

For input $G$: $s = a$, $t = f$, and $k = 2$
1.0 Why languages

Consider the following decision problem:

**Shortest Path Length**

Input: graph \( G = (V, E) \), vertices \( s, t \in V \) and integer \( k \geq 0 \);
Output: \textbf{TRUE if and only if} the shortest path from \( s \) to \( t \) contains \( k \) edges.

For input \( G \):

\[ \begin{align*}
&\text{the output should be } \textbf{TRUE}; \\
&\text{, } s = a, t = f, \text{ and } k = 2
\end{align*} \]
1.0 Why languages

Consider the following decision problem:

**Shortest Path Length**
- Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
- Output: **TRUE if and only if** the shortest path from $s$ to $t$ contains $k$ edges.

For input $G$, $s = a$, $t = f$, and $k = 2$, the output should be **TRUE**;

But for input $G$, $s = a$, $t = c$, and $k = 3$, the output should be **FALSE**.
1.0 Why languages

Consider the following decision problem:

**Shortest Path Length**

Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
Output: **TRUE if and only if** the shortest path from $s$ to $t$ contains $k$ edges.

For input $G$: $s = a$, $t = f$, and $k = 2$
the output should be **TRUE**;

But for input $G$, $s = a$, $t = c$, and $k = 3$, the output should be **FALSE**.

Can an algorithms for **Shortest Path Length** and for **Shortest Path** help each other?
1.0 Why languages

Consider the following decision problem:

**Shortest Path Length**
- Input: graph \( G = (V, E) \), vertices \( s, t \in V \) and integer \( k \geq 0 \);
- Output: **TRUE if and only if** the shortest path from \( s \) to \( t \) contains \( k \) edges.

For input \( G \):
- \( s = a, t = f, \) and \( k = 2 \)
the output should be **TRUE**;

But for input \( G \), \( s = a, t = c, \) and \( k = 3 \), the output should be **FALSE**.

Can an algorithms for **Shortest Path Length** and for **Shortest Path** help each other? 

**YES!**
1.0 Why languages

We want

Assume we have an algorithm

Oracle

A

that can decide whether

the shortest path between vertices

ts and t

in graph

G

contains

k

edges

A(G,s,t,k) =

Yes shortest path between s and t contains k edges

No otherwise
1.0 Why languages

Assume we have an algorithm **Oracle** $A$ that can decide whether the shortest path between vertices $s$ and $t$ in graph $G$ contains $k$ edges

$$A(G, s, t, k) = \begin{cases} 
    \text{Yes} & \text{shortest path between } s \text{ and } t \text{ contains } k \text{ edges} \\
    \text{No} & \text{otherwise}
\end{cases}$$
1.0 Why languages

Assume we have an algorithm **Oracle** \( A \) that can decide whether the shortest path between vertices \( s \) and \( t \) in graph \( G \) contains \( k \) edges

\[
A(G, s, t, k) = \begin{cases} 
\text{Yes} & \text{shortest path between } s \text{ and } t \text{ contains } k \text{ edges} \\
\text{No} & \text{otherwise}
\end{cases}
\]
1.0 Why languages

The following process $B(G,s,t)$ finds the shortest path between $s$ and $t$ in the input graph $G$:

1. Input: graph $G$ and two vertices $s$ and $t$;
2. for $k = 1, 2, ..., n - 1$, where $n = |V|$;
   2.1 ask Oracle $A(G,s,t,k)$;
   2.2 if $A(G,s,t,k) = Yes$, set $k_0 = k$; exit from the loop;
3. while $G$ contains unmarked edges do
   3.1 pick any unmarked edge $e$ and remove it from $G$;
   3.2 if $A(G,s,t,k_0) = No$, restore $e$ in $G$ and marked $e$;
4. return (marked edges) (which form a shortest path from $s$ and $t$).
1.0 Why languages

The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$ in the input graph $G$;
The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$ in the input graph $G$;

1. Input: graph $G$ and two vertices $s$ and $t;$
The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$ in the input graph $G$;

1. Input: graph $G$ and two vertices $s$ and $t$;

2. for $k = 1, 2, \ldots, n - 1$, where $n = |V|$;
   2.1 ask Oracle $A(G, s, t, k)$;
1.0 Why languages

The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$ in the input graph $G$:

1. Input: graph $G$ and two vertices $s$ and $t$;

2. for $k = 1, 2, \ldots, n - 1$, where $n = |V|$;
   2.1 ask Oracle $A(G, s, t, k)$;
   2.2 if $A(G, s, t, k) = \text{Yes}$, set $k_0 = k$; exit from the loop

3. while $G$ contains unmarked edges do
   3.1 pick any unmarked edge $e$ and remove it from $G$;
   3.2 if $A(G, s, t, k_0) = \text{No}$, restore $e$ in $G$ and marked $e$;

4. return (marked edges) (which form a shortest path from $s$ and $t$)
1.0 Why languages

The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$
in the input graph $G$;

1. Input: graph $G$ and two vertices $s$ and $t$;

2. for $k = 1, 2, \ldots, n - 1$, where $n = |V|$;
   2.1 ask Oracle $A(G, s, t, k)$;
   2.2 if $A(G, s, t, k) = \text{Yes}$, set $k_0 = k$; exit from the loop

3. while $G$ contains unmarked edges do

4. return (marked edges) (which form a shortest path from $s$ and $t$)
1.0 Why languages

The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$ in the input graph $G$;

1. Input: graph $G$ and two vertices $s$ and $t$;

2. for $k = 1, 2, \ldots, n - 1$, where $n = |V|$;
   2.1 ask Oracle $A(G, s, t, k)$;
   2.2 if $A(G, s, t, k) = \text{Yes}$, set $k_0 = k$; exit from the loop

3. while $G$ contains unmarked edges do
   3.1 pick any unmarked edge $e$ and remove it from $G$;
1.0 Why languages

The following process $B(G, s, t)$ finds the shortest path between $s$ and $t$ in the input graph $G$;

1. Input: graph $G$ and two vertices $s$ and $t$;

2. for $k = 1, 2, \ldots, n - 1$, where $n = |V|$;
   2.1 ask Oracle $A(G, s, t, k)$;
   2.2 if $A(G, s, t, k) = \text{Yes}$, set $k_0 = k$; exit from the loop

3. while $G$ contains unmarked edges do
   3.1 pick any unmarked edge $e$ and remove it from $G$;
   3.2 if $A(G, s, t, k_0) = \text{No}$, restore $e$ in $G$ and marked $e$;

4. return (marked edges) (which form a shortest path from $s$ and $t$)
1.0 Why languages

The following process \( B(G, s, t) \) finds the shortest path between \( s \) and \( t \) in the input graph \( G \);

1. Input: graph \( G \) and two vertices \( s \) and \( t \);

2. for \( k = 1, 2, \ldots, n - 1 \), where \( n = |V| \);
   2.1 ask Oracle \( A(G, s, t, k) \);
   2.2 if \( A(G, s, t, k) = \text{Yes} \), set \( k_0 = k \); exit from the loop

3. while \( G \) contains unmarked edges do
   3.1 pick any unmarked edge \( e \) and remove it from \( G \);
   3.2 if \( A(G, s, t, k_0) = \text{No} \), restore \( e \) in \( G \) and marked \( e \);

4. return (marked edges) (which form a shortest path from \( s \) and \( t \))
1.0 Why languages
1.0 Why languages

We observe that

- The solution to the decision problem Shortest Path Length is essential to the solution to the search problem Shortest Path.
- In general, the solution to a search problem is underscored by the solution to its corresponding decision problem.
- It suffices to focus on investigating decision problems.
1.0 Why languages

We observe that

- solution to the decision problem **Shortest Path Length** is essential to solution to the search problem **Shortest Path**
1.0 Why languages

We observe that

- solution to the decision problem Shortest Path Length is essential to solution to the search problem Shortest Path

- in general, the solution to a search problem is underscored by the solution to its corresponding decision problem
1.0 Why languages

We observe that

- solution to the decision problem Shortest Path Length is essential to solution to the search problem Shortest Path

- in general, the solution to a search problem is underscored by the solution to its corresponding decision problem

- it suffices to focus on investigating decision problems
1.0 Why languages

Now we relate decision problems to languages. There is a one-to-one correspondence between the two. Specifically, for every decision problem $\Pi$, we can define a language $L^\Pi$ such that any algorithm recognizing the language $L^\Pi$ can be used to solve the decision problem $\Pi$; and vice versa.
1.0 Why languages

Now we relate decision problems to languages
1.0 Why languages

Now we relate decision problems to languages.

There is one-to-one correspondence between the two.
1.0 Why languages

Now we relate decision problems to languages. There is one-to-one correspondence between the two. Specifically, for every decision problem $\Pi$, we can define a language $L_\Pi$ such that...
Now we relate decision problems to languages. There is one-to-one correspondence between the two. Specifically, for every decision problem $\Pi$, we can define a language $L_\Pi$ such that any algorithm recognizing the language $L_\Pi$ can be used to solve the decision problem $\Pi$.
1.0 Why languages

Now we relate decision problems to languages.

There is one-to-one correspondence between the two. Specifically,

for every decision problem $\Pi$, we can define a language $L_\Pi$

such that

any algorithm recognizing the language $L_\Pi$ can be used to
solve the decision problem $\Pi$; and vice versa.
1.0 Why languages

For decision problem Shortest Path Length:

Shortest Path Length (abbr. SPL)

Input: graph \( G = (V,E) \), vertices \( s,t \in V \) and integer \( k \geq 0 \);

Output: TRUE if and only if the shortest path from \( s \) to \( t \) contains \( k \) edges.

We define language \( L_{SPL} = \{ \langle G,s,t,k \rangle : \text{shortest path from } s \text{ to } t \text{ has } k \text{ edges in } G \} \) where \( \langle G,s,t,k \rangle \) is a string that encodes graph \( G \), vertices \( s \), \( t \), and integer \( k \).

That is, \( L_{SPL} \) contains exactly positive instances of decision problem SPL.

Algorithms for SPL can be used to recognize strings in \( L_{SPL} \), and vice versa.
1.0 Why languages

For decision problem Shortest Path Length:

Shortest Path Length (abbr. SPL)

Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
Output: TRUE if and only if the shortest path from $s$ to $t$ contains $k$ edges.
1.0 Why languages

For decision problem **Shortest Path Length**:

**Shortest Path Length (abbr. SPL)**
- Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
- Output: TRUE if and only if the shortest path from $s$ to $t$ contains $k$ edges.

we define language

$$L_{SPL} = \{ \langle G, s, t, k \rangle : \text{shortest path from } s \text{ to } t \text{ has } k \text{ edges in } G \}$$
1.0 Why languages

For decision problem **Shortest Path Length**:

**Shortest Path Length (abbr. SPL)**

Input: graph \( G = (V, E) \), vertices \( s, t \in V \) and integer \( k \geq 0 \);
Output: **TRUE** if and only if the shortest path from \( s \) to \( t \) contains \( k \) edges.

we define language

\[
L_{\text{SPL}} = \{ \langle G, s, t, k \rangle : \text{shortest path from } s \text{ to } t \text{ has } k \text{ edges in } G \}
\]

where \( \langle G, s, t, k \rangle \) is a string that encodes graph \( G \), vertices \( s, t \), and integer \( k \).
For decision problem **Shortest Path Length**:

**Shortest Path Length** (abbr. SPL)
- Input: graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
- Output: TRUE if and only if the shortest path from $s$ to $t$ contains $k$ edges.

we define language

$$L_{SPL} = \{ \langle G, s, t, k \rangle : \text{shortest path from } s \text{ to } t \text{ has } k \text{ edges in } G \}$$

where $\langle G, s, t, k \rangle$ is a string that encodes graph $G$, vertices $s$, $t$, and integer $k$.

- That is, $L_{SPL}$ contains exactly **positive instances** of decision problem SPL.
1.0 Why languages

For decision problem **Shortest Path Length**:

**Shortest Path Length** (abbr. SPL)

- **Input:** graph $G = (V, E)$, vertices $s, t \in V$ and integer $k \geq 0$;
- **Output:** TRUE if and only if the shortest path from $s$ to $t$ contains $k$ edges.

we define language

$$L_{\text{SPL}} = \{⟨G, s, t, k⟩ : \text{shortest path from } s \text{ to } t \text{ has } k \text{ edges in } G\}$$

where $⟨G, s, t, k⟩$ is a string that encodes graph $G$, vertices $s$, $t$, and integer $k$.

- That is, $L_{\text{SPL}}$ contains exactly **positive instances** of decision problem SPL.
- Algorithms for SPL can be used to recognize strings in $L_{\text{SPL}}$, and vice versa.
1.0 Why languages

For example, the language $\mathcal{L}$ contains the strings like $\langle G_1, A, P, 2 \rangle$, $\langle G_1, C, L, 3 \rangle$, $\langle G_2, 1, 4, 3 \rangle$, $\cdots$ where graphs $G_1$ and $G_2$ but not strings like $\langle G_1, B, S, 3 \rangle$ or $\langle G_2, 2, 3, 1 \rangle$. 
1.0 Why languages

For example, the language $L_{SPL}$ contains the strings like

$$\langle G_1, A, P, 2 \rangle, \langle G_1, C, L, 3 \rangle, \langle G_2, 1, 4, 3 \rangle, \cdots$$

where graphs $G_1$ and $G_2$ are
1.0 Why languages

For example, the language $L_{\text{SPL}}$ contains the strings like

$$\langle G_1, A, P, 2 \rangle, \langle G_1, C, L, 3 \rangle, \langle G_2, 1, 4, 3 \rangle, \cdots$$

where graphs $G_1$ and $G_2$ are

![Diagram of graphs $G_1$ and $G_2$]
1.0 Why languages

For example, the language $L_{SPL}$ contains the strings like

$$\langle G_1, A, P, 2 \rangle, \langle G_1, C, L, 3 \rangle, \langle G_2, 1, 4, 3 \rangle, \cdots$$

where graphs $G_1$ and $G_2$ are

but not strings like $\langle G_1, B, S, 3 \rangle$ or $\langle G_2, 2, 3, 1 \rangle$, 
1.0 Why languages

We conclude: it suffices to investigate languages and power of computational models to recognize languages.
1.0 Why languages

We conclude: it suffices to investigate languages and power of computational models to recognize languages.
1.0 Why languages

We conclude: it suffices to investigate languages and power of computational models to recognize languages.
1.0 Why languages

Let $\Sigma$ be a finite alphabet, e.g., $\Sigma = \{0, 1\}$; we will consider languages $L$ defined over $\Sigma$. That is $L \subseteq \Sigma^*$. We investigate classes of languages by examining different types of computational models that can be programmed to compute them.
Let $\Sigma$ be a finite alphabet, e.g., $\Sigma = \{0, 1\}$;

• we will consider languages $L$ defined over $\Sigma$. That is

\[ L \subseteq \Sigma^* \]
1.0 Why languages

Let $\Sigma$ be a finite alphabet, e.g., $\Sigma = \{0,1\}$;

- we will consider languages $L$ defined over $\Sigma$. That is

$$L \subseteq \Sigma^*$$

- We investigate classes of languages by examining different types of computational models that can be programmed to compute them.
1.0 Why languages

All such models have the following:

• an read-(from left to right)-only input buffer – called input tape
• a finite set of states, some are accept states
• model accepts a string iff it halts at an accept state

Different models may have different extra memory "facilities" to use.
1.0 Why languages

All such models have the following:
1.0 Why languages

All such models have the following:

- an read-(from left to right)-only input buffer – called *input tape*
1.0 Why languages

All such models have the following:

- an read-(from left to right)-only input buffer – called *input tape*
- a finite set of states, some are *accept* states
1.0 Why languages

All such models have the following:

- an read-(from left to right)-only input buffer – called *input tape*
- a finite set of states, some are *accept* states
- model accepts a string iff it halts at an accept state
1.0 Why languages

All such models have the following:

- an read-(from left to right)-only input buffer – called input tape
- a finite set of states, some are accept states
- model accepts a string iff it halts at an accept state
1.0 Why languages

All such models have the following:

- an read-(from left to right)-only input buffer – called *input tape*
- a finite set of states, some are *accept* states
- model accepts a string iff it halts at an accept state

Different models may have different extra memory "facilities" to use.
1.1 Finite automata

A finite automaton that has no extra memory facility.

1. A finite number of states, some “accept state”
2. It reads one symbol at a time from the input (left to right)
3. The next state is determined by the current state and the symbol just read
4. The input string is accepted if it ends at an “accept” state.
5. Otherwise, the input is “rejected.”

What strings does the FA accept?
1.1 Finite automata

finite automaton that has no extra memory facility.
1.1 Finite automata

finite automaton that has no extra memory facility.
1.1 Finite automata

finite automaton that has no extra memory facility.

What strings does the FA accept?

(1) a finite number of states, some “accept state”
1.1 Finite automata

finite automaton that has no extra memory facility.

(1) a finite number of states, some “accept state”
(2) it reads one symbol at a time from the input (left to right)
1.1 Finite automata

finite automaton that has no extra memory facility.

(1) a finite number of states, some “accept state”
(2) it reads one symbol at a time from the input (left to right)
(3) the next state is determined by the current state and the symbol just read
1.1 Finite automata

A finite automaton that has no extra memory facility.

(1) a finite number of states, some “accept state”
(2) it reads one symbol at a time from the input (left to right)
(3) the next state is determined by the current state and the symbol just read
(4) the input string is accepted if it ends at an “accept” state.
1.1 Finite automata

finite automaton that has no extra memory facility.

(1) a finite number of states, some “accept state”
(2) it reads one symbol at a time from the input (left to right)
(3) the next state is determined by the current state and the symbol just read
(4) the input string is accepted if it ends at an “accept” state.
(5) otherwise, the input is “rejected”.

What strings does the FA accept?
1.1 Finite automata

A finite automaton is a finite automaton that has no extra memory facility.

(1) a finite number of states, some “accept state”
(2) it reads one symbol at a time from the input (left to right)
(3) the next state is determined by the current state and the symbol just read
(4) the input string is accepted if it ends at an “accept” state.
(5) otherwise, the input is “rejected”.

What strings does the FA accept?
1.1 Finite automata

transition function:
\[ Q \times \Sigma \rightarrow Q \]
where
\[ Q = \{ a, b, c, d \} \]
and
\[ \Sigma = \{ 0, 1 \} \]
1.1 Finite automata

description of state transitions
1.1 Finite automata

description of state transitions
1.1 Finite automata

description of state transitions

\[(a, 1) \rightarrow a\]
1.1 Finite automata

description of state transitions

(a, 1) → a  (a, 0) → b
1.1 Finite automata

description of state transitions

\[(a, 1) \Rightarrow a \quad (a, 0) \Rightarrow b \quad (b, 0) \Rightarrow b\]
1.1 Finite automata

description of state transitions

\[(a, 1) \rightarrow a \quad (a, 0) \rightarrow b \quad (b, 0) \rightarrow b \quad (b, 1) \rightarrow c\]
1.1 Finite automata

description of state transitions

\[(a, 1) \rightarrow a\]  \[(a, 0) \rightarrow b\]  \[(b, 0) \rightarrow b\]  \[(b, 1) \rightarrow c\]  
\[(c, 1) \rightarrow a\]
1.1 Finite automata

description of state transitions

\[ (a, 1) \rightarrow a \quad (a, 0) \rightarrow b \quad (b, 0) \rightarrow b \quad (b, 1) \rightarrow c \]
\[ (c, 1) \rightarrow a \quad (c, 0) \rightarrow d \]
1.1 Finite automata

description of state transitions

transition function: $Q \times \Sigma \rightarrow Q$
where $Q = \{a, b, c, d\}$ and $\Sigma = \{0, 1\}$

$(a, 1) \rightarrow a$  $(a, 0) \rightarrow b$  $(b, 0) \rightarrow b$  $(b, 1) \rightarrow c$
$(c, 1) \rightarrow a$  $(c, 0) \rightarrow d$  $(d, 1) \rightarrow d$
1.1 Finite automata

description of state transitions

\[
\begin{align*}
(a, 1) &\rightarrow a & (a, 0) &\rightarrow b & (b, 0) &\rightarrow b & (b, 1) &\rightarrow c \\
(c, 1) &\rightarrow a & (c, 0) &\rightarrow d & (d, 1) &\rightarrow d & (d, 0) &\rightarrow d
\end{align*}
\]
1.1 Finite automata

description of state transitions

\[
\begin{align*}
(a, 1) & \rightarrow a & (a, 0) & \rightarrow b & (b, 0) & \rightarrow b & (b, 1) & \rightarrow c \\
(c, 1) & \rightarrow a & (c, 0) & \rightarrow d & (d, 1) & \rightarrow d & (d, 0) & \rightarrow d
\end{align*}
\]

transition function: \( Q \times \Sigma \rightarrow Q \)

where \( Q = \{a, b, c, d\} \) and \( \Sigma = \{0, 1\} \)
1.1 Finite automata

What strings does $M_1$ accept?
1.1 Finite automata

Another FA $M_1$
1.1 Finite automata

Another FA $M_1$

What strings does $M_1$ accept?
1.1 Finite automata

Formal definition of FA

Definition 1.5 (page 35)

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the states,
2. $\Sigma$ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.
1.1 Finite automata

Formal definition of FA

Definition 1.5 (page 35)
1.1 Finite automata

Formal definition of FA

**Definition 1.5** (page 35)

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where
1.1 Finite automata

Formal definition of FA

**Definition 1.5** (page 35)

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
1.1 Finite automata

Formal definition of FA

Definition 1.5 (page 35)

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the states,
2. \(\Sigma\) is a finite set called the alphabet,
1.1 Finite automata

Formal definition of FA

**Definition 1.5** (page 35)

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta : Q \times \Sigma \rightarrow Q\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*.
1.1 Finite automata

Formal definition of FA

**Definition 1.5** (page 35)

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta : Q \times \Sigma \rightarrow Q\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*. 
1.1 Finite automata

Formal definition of FA

**Definition 1.5** (page 35)

A *finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta : Q \times \Sigma \rightarrow Q\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*. 
1.1 Finite automata

For FA $M_1$, the formal description is $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $F = \{q_1\}$

$\delta(q_0, 1) = q_0$, $\delta(q_0, 0) = q_1$,

$\delta(q_1, 1) = q_1$, $\delta(q_1, 0) = q_2$,

$\delta(q_2, 0) = q_1$, $\delta(q_2, 1) = q_1$
1.1 Finite automata

For FA $M$, the formal description is $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $F = \{q_1\}$.

$\delta(q_0, 1) = q_0$, $\delta(q_0, 0) = q_1$

$\delta(q_1, 1) = q_1$, $\delta(q_1, 0) = q_2$

$\delta(q_2, 0) = q_1$, $\delta(q_2, 1) = q_1$
1.1 Finite automata

For FA $M_1$, the formal description is

\[ Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1\}, F = \{q_1\} \]
1.1 Finite automata

For FA $M_1$, the formal description is

$$Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1\}, F = \{q_1\}$$

$$\delta(q_0, 1) = q_0, \delta(q_0, 0) = q_1, \delta(q_1, 1) = q_1$$
1.1 Finite automata

For FA $M_1$, the formal description is

$$Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1\}, F = \{q_1\}$$

$$\delta(q_0, 1) = q_0, \delta(q_0, 0) = q_1, \delta(q_1, 1) = q_1$$

$$\delta(q_1, 0) = q_2, \delta(q_2, 0) = q_1, \delta(q_2, 1) = q_1$$
1.1 Finite automata

More FA examples

FA $M_2$ (Figure 1.8 page 37).

$L_{M_2} = \{w : \text{string } w \text{ ends with symbol } a\}$

FA $M_3$ (Figure 1.10, page 38)

$L_{M_3} = \{w : \text{string } w \text{ contains even number of } 0's\}$
1.1 Finite automata

More FA examples
1.1 Finite automata

More FA examples

FA $M_2$ (Figure 1.8 page 37).
1.1 Finite automata

More FA examples

FA $M_2$ (Figure 1.8 page 37).

![Finite automaton diagram]

$L_{M_2} = \{w : \text{string } w \text{ ends with symbol } a \}$

FA $M_3$ (Figure 1.10, page 38)

![Finite automaton diagram]
1.1 Finite automata

More FA examples

FA $M_2$ (Figure 1.8 page 37).

$L_{M_2} = \{ w : \text{ string } w \text{ ends with symbol } a \} $

FA $M_3$ (Figure 1.10, page 38)

$L_{M_3} = \{ w : \text{ string } w \text{ contains even number of 0's} \} $
1.1 Finite automata

FA $M_4$ (Figure 1.12 page 38).
1.1 Finite automata

FA $M_4$ (Figure 1.12 page 38).

$L_{M_4} = \{w : w \text{ starts and ends with the same symbol}\}$
1.1 Finite automata

FA $M_5$ (Figure 1.13 page 39).
1.1 Finite automata

FA $M_5$ (Figure 1.13 page 39).

$L_{M_5} = \{ w : \text{the sum of symbols in } w \text{ is multiple of 3} \}$
1.1 Finite automata

Example 1.15 page 40, Similar to $M_5$ but modulo $i$ instead of 3,
Example 1.15 page 40, Similar to $M_5$ but modulo $i$ instead of 3,
Can you draw an FA for $i = 4$?
1.1 Finite automata

Formal definition of FA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. Let $w = w_1w_2...w_n$ be a string where each symbol $w_i \in \Sigma$. Then $M$ accepts $w$ if a sequence of states $r_0, r_1, ..., r_n$ in $Q$ exist with three conditions:

1. $r_0 = q_0$;
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, ..., n-1$,
3. $r_n \in F$.

We say $M$ recognizes language $L$ if $L = \{ w : w \in \Sigma^* \text{ and } M \text{ accepts } w \}$.
1.1 Finite automata

Formal definition of FA computation
1.1 Finite automata

Formal definition of FA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.
1.1 Finite automata

Formal definition of FA computation

Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a finite automaton.

Let \( w = w_1w_2 \ldots w_n \) be a string where each symbol \( w_i \in \Sigma \).
1.1 Finite automata

Formal definition of FA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w = w_1w_2 \ldots w_n$ be a string where each symbol $w_i \in \Sigma$.

Then $M$ accepts $w$ if a sequence states $r_0, r_1, \ldots, r_n$ in $Q$ exist with three conditions:
1.1 Finite automata

Formal definition of FA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w = w_1w_2 \ldots w_n$ be a string where each symbol $w_i \in \Sigma$.

Then $M$ accepts $w$ if a sequence states $r_0, r_1, \ldots, r_n$ in $Q$ exist with three conditions:

1. $r_0 = q_0$;
1.1 Finite automata

Formal definition of FA computation

Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a finite automaton.

Let \( w = w_1w_2 \ldots w_n \) be a string where each symbol \( w_i \in \Sigma \).

Then \( M \) accepts \( w \) if a sequence states \( r_0, r_1, \ldots, r_n \) in \( Q \) exist with three conditions:

1. \( r_0 = q_0 \);
2. \( \delta(r_i, w_{i+1}) = r_{i+1} \), for \( i = 0, 1, \ldots, n - 1 \), and
1.1 Finite automata

Formal definition of FA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w = w_1w_2 \ldots w_n$ be a string where each symbol $w_i \in \Sigma$.

Then $M$ accepts $w$ if a sequence states $r_0, r_1, \ldots, r_n$ in $Q$ exist with three conditions:

1. $r_0 = q_0$;
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, \ldots, n - 1$, and
3. $r_n \in F$. 

1.1 Finite automata

Formal definition of FA computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.

Let $w = w_1w_2 \ldots w_n$ be a string where each symbol $w_i \in \Sigma$.

Then $M$ accepts $w$ if a sequence states $r_0, r_1, \ldots, r_n$ in $Q$ exist with three conditions:

1. $r_0 = q_0$;
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, \ldots, n - 1$, and
3. $r_n \in F$.

We say $M$ recognizes language $L$ if

$$ L = \{ w : w \in \Sigma^* \text{ and } M \text{ accepts } w \} $$
1.1 Finite automata

Definition 1.16
A language is called a regular language if it can be recognized by some finite automaton.

Some basic questions:
1. Is $\{\epsilon\}$ regular?
2. Is $\Sigma^*$ regular?
3. Is $\emptyset$ regular?
4. Can two FA recognize the same language?
5. Is any finite language $L$ always regular?
6. Is a single string language regular?
1.1 Finite automata

Definition 1.16
A language is called a *regular language* if it can be recognized by some finite automaton.
1.1 Finite automata

Definition 1.16
A language is called a regular language if it can be recognized by some finite automaton.

Some basic questions:
1.1 Finite automata

**Definition 1.16**
A language is called a *regular language* if it can be recognized by some finite automaton.

Some basic questions:

1. Is \{\epsilon\} regular?
1.1 Finite automata

**Definition 1.16**
A language is called a *regular language* if it can be recognized by some finite automaton.

Some basic questions:

1. Is $\{\epsilon\}$ regular?
2. Is $\Sigma^*$ regular?
1.1 Finite automata

**Definition 1.16**
A language is called a *regular language* if it can be recognized by some finite automaton.

Some basic questions:

1. Is \( \{ \epsilon \} \) regular?
2. Is \( \Sigma^* \) regular?
3. Is \( \{ \} \) regular?
Definition 1.16
A language is called a *regular language* if it can be recognized by some finite automaton.

Some basic questions:

1. Is \( \{ \epsilon \} \) regular?
2. Is \( \Sigma^* \) regular?
3. Is \( \{ \} \) regular?
4. Can two FA recognize the same language?
1.1 Finite automata

**Definition 1.16**
A language is called a *regular language* if it can be recognized by some finite automaton.

Some basic questions:

1. Is \( \{ \epsilon \} \) regular?
2. Is \( \Sigma^* \) regular?
3. Is \( \{ \} \) regular?
4. Can two FA recognize the same language?
5. Is any finite language \( L \) always regular?
1.1 Finite automata

**Definition 1.16**
A language is called a *regular language* if it can be recognized by some finite automaton.

Some basic questions:

1. Is \( \{ \epsilon \} \) regular?
2. Is \( \Sigma^* \) regular?
3. Is \( \{ \} \) regular?
4. Can two FA recognize the same language?
5. Is any finite language \( L \) always regular?
6. Is a single string language regular?
1.1 Finite automata

- Design Finite Automata
- Some suggestions:
  - put yourself in the position of an automaton
  - have small amount of memory (which is a few states) to use
  - scan the input from left to right
  - e.g., construct an FA to recognize strings containing an odd number of 1s.
  - ignore 0s.
  - use two different states for odd and even 1s
  - let state change if additional 1 is encountered.
1.1 Finite automata

Design Finite Automata

- put yourself in the position of an automaton
- have small amount of memory (which is a few states) to use
- scan the input from left to right
e.g., construct an FA to recognize strings containing an odd number of 1s.
ignore 0s.
use two different states for odd and even 1s
let state change if additional 1 is encountered.
Design Finite Automata

Some suggestions:

- put yourself in the position of an automaton
1.1 Finite automata

Design Finite Automata

Some suggestions:

- put yourself in the position of an automaton
- have small amount of memory (which is a few states) to use
1.1 Finite automata

Design Finite Automata

Some suggestions:

• put yourself in the position of an automaton
• have small amount of memory (which is a few states) to use
• scan the input from left to right
1.1 Finite automata

**Design Finite Automata**

Some suggestions:

- put yourself in the position of an automaton
- have small amount of memory (which is a few states) to use
- scan the input from left to right

e.g., construct an FA to recognize strings containing an odd number 1s.
1.1 Finite automata

Design Finite Automata

Some suggestions:

- put yourself in the position of an automaton
- have small amount of memory (which is a few states) to use
- scan the input from left to right

e.g., construct an FA to recognize strings containing an odd number 1s. ignore 0s.
1.1 Finite automata

Design Finite Automata

Some suggestions:

- put yourself in the position of an automaton
- have small amount of memory (which is a few states) to use
- scan the input from left to right

E.g., construct an FA to recognize strings containing an odd number 1s.

  ignore 0s.
  use two different states for odd and even 1s
1.1 Finite automata

Design Finite Automata

Some suggestions:

• put yourself in the position of an automaton
• have small amount of memory (which is a few states) to use
• scan the input from left to right

  e.g., construct an FA to recognize strings containing an odd number 1s.
  
  ignore 0s.
  use two different states for odd and even 1s
  let state change if additional 1 is encountered.
1.1 Finite automata

Design Finite Automata

Some suggestions:

• put yourself in the position of an automaton
• have small amount of memory (which is a few states) to use
• scan the input from left to right

e.g., construct an FA to recognize strings containing an odd number 1s. ignore 0s.
use two different states for odd and even 1s
let state change if additional 1 is encountered.

![Finite automaton diagram]
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring. There are four possibilities:

• have not seen any symbols of the pattern 001
• have seen just a 0
• have seen 00
• have seen 001

You may use 4 states for these 4 possibilities, respectively.
e.g., construct an FA to recognize strings containing 001 as a substring.
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.
There are four possibilities:

- have not see any symbols of the pattern 001
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:

- have not seen any symbols of the pattern 001
- have seen just a 0
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:

- have not see any symbols of the pattern 001
- have seen just a 0
- have seen 00
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:

- have not see any symbols of the pattern 001
- have seen just a 0
- have seen 00
- have seen 001
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:

- have not see any symbols of the pattern 001
- have seen just a 0
- have seen 00
- have seen 001

You may use 4 states for these 4 possibilities, respectively.
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:

- have not seen any symbols of the pattern 001
- have seen just a 0
- have seen 00
- have seen 001

You may use 4 states for these 4 possibilities, respectively.
1.1 Finite automata

e.g., construct an FA to recognize strings containing 001 as a substring.

There are four possibilities:

- have not see any symbols of the pattern 001
- have seen just a 0
- have seen 00
- have seen 001

You may use 4 states for these 4 possibilities, respectively.
1.1 Finite automata

Creating languages through regular operations

- Union
- Concatenation
- Star
Creating languages through regular operations
1.1 Finite automata

Creating languages through regular operations

We will consider operations on languages, with
1.1 Finite automata

Creating languages through regular operations

We will consider operations on languages, with

UNION
CONCATENATION
STAR
1.1 Finite automata
1.1 Finite automata

e.g,

$L_1$ contains all strings with substring 11
1.1 Finite automata

e.g,

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000
1.1 Finite automata

e.g,

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000

Then

$L_1 \cup L_2 = \{ x : x \in L_1 \text{ or } x \in L_2 \}$
1.1 Finite automata

e.g,

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000

Then

$L_1 \cup L_2 = \{ x : x \in L_1 \text{ or } x \in L_2 \}$
$L_1L_2 = \{ x_1x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2 \}$
1.1 Finite automata

e.g,

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000

Then

$L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$
$L_1L_2 = \{x_1x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}$
$L_1^* = \{\epsilon\} \cup L_1 \cup L_1L_2 \cup L_1L_1L_1 \ldots$
1.1 Finite automata

e.g,

$L_1$ contains all strings with substring 11  
$L_2$ contains all strings with substring 000

Then

$L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$
$L_1L_2 = \{x_1x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}$
$L_1^* = \{\epsilon\} \cup L_1 \cup L_1L_2 \cup L_1L_1L_1 \ldots$

$= \{w_1w_2\ldots w_k : k \geq 0 \text{ and each } x_i \in L_1\}$
1.1 Finite automata

Intuitively,

- $L_1 \cup L_2$ contains strings that are from either $L_1$ or $L_2$;
- $L_1 L_2$ contains strings consisting of two segments, with the first segment belonging to $L_1$ and the second to $L_2$;
- $L_1^*$ contains strings consisting of zero or more segments, each belonging to $L_1$.
1.1 Finite automata

Intuitively,

- $L_1 \cup L_2$ contains strings that are from either $L_1$ or $L_2$;
1.1 Finite automata

Intuitively,

- \( L_1 \cup L_2 \) contains strings that are from either \( L_1 \) or \( L_2 \);

- \( L_1 L_2 \) contains strings consisting of two segments, with the first segment belonging to \( L_1 \) and the second to \( L_2 \).
1.1 Finite automata

Intuitively,

- $L_1 \cup L_2$ contains strings that are from either $L_1$ or $L_2$;

- $L_1 L_2$ contains strings consisting of two segments, with the first segment belonging to $L_1$ and the second to $L_2$;

- $L_1^*$ contains strings consisting of zero or more segments, each belonging to $L_1$. 
1.1 Finite automata

$L_1$ contains all strings with substring 11

$L_2$ contains all strings with substring 000

How to construct an FA to recognize $L_1 \cup L_2$

We may need to use FAs designed for $L_1$ and for $L_2$.

but first, let us construct such $M_1$ and $M_2$. 
1.1 Finite automata

$L_1$ contains all strings with substring 11
1.1 Finite automata

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000
1.1 Finite automata

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000

How to construct an FA to recognize $L_1 \cup L_2$
1.1 Finite automata

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000

How to construct an FA to recognize $L_1 \cup L_2$

We may need to use FAs designed for $L_1$ and for $L_2$. 
1.1 Finite automata

$L_1$ contains all strings with substring 11
$L_2$ contains all strings with substring 000

How to construct an FA to recognize $L_1 \cup L_2$

We may need to use FAs designed for $L_1$ and for $L_2$.

but first, let us construct such $M_1$ and $M_2$. 

1.1 Finite automata

However, given two FAs, one for $L_1$ and another for $L_2$, it may be difficult to simply combine them for the recognition of $L_1 \cup L_2$.

For example, simply putting the two starting states together does not work! What seems to be causing the problem?

We cannot "rewind the input tape" to read it again.
1.1 Finite automata

However, given two FAs, one for $L_1$ and another for $L_2$, it may be difficult to simply combine them for the recognition of $L_1 \cup L_2$. 

Moreover, it is important to note that

We cannot "rewind the input tape" to read it again.
1.1 Finite automata

However, given two FAs, one for $L_1$ and another for $L_2$, it may be difficult to simply combine them for the recognition of $L_1 \cup L_2$.

e.g., simply put the two starting states together does not work!
However, given two FAs, one for $L_1$ and another for $L_2$, it may be difficult to simply combine them for the recognition of $L_1 \cup L_2$.

e.g., simply put the two starting states together does not work!

What seem to be causing the problem?
1.1 Finite automata

However, given two FAs, one for $L_1$ and another for $L_2$, it may be difficult to simply combine them for the recognition of $L_1 \cup L_2$.

e.g., simply put the two starting states together does not work!

What seem to be causing the problem?

We cannot “rewind the input tape” to read it again.
1.1 Finite automata

Allow the input string to be checked simultaneously by the two FAs when it is being read.

• to combine states from the two FAs, and
• to combine transition functions from the two FAs

For example, for $\delta(q, 1) = q'$ of $M_1$ and $\gamma(p, 1) = p'$ of $M_2$, we define:

$\theta([q, p], 1) = [q', p']$
1.1 Finite automata

Allow the input string to be checked *simultaneously by the two FAs* when it is being read
1.1 Finite automata

Allow the input string to be checked *simultaneously by the two FAs* when it is being read

How?
1.1 Finite automata

Allow the input string to be checked \textit{simultaneously by the two FAs} when it is being read.

How?

- to \textit{combine states from the two FAs}, and
1.1 Finite automata

Allow the input string to be checked simultaneously by the two FAs when it is being read

How?

• to combine states from the two FAs, and
• to combine transition functions from the two FAs
1.1 Finite automata

Allow the input string to be checked \textit{simultaneously by the two FAs} when it is being read.

How?

- to combine states from the two FAs, and
- to combine transition functions from the two FAs

For example, for $\delta(q, 1) = q'$ of $M_1$ and $\gamma(p, 1) = p'$ of $M_2$,
1.1 Finite automata

Allow the input string to be checked *simultaneously by the two FAs* when it is being read

How?

- to combine states from the two FAs, and
- to combine transition functions from the two FAs

For example, for $\delta(q, 1) = q'$ of $M_1$ and $\gamma(p, 1) = p'$ of $M_2$,

we define: $\theta([q]_p, 1) = [q']_p$
1.1 Finite automata

Assuming two simpler languages

\[ L_1 = \{ x : x \text{ contains substring 1} \} \]

\[ L_2 = \{ x : x \text{ contains substring 00} \} \]

with the corresponding FAs

\[ M_1 \]

has start state \( q \) and accepting state \( q_1 \)

\[ M_2 \]

has start state \( p \), state \( p_0 \), accepting state \( p_{00} \)
1.1 Finite automata

Assuming two simpler languages

\[ L_1 = \{ x : x \text{ contains substring } 1 \} \]
\[ L_2 = \{ x : x \text{ contains substring } 00 \} \]
1.1 Finite automata

Assuming two simpler languages

\[ L_1 = \{ x : x \text{ contains substring 1} \} \]
\[ L_2 = \{ x : x \text{ contains substring 00} \} \]

with the corresponding FAs \( M_1 \) and \( M_2 \).

where
1.1 Finite automata

Assuming two simpler languages

\[ L_1 = \{ x : x \text{ contains substring 1} \} \]
\[ L_2 = \{ x : x \text{ contains substring 00} \} \]

with the corresponding FAs \( M_1 \) and \( M_2 \).

where

\( M_1 \) has start state \( q \) and accepting state \( q_1 \)


\[
\begin{array}{c|cc}
0 & 1 \\
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]
1.1 Finite automata

Assuming two simpler languages

\[ L_1 = \{ x : x \text{ contains substring 1} \} \]
\[ L_2 = \{ x : x \text{ contains substring 00} \} \]

with the corresponding FAs \( M_1 \) and \( M_2 \), where

\( M_1 \) has start state \( q \) and accepting state \( q_1 \)

\[
\begin{array}{ccc}
0 & 1 \\
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]

\( M_2 \) has start state \( p \), state \( p_0 \), accepting state \( p_{00} \)

\[
\begin{array}{ccc}
0 & 1 \\
p & p_0 & p \\
p_0 & p_{00} & p \\
p_{00} & p_{00} & p_{00} \\
\end{array}
\]
1.1 Finite automata
### 1.1 Finite automata

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>$q$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_1$</td>
</tr>
</tbody>
</table>

Then the new machine $M$ has 6 states:

- $[q \, p]$
- $[q \, p \, 0]$
- $[q \, p \, 00]$
- $[q_1 \, p]$  
- $[q_1 \, p \, 0]$  
- $[q_1 \, p \, 00]$

Which are the accepting states?
### 1.1 Finite automata

\[
M_1
\begin{array}{cccc}
  & 0 & 1 \\
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]

\[
M_2
\begin{array}{cccc}
  & 0 & 1 \\
p & p_0 & p \\
p_0 & p_00 & p \\
p_00 & p_00 & p_00 \\
\end{array}
\]

Which are the accepting states?
1.1 Finite automata

\[ M_1 \]
\[
\begin{array}{ccc}
0 & 1 \\
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]

\[ M_2 \]
\[
\begin{array}{ccc}
0 & 1 \\
p & p_0 & p \\
p_0 & p_{00} & p \\
p_{00} & p_{00} & p_{00} \\
\end{array}
\]

Then the new machine \( M \) has 6 states: \([q_{p0}], [q_{p00}], [q_{q1}], [q_{q1}], [q_{q1}], [q_{q1}]\)
1.1 Finite automata

\[
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
 & 0 & 1 \\
\hline
p & p_0 & p \\
p_0 & p_{00} & p \\
p_{00} & p_{00} & p_{00} \\
\end{array}
\]

Then the new machine \( M \) has 6 states: \( [q], [q_1], [q], [q_1], [q_1], [q_1] \)

\[
\begin{array}{c|cc}
 & 0 & 1 \\
\hline
\end{array}
\]
1.1 Finite automata

\[ M_1 \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>q</td>
<td>q</td>
</tr>
<tr>
<td>q_1</td>
<td>q</td>
<td>q</td>
</tr>
</tbody>
</table>

\[ M_2 \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>p_0</td>
<td>p</td>
</tr>
<tr>
<td>p_0</td>
<td>p_00</td>
<td>p</td>
</tr>
<tr>
<td>p_00</td>
<td>p_00</td>
<td>p_00</td>
</tr>
</tbody>
</table>

Then the new machine \( M \) has 6 states: \([q]_p, [q]_{p_0}, [q]_{p_00}, [q_1]_p, [q_1]_{p_0}, [q_1]_{p_00}\)
### 1.1 Finite automata

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>q</td>
<td>q₁</td>
</tr>
<tr>
<td>q₁</td>
<td>q₁</td>
<td>q₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$M_2$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>p₀</td>
<td>p</td>
</tr>
<tr>
<td>p₀</td>
<td>p₀₀</td>
<td>p</td>
</tr>
<tr>
<td>p₀₀</td>
<td>p₀₀</td>
<td>p₀₀</td>
</tr>
</tbody>
</table>

Then the new machine $M$ has 6 states: $[q]_{p}, [q]_{p₀}, [q]_{p₀₀}, [q₁]_{p}, [q₁]_{p₀}, [q₁]_{p₀₀}$
1.1 Finite automata

\[ M_1 \]
\[
\begin{array}{ccc}
0 & 1 \\
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]

\[ M_2 \]
\[
\begin{array}{ccc}
0 & 1 \\
p & p_0 & p \\
p_0 & p_{00} & p \\
p_{00} & p_{00} & p_{00} \\
\end{array}
\]

Then the new machine \( M \) has 6 states: \( [q] \), \( [q_0] \), \( [q] \), \( [q_1] \), \( [q_1] \), \( [q_{00}] \)
1.1 Finite automata

\[ M_1 \]
\[
\begin{array}{c|cc}
0 & 1 \\
q & q & q_1 \\
q_1 & q_1 & q_1 \\
\end{array}
\]

\[ M_2 \]
\[
\begin{array}{c|ccc}
0 & 1 \\
p & p_0 & p \\
p_0 & p_00 & p \\
p_00 & p_00 & p_00 \\
\end{array}
\]

Then the new machine \( M \) has 6 states: \([q], [q_0], [q_1], [q_1], [q_1], [q_1]\)

\[
\begin{array}{c|cc}
0 & 1 \\
\begin{bmatrix} q \\ p \end{bmatrix} & \begin{bmatrix} q_0 \\ q \end{bmatrix} & \begin{bmatrix} q_1 \\ p \end{bmatrix} \\
\begin{bmatrix} p_0 \\ q \end{bmatrix} & \begin{bmatrix} p_00 \\ q_0 \\ q \end{bmatrix} & \begin{bmatrix} p_00 \\ q_1 \\ p \end{bmatrix} \\
\begin{bmatrix} p_00 \\ q \end{bmatrix} & \begin{bmatrix} p_00 \\ q_1 \\ p \end{bmatrix} & \begin{bmatrix} p_00 \\ q_1 \\ p_0 \end{bmatrix} \\
\begin{bmatrix} q_0 \\ p_0 \end{bmatrix} & \begin{bmatrix} q_0 \\ p_00 \\ q_1 \\ p_0 \end{bmatrix} & \begin{bmatrix} q_0 \\ p_00 \\ q_1 \\ p_00 \end{bmatrix} \\
\begin{bmatrix} q_1 \\ p_0 \end{bmatrix} & \begin{bmatrix} q_1 \\ p_00 \\ q_1 \\ p_00 \end{bmatrix} & \begin{bmatrix} q_1 \\ p_00 \\ q_1 \\ p_00 \end{bmatrix} \\
\end{array}
\]

Which are the accepting states?
1.1 Finite automata
1.1 Finite automata
1.1 Finite automata
1.1 Finite automata

The technique applies to constructing an FA for $L_1 \cap L_2$. But what would the accepting states be? Seems not applicable to constructing an FA for concatenation $L_1 L_2$. Or can we just “connect” two FAs to form one? What might seem to be the problem?
1.1 Finite automata

The technique applies to constructing an FA for $L_1 \cap L_2$. But what would the accepting states be? Seems not applicable to constructing an FA for $L_1L_2$. Or can we just "connect" two FAs to form one? What might seem to be the problem?
1.1 Finite automata

The technique applies to constructing an FA for $L_1 \cap L_2$.

But what would the accepting states be?
1.1 Finite automata

The technique applies to constructing an FA for $L_1 \cap L_2$.

But what would the accepting states be?

Seems not applicable to constructing an FA for concatenation $L_1 L_2$. 
1.1 Finite automata

The technique applies to constructing an FA for $L_1 \cap L_2$.

But what would the accepting states be?

Seems not applicable to constructing an FA for concatenation $L_1 L_2$.

Or can we just “connect” two FAs to form one?
1.1 Finite automata

The technique applies to constructing an FA for $L_1 \cap L_2$.

But what would the accepting states be?

Seems not applicable to constructing an FA for concatenation $L_1 L_2$.

Or can we just “connect” two FAs to form one?

What might seem to be the problem?
1.2 Nondeterminism

The need for nondeterminism in the computation with FAs, e.g., combining two FAs for $L_1 \cup L_2$, for $L_1 \cap L_2$.

- Nondeterminism is "uncertainty" or "guessing".
- In FAs, nondeterminism is such that a transition from given (state, symbol) has more than one option, e.g., $\delta(q_0, 1) = \{q_1, q_2\}$.
- In FA, nondeterministic computation results in more than one sequences of states.
1.2 Nondeterminism

Nondeterminism

- Nondeterminism is "uncertainty" or "guessing".
- In FAs, nondeterminism is such that a transition from a given state, symbol has more than one option, e.g., \( \delta(q_0, 1) = \{ q_1, q_2 \} \).
- In FA, nondeterministic computation results in more than one sequence of states.
1.2 Nondeterminism

Nondeterminism

The need for nondeterminism in the computation with FAs e.g., combining two FAs for $L_1 \cup L_2$, for $L_1L2$. 

1.2 Nondeterminism

Nondeterminism

The need for nondeterminism in the computation with FAs
e.g., combining two FAs for $L_1 \cup L_2$, for $L_1 L_2$.

- nondeterminism is “uncertainty” or “guessing”.
1.2 Nondeterminism

Nondeterminism

The need for nondeterminism in the computation with FAs
e.g., combining two FAs for $L_1 \cup L_2$, for $L_1L_2$.

- nondeterminism is “uncertainty” or “guessing”.
- in FAs, nondeterminism is such that
  a transition from given (state, symbol) has more than one option
1.2 Nondeterminism

**Nondeterminism**

The need for nondeterminism in the computation with FAs
e.g., combining two FAs for $L_1 \cup L_2$, for $L_1L2$.

- nondeterminism is “uncertainty” or “guessing”.

- in FAs, nondeterminism is such that
  a transition from given (state, symbol) has more than one option

e.g., $\delta(q_0, 1) = \{q_1, q_2\}$
1.2 Nondeterminism

Nondeterminism

The need for nondeterminism in the computation with FAs

- Combining two FAs for $L_1 \cup L_2$, for $L_1L_2$.

  - Nondeterminism is “uncertainty” or “guessing”.

  - In FAs, nondeterminism is such that a transition from given (state, symbol) has more than one option.

    e.g., $\delta(q_0, 1) = \{q_1, q_2\}$

  - In FA, nondeterministic computation results in more than one sequences of states.
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs. E.g., construct an FA for $L_1 \cup L_2$. More examples: Figure 1.34 (page 52) and Figure 1.36 (page 53).
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs

E.g., construct an FA for $L_1 \cup L_2$
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs

E.g., construct an FA for $L_1 \cup L_2$

E.g., Construct an FA for all strings whose third position is a 1
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs

E.g., construct an FA for \( L_1 \cup L_2 \)

E.g., Construct an FA for all strings whose **third position** is a 1

compared with
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs

E.g., construct an FA for $L_1 \cup L_2$

E.g., Construct an FA for all strings whose third position is a 1

compared with

Construct an FA for all strings whose third last position is a 1
1.2 Nondeterminism

Nondeterminism increases convenience in constructing FAs

E.g., construct an FA for $L_1 \cup L_2$

E.g., Construct an FA for all strings whose third position is a 1

compared with

Construct an FA for all strings whose third last position is a 1

More examples: Figure 1.34 (page 52) and Figure 1.36 (page 53)
1.2 Nondeterminism

What are $\epsilon$-transitions? $\delta(q, \epsilon) = \{q_1\}$.

What does an $\epsilon$-transition mean?

$\epsilon$-transition makes FA construction convenient. e.g., FA for concatenation $L_1 L_2$.

More example: Example 1.38 (page 54)
1.2 Nondeterminism

$\epsilon$-transitions

What are they?

$\delta(q, \epsilon) = \{ q_1 \}$. What does an $\epsilon$-transition mean?

$\epsilon$-transition makes FA construction convenient. e.g., FA for concatenation $L_1L_2$.

More example: Example 1.38 (page 54)
1.2 Nondeterminism

$\epsilon$-transitions

What are they?
1.2 Nondeterminism

$\epsilon$-transitions

What are they?

$$\delta(q, \epsilon) = \{q_1\}.$$
1.2 Nondeterminism

$\epsilon$-transitions

What are they?

$$\delta(q, \epsilon) = \{q_1\}.$$  

What does an $\epsilon$-transition mean?
1.2 Nondeterminism

$\epsilon$-transitions

What are they?

$$\delta(q, \epsilon) = \{ q_1 \}.$$  

What does an $\epsilon$-transition mean?

$\epsilon$-transition makes FA construction convenient.

e.g., FA for concatenation $L_1L_2$.  

1.2 Nondeterminism

ε-transitions

What are they?

\[ \delta(q, \epsilon) = \{q_1\}. \]

What does an ε-transition mean?

ε-transition makes FA construction convenient.

e.g., FA for concatenation \( L_1 L_2 \).

More example: Example 1.38 (page 54)
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A nondeterministic finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where:

1. \(Q\) is a finite set called the states,
2. \(\Sigma\) is a finite set called the alphabet,
3. \(\delta: Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

where \(\Sigma \cup \{\epsilon\}\) is the alphabet extended with the empty string, and \(\mathcal{P}(Q)\) is the power set of \(Q\), also written as \(2^Q\).
Formal Definition of a nondeterministic FA
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A nondeterministic finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where
1. \(Q\) is a finite set called the states,
Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A **nondeterministic finite automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta: Q \times \Sigma \epsilon \rightarrow P(Q)\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*. 

\(\Sigma \epsilon = \Sigma \cup \{\epsilon\}\)

\(P(Q)\) is the power set of \(Q\), also written as \(2^Q\).
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A *nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the *states*,
2. $\Sigma$ is a finite set called the *alphabet*,
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.

where $\Sigma_\epsilon = \Sigma \cup \left\{ \epsilon \right\}$ and $\mathcal{P}(Q)$ is the power set of $Q$, also written as $2^Q$. 
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

**Definition 1.37** (page 35)

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where
1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta : Q \times \Sigma_e \rightarrow \mathcal{P}(Q)\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*.
1.2 Nondeterminism

Formal Definition of a nondeterministic FA

Definition 1.37 (page 35)

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the *states*,
2. \(\Sigma\) is a finite set called the *alphabet*,
3. \(\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)\) is the *transition function*,
4. \(q_0 \in Q\) is the *start state*, and
5. \(F \subseteq Q\) is the *set of accept states*.

where

\[\Sigma_\epsilon = \Sigma \cup \{\epsilon\}\]

\(\mathcal{P}(Q)\) is the power set of \(Q\), also written as \(2^Q\).
1.2 Nondeterminism

Formal definition of computation with NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton. Let $w = w_1w_2...w_n$ be a string where each symbol $w_i \in \Sigma \in \Sigma$. Then $N$ accepts $w$ if a sequence of states $r_0, r_1, ..., r_n$ exists with three conditions:

1. $r_0 = q_0$;
2. $r_{i+1} \in \delta(r_i, w_{i+1})$, for $i = 0, 1, ..., n - 1$,
3. $r_n \in F$.

We say $N$ recognizes language $L$ if $L = \{w : w \in \Sigma^* \text{ and } N \text{ accepts } w\}$. 
1.2 Nondeterminism

Formal definition of computation with NFA
1.2 Nondeterminism

Formal definition of computation with NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton.
1.2 Nondeterminism

**Formal definition of computation with NFA**

Let $N = (Q, \Sigma_\epsilon, \delta, q_0, F)$ be a nondeterministic finite automaton.
Let $w = w_1w_2 \ldots w_n$ be a string where each symbol $w_i \in \Sigma_\epsilon$. 
1.2 Nondeterminism

**Formal definition of computation with NFA**

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be a nondeterministic finite automaton. Let \( w = w_1 w_2 \ldots w_n \) be a string where each symbol \( w_i \in \Sigma \).

Then \( N \) accepts \( w \) if a sequence states \( r_0, r_1, \ldots, r_n \) exists with three conditions:

1. \( r_0 = q_0 \);
2. \( r_{i+1} \in \delta(r_i, w_{i+1}) \) for \( i = 0, 1, \ldots, n-1 \), and
3. \( r_n \in F \).

We say \( N \) recognizes language \( L \) if \( L = \{ w : w \in \Sigma^* \text{ and } N \text{ accepts } w \} \).
Formal definition of computation with NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton. Let $w = w_1w_2\ldots w_n$ be a string where each symbol $w_i \in \Sigma$. Then $N$ accepts $w$ if a sequence states $r_0, r_1, \ldots, r_n$ exists with three conditions:

1. $r_0 = q_0$;
1.2 Nondeterminism

Formal definition of computation with NFA

Let \( N = (Q, \Sigma_\epsilon, \delta, q_0, F) \) be a nondeterministic finite automaton.

Let \( w = w_1w_2 \ldots w_n \) be a string where each symbol \( w_i \in \Sigma_\epsilon \).

Then \( N \) accepts \( w \) if a sequence states \( r_0, r_1, \ldots, r_n \) exists with three conditions:

1. \( r_0 = q_0 \);
2. \( r_{i+1} \in \delta(r_i, w_{i+1}) \), for \( i = 0, 1, \ldots, n - 1 \), and
1.2 Nondeterminism

Formal definition of computation with NFA

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be a nondeterministic finite automaton. Let \( w = w_1w_2 \ldots w_n \) be a string where each symbol \( w_i \in \Sigma \).

Then \( N \) accepts \( w \) if a sequence states \( r_0, r_1, \ldots, r_n \) exists with three conditions:

1. \( r_0 = q_0 \);
2. \( r_{i+1} \in \delta(r_i, w_{i+1}) \), for \( i = 0, 1, \ldots, n - 1 \), and
3. \( r_n \in F \).
1.2 Nondeterminism

Formal definition of computation with NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton. Let $w = w_1w_2 \ldots w_n$ be a string where each symbol $w_i \in \Sigma \epsilon$.

Then $N$ accepts $w$ if a sequence states $r_0, r_1, \ldots, r_n$ exists with three conditions:

1. $r_0 = q_0$;
2. $r_{i+1} \in \delta(r_i, w_{i+1})$, for $i = 0, 1, \ldots, n - 1$, and
3. $r_n \in F$.

We say $N$ recognizes language $L$ if

$$ L = \{ w : w \in \Sigma^* \text{ and } N \text{ accepts } w \} $$
1.2 Nondeterminism

Examples of NFAs (NFA $N_1$ in example 1.38, page 54)

The formal definition of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_1$ is the start state, $F = \{q_4\}$ and $\delta$ is given as:

- $\delta(0, q_1) = \{q_1\}$
- $\delta(1, q_1) = \emptyset$
- $\delta(\epsilon, q_2) = \emptyset$
- $\delta(0, q_2) = \emptyset$
- $\delta(1, q_2) = \{q_3\}$
- $\delta(0, q_3) = \emptyset$
- $\delta(1, q_3) = \emptyset$
- $\delta(\epsilon, q_4) = \emptyset$
- $\delta(0, q_4) = \emptyset$
- $\delta(1, q_4) = \emptyset$

What language does it accept?
1.2 Nondeterminism

Examples of NFAs (NFA $N_1$ in example 1.38, page 54)
1.2 Nondeterminism

Examples of NFAs (NFA $N_1$ in example 1.38, page 54)

The formal definition of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_1$ is the start state, $F = \{q_4\}$
1.2 Nondeterminism

Examples of NFAs (NFA $N_1$ in example 1.38, page 54)

![Diagram of NFA $N_1$]

The formal definition of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_1$ is the start state, $F = \{q_4\}$

and $\delta$ is given as

$$
\begin{array}{c|ccc}
& 0 & 1 & \epsilon \\
\hline
q_1 & \{q_1\} & \{q_1, q_2\} & \emptyset \\
\end{array}
$$
1.2 Nondeterminism

Examples of NFAs (NFA $N_1$ in example 1.38, page 54)

The formal definition of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_1$ is the start state, $F = \{q_4\}$
and $\delta$ is given as

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
</tbody>
</table>

What language does it accept?
1.2 Nondeterminism

Examples of NFAs (NFA $N_1$ in example 1.38, page 54)

The formal definition of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_1$ is the start state, $F = \{q_4\}$ and $\delta$ is given as

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\emptyset$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_4}$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

What language does it accept?
1.2 Nondeterminism

Are NFAs more powerful than DFAs?

Actually, NFAs are not more powerful than DFAs.
Are NFAs more powerful than DFAs?
1.2 Nondeterminism

Are NFAs more powerful than DFAs?

Are there languages recognized by NFAs but not by DFAs?
1.2 Nondeterminism

Are NFAs more powerful than DFAs?

Are there languages recognized by NFAs but not by DFAs?

Actually, NFAs are not more powerful than DFAs.
1.2 Nondeterminism

Equivalence of NFA and DFA

• The classes of languages recognized by NFAs and by DFAs are the same! First the class of languages recognized by DFAs can be recognized by NFAs as well. This is because $\Sigma \subseteq \Sigma^*$, and $\delta(q,a) = p$ can be redefined as $\delta(q,a) = \{p\}$.

• To see the other way around is also true, we only need to show that for every NFA, there is a DFA accepting the same language.
1.2 Nondeterminism

Equivalence of NFA and DFA

The classes of languages recognized by NFAs and by DFAs are the same! First the class of languages recognized by DFAs can be recognized by NFAs as well, this is because $\Sigma \subseteq \Sigma^\epsilon$, and $\delta(q,a) = p$ can be redefined as $\delta(q,a) = \{p\}$.

To see the other way around is also true, we only need to show that for every NFA, there is a DFA accepting the same language.
1.2 Nondeterminism

Equivalence of NFA and DFA

- The classes of languages recognized by NFAs and by DFAs are the same!
1.2 Nondeterminism

Equivalence of NFA and DFA

- The classes of languages recognized by NFAs and by DFAs are the same!

First the class of languages recognized by DFAs can be recognized by NFAs as well.
1.2 Nondeterminism

Equivalence of NFA and DFA

- The classes of languages recognized by NFAs and by DFAs are the same!

First the class of languages recognized by DFAs can be recognized by NFAs as well.

this is because

\[ \Sigma \subseteq \Sigma_{\epsilon}, \quad \text{and} \]
\[ \delta(q, a) = p \quad \text{can be redefined as} \quad \delta(q, a) = \{p\}. \]
1.2 Nondeterminism

Equivalence of NFA and DFA

• The classes of languages recognized by NFAs and by DFAs are the same!

First the class of languages recognized by DFAs can be recognized by NFAs as well.

this is because

\[ \Sigma \subseteq \Sigma_\epsilon, \quad \text{and} \]

\[ \delta(q, a) = p \] can be redefined as \( \delta(q, a) = \{p\} \).

• To see the other way around is also true, we only need to show
1.2 Nondeterminism

Equivalence of NFA and DFA

- The classes of languages recognized by NFAs and by DFAs are the same!

First the class of languages recognized by DFAs can be recognized by NFAs as well.

this is because
\[ \Sigma \subseteq \Sigma_\epsilon, \text{ and} \]
\[ \delta(q, a) = p \text{ can be redefined as } \delta(q, a) = \{p\}. \]

- To see the other way around is also true, we only need to show that for every NFA, there is a DFA accepting the same language.
1.2 Nondeterminism

Theorem 1.39

Every NFA has an equivalent DFA.

Proof idea:
To use a DFA to "simulate the NFA's computation".
How:
- Keep tracking on all the transitions of the NFA.
- Assume the NFA has $k$ states in total.
- Given state $q$ and symbol $a$, transiting to a subset of $k$ states.
- Make each such subset to a single state in the new DFA.
1.2 Nondeterminism

**Theorem 1.39** (page 55) *Every NFA has an equivalent DFA.*
1.2 Nondeterminism

**Theorem 1.39** (page 55) *Every NFA has an equivalent DFA.*

**Proof idea**
To use a DFA to “simulate the NFA’s computation”
1.2 Nondeterminism

**Theorem 1.39** (page 55) *Every NFA has an equivalent DFA.*

**Proof idea**
To use a DFA to “simulate the NFA’s computation”

How: keep tracking on all the transitions of the NFA

- assume the NFA has $k$ states in total
1.2 Nondeterminism

Theorem 1.39 (page 55) Every NFA has an equivalent DFA.

Proof idea
To use a DFA to “simulate the NFA’s computation”

How: keep tracking on all the transitions of the NFA

- assume the NFA has \( k \) states in total
- given state \( q \) and symbol \( a \), transiting to a subset of \( k \) states
1.2 Nondeterminism

**Theorem 1.39** (page 55) *Every NFA has an equivalent DFA.*

**Proof idea**
To use a DFA to “simulate the NFA’s computation”

How: keep tracking on all the transitions of the NFA

- assume the NFA has \( k \) states in total
- given state \( q \) and symbol \( a \), transiting to a subset of \( k \) states
- make each such subset to single state in the new DFA,
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{ p \} \]
\[ \delta(p, 1) = \{ p, q \} \]
\[ \delta(q, 0) = \emptyset \]
\[ \delta(q, 1) = \emptyset \]

What language does it accept?

Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?

Define a DFA with states \( \emptyset, p, q, p, q \) with new transition function

\[ \Delta(p, 0) = \{ p \} \]
\[ \Delta(p, 1) = \{ p, q \} \]
\[ \Delta(q, 0) = \emptyset \]
\[ \Delta(q, 1) = \emptyset \]

\[ \Delta(p, q) = \{ p \} \]
\[ \Delta(p, q) = \{ p, q \} \]

What does it look like?
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept?
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?

Define a DFA with states \( \emptyset, \{p\}, \{q\}, \) and \( \{p, q\} \) with new transition function \( \Delta \).
1.2 Nondeterminism

E.g., Consider NFA

![NFA Diagram](image)

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \(\delta(q, 0) = p\) needed?

Can you design a DFA for this language?

Define a DFA with states \([\emptyset]\), \([p]\), \([q]\), and \([p, q]\) with new transition function \(\Delta\)

\[ \Delta([p], 0) = [p], \]

\[ \Delta([p], 1) = [p, q], \]

\[ \Delta([q], 0) = [\emptyset], \]

\[ \Delta([q], 1) = [\emptyset], \]

\[ \Delta([p, q], 0) = [p], \]

\[ \Delta([p, q], 1) = [p, q]. \]
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?

Define a DFA with states \([\emptyset]\), \([p]\), \([q]\), and \([p, q]\) with new transition function \( \Delta \)

\[ \Delta([p], 0) = [p], \Delta([p], 1) = [p, q], \]

What does it look like?
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?

Define a DFA with states \([\emptyset]\), \([p]\), \([q]\), and \([p, q]\) with new transition function \( \Delta \)

\[ \Delta([p], 0) = [p], \Delta([p], 1) = [p, q], \Delta([q], 0) = [\emptyset], \]
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?

Define a DFA with states \([\emptyset]\), \([p]\), \([q]\), and \([p, q]\) with new transition function \( \Delta \)

\[ \Delta([p], 0) = [p], \Delta([p], 1) = [p, q], \Delta([q], 0) = [\emptyset], \Delta([q], 1) = [\emptyset] \]
1.2 Nondeterminism

E.g., Consider NFA

\[ \delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset. \]

What language does it accept? Is transition \( \delta(q, 0) = p \) needed?

Can you design a DFA for this language?

Define a DFA with states \( \emptyset, [p], [q], \) and \( [p, q] \) with new transition function \( \Delta \)

\[
\begin{align*}
\Delta([p], 0) &= [p], \\
\Delta([p], 1) &= [p, q], \\
\Delta([q], 0) &= \emptyset, \\
\Delta([q], 1) &= \emptyset, \\
\Delta([p, q], 0) &= [p], \\
\end{align*}
\]
1.2 Nondeterminism

E.g., Consider NFA

δ(p, 0) = {p}, δ(p, 1) = {p, q}, δ(q, 0) = ∅, δ(q, 1) = ∅.

What language does it accept? Is transition δ(q, 0) = p needed?

Can you design a DFA for this language?

Define a DFA with states [∅], [p], [q], and [p, q] with new transition function ∆

Δ([p], 0) = [p], Δ([p], 1) = [p, q], Δ([q], 0) = [∅], Δ([q], 1) = [∅]
Δ([p, q], 0) = [p], Δ([p, q], 1) = [p, q]
1.2 Nondeterminism

E.g., Consider NFA

\[
\delta(p, 0) = \{p\}, \delta(p, 1) = \{p, q\}, \delta(q, 0) = \emptyset, \delta(q, 1) = \emptyset.
\]

What language does it accept? Is transition \(\delta(q, 0) = p\) needed?

Can you design a DFA for this language?

Define a DFA with states \([\emptyset]\), \([p]\), \([q]\), and \([p, q]\) with new transition function \(\Delta\)

\[
\begin{align*}
\Delta([p], 0) &= [p], & \Delta([p], 1) &= [p, q], & \Delta([q], 0) &= \emptyset, & \Delta([q], 1) &= \emptyset \\
\Delta([p, q], 0) &= [p], & \Delta([p, q], 1) &= [p, q]
\end{align*}
\]

What does it look like?
1.2 Nondeterminism

NFA to DFA conversion further explained: consider again NFA on input string 011:

\[ q_1 \]

\[ 0 = \Rightarrow q_1 \]

\[ 1 = \Rightarrow q_1, q_2 \]

\[ q_1 1 = \Rightarrow q_1, q_2 \]

\[ q_2 \epsilon = \Rightarrow q_3 \]

\[ 1 = \Rightarrow q_1, q_2, q_3 \]

We can represent these branchings on string 011 deterministically:
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA on input string 011
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA
on input string 011

\[ \{q_1\} \xrightarrow{0} \]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA on input string 011

\( \{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \)
1.2 Nondeterminism

NFA to DFA conversion further explained:

Consider again NFA on input string 011:

\[ \{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \]
1.2 Nondeterminism

NFA to DFA conversion further explained:

Consider again NFA on input string 011

\[
\begin{align*}
\{q_1\} & \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 & \Rightarrow q_1
\end{align*}
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

Consider again NFA on input string 011

\[
\{q_1\} \Rightarrow \{q_1\} \Rightarrow \{q_1, q_2\}
\]

\[
q_1 \Rightarrow \{q_1, q_2\}
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

Consider again NFA on input string 011

\[ \{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \]

\[ q_1 \xrightarrow{1} \{q_1, q_2\} \]

\[ q_2 \]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA on input string 011

\[
\begin{align*}
\{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 \xrightarrow{1} \{q_1, q_2\} \\
q_2 \xrightarrow{1} \emptyset
\end{align*}
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA
on input string 011

\[
\begin{align*}
\{q_1\} & \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 & \xrightarrow{1} \{q_1, q_2\} \\
q_2 & \xrightarrow{1} \emptyset
\end{align*}
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

Consider again NFA on input string 011

\[
\begin{align*}
\{ q_1 \} & \xrightarrow{0} \{ q_1 \} \xrightarrow{1} \{ q_1, q_2 \} \\
q_1 & \xrightarrow{1} \{ q_1, q_2 \} \\
q_2 & \xrightarrow{1} \varnothing \\
q_2 & \xrightarrow{\varepsilon} \{ q_3 \}
\end{align*}
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

Consider again NFA on input string 011

\[
\begin{align*}
\{q_1\} & \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 & \xrightarrow{1} \{q_1, q_2\} \\
q_2 & \xrightarrow{\varepsilon} \emptyset \\
q_2 & \xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}
\]

We can represent these branchings on string 011 deterministically:
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA on input string 011

\[
\begin{align*}
\{q_1\} & \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 & \xrightarrow{1} \{q_1, q_2\} \\
q_2 & \xrightarrow{\varepsilon} \emptyset \\
q_2 & \xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}
\]

We can represent these branchings on string 011 deterministically:
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA on input string 011

\[ \{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \{ q_1, q_2 \} \]

\[ q_1 \xrightarrow{1} \{ q_1, q_2 \} \]

\[ q_2 \xrightarrow{1} \emptyset \]

\[ q_2 \xrightarrow{\varepsilon} \{ q_3 \} \xrightarrow{1} \{ q_4 \} \]

We can represent these branchings on string 011 deterministically:

\[ [q_1] \xrightarrow{0} \]
1.2 Nondeterminism

**NFA to DFA conversion further explained:**

Consider again NFA on input string 011

\[
\begin{align*}
\{q_1\} &\xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 &\xrightarrow{1} \{q_1, q_2\} \\
q_2 &\xrightarrow{\varepsilon} \emptyset \\
q_2 &\xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}
\]

We can represent these branchings on string 011 **deterministically**:

\[
[q_1] \xrightarrow{0} [q_1]
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA on input string 011

\[
\begin{align*}
\{q_1\} &\xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 &\xrightarrow{1} \{q_1, q_2\} \\
q_2 &\xrightarrow{1} \emptyset \\
q_2 &\xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}
\]

We can represent these branchings on string 011 deterministically:

\[
[q_1] \xrightarrow{0} [q_1] \xrightarrow{1} [q_1, q_2]
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA
on input string 011

\[
\begin{align*}
\{q_1\} & \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 & \xrightarrow{1} \{q_1, q_2\} \\
q_2 & \xrightarrow{1} \emptyset \\
q_2 & \xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}
\]

We can represent these branchings on string 011 deterministically:

\[
[q_1] \xrightarrow{0} [q_1] \xrightarrow{1} [q_1, q_2]
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA
on input string 011

\[
\begin{align*}
\{q_1\} & \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \\
q_1 & \xrightarrow{1} \{q_1, q_2\} \\
q_2 & \xrightarrow{\varepsilon} \emptyset \\
q_2 & \xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}
\]

We can represent these branchings on string 011 deterministically:

\[
[q_1] \xrightarrow{0} [q_1] \xrightarrow{1} [q_1, q_2] \xrightarrow{\text{with } E(q_1, q_2)} [q_1, q_2, q_3]
\]
1.2 Nondeterminism

NFA to DFA conversion further explained:

consider again NFA
on input string 011

\[ \{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\} \]
\[ q_1 \xrightarrow{1} \{q_1, q_2\} \]
\[ q_2 \xrightarrow{1} \emptyset \]
\[ q_2 \xrightarrow{\epsilon} \{q_3\} \xrightarrow{1} \{q_4\} \]

We can represent these branchings on string 011 deterministically:

\[ [q_1] \xrightarrow{0} [q_1] \xrightarrow{1} [q_1, q_2] \xrightarrow{\text{with } E(q_1, q_2)} [q_1, q_2, q_3] \xrightarrow{1} [q_1, q_2, q_4] \]
NFA to DFA conversion further explained:

consider again NFA
on input string 011

\{q_1\} \xrightarrow{0} \{q_1\} \xrightarrow{1} \{q_1, q_2\}

\begin{align*}
q_1 & \xrightarrow{1} \{q_1, q_2\} \\
q_2 & \xrightarrow{1} \emptyset \\
q_2 & \xrightarrow{\varepsilon} \{q_3\} \xrightarrow{1} \{q_4\}
\end{align*}

We can represent these branchings on string 011 deterministically:

\begin{align*}
[q_1] & \xrightarrow{0} [q_1] \xrightarrow{1} [q_1, q_2] \xrightarrow{\text{with } E(q_1, q_2)} [q_1, q_2, q_3] \xrightarrow{1} [q_1, q_2, q_4]
\end{align*}

using \( E(\{q_1, q_2\}) = \{q_1, q_2, q_3\} \), states reachable from \( \{q_1, q_2\} \) via \( \varepsilon \)-transitions
1.2 Nondeterminism

But how exactly is the new transition \( \Delta \) defined?

- \( Q' = P(Q) \), consider all \( 2^k \) states – if the NFA has \( k \) states in \( Q \);
- let \( R \in Q' \) (i.e., \( R \subseteq Q \)), \( \Delta(R, a) = \bigcup_{r \in R} \delta(r, a) = \{q: q \in \delta(r, a), \text{ for some } r \in R\} \)

\( e.g., \Delta(\{p,q\}, 0) = \delta(p, 0) \cup \delta(q, 0) = \{p\} \cup \emptyset = \{p\} \)

\( \Delta(\{p,q\}, 1) = \delta(p, 1) \cup \delta(q, 1) = \{p,q\} \cup \emptyset = \{p,q\} \)

that is: \( \Delta([p,q], 0) = [p], \Delta([p,q], 1) = [p,q] \)
1.2 Nondeterminism

But how exactly is the new transition $\Delta$ defined?
1.2 Nondeterminism

But how exactly is the new transition $\Delta$ defined?

- $Q' = \mathcal{P}(Q)$, consider all $2^k$ states – if the NFA has $k$ states in $Q$;
1.2 Nondeterminism

But how exactly is the new transition $\Delta$ defined?

- $Q' = \mathcal{P}(Q)$, consider all $2^k$ states – if the NFA has $k$ states in $Q$;
- let $R \in Q'$ (i.e., $R \subseteq Q$),
1.2 Nondeterminism

But how exactly is the new transition \( \Delta \) defined?

- \( Q' = \mathcal{P}(Q) \), consider all \( 2^k \) states – if the NFA has \( k \) states in \( Q \);
- let \( R \in Q' \) (i.e., \( R \subseteq Q \)),
  
  \[
  \Delta(R, a) = \bigcup_{r \in R} \delta(r, a) = \{ q : q \in \delta(r, a), \text{ for some } r \in R \}
  \]
But how exactly is the new transition $\Delta$ defined?

- $Q' = \mathcal{P}(Q)$, consider all $2^k$ states – if the NFA has $k$ states in $Q$;
- let $R \in Q'$ (i.e., $R \subseteq Q$),

$$\Delta(R, a) = \bigcup_{r \in R} \delta(r, a) = \{q : q \in \delta(r, a), \text{ for some } r \in R\}$$

e.g., $\Delta(\{p, q\}, 0) = \delta(p, 0) \cup \delta(q, 0) = \{p\} \cup \emptyset = \{p\}$
$\Delta(\{p, q\}, 1) = \delta(p, 1) \cup \delta(q, 1) = \{p, q\} \cup \emptyset = \{p, q\}$
1.2 Nondeterminism

But how exactly is the new transition $\Delta$ defined?

- $Q' = \mathcal{P}(Q)$, consider all $2^k$ states – if the NFA has $k$ states in $Q$;

- let $R \in Q'$ (i.e., $R \subseteq Q$),

\[
\Delta(R, a) = \bigcup_{r \in R} \delta(r, a) = \{q : q \in \delta(r, a), \text{ for some } r \in R\}
\]

e.g., $\Delta(\{p, q\}, 0) = \delta(p, 0) \cup \delta(q, 0) = \{p\} \cup \emptyset = \{p\}$

$\Delta(\{p, q\}, 1) = \delta(p, 1) \cup \delta(q, 1) = \{p, q\} \cup \emptyset = \{p, q\}$

that is: $\Delta([p, q], 0) = [p]$,
1.2 Nondeterminism

But how exactly is the new transition $\Delta$ defined?

- $Q' = \mathcal{P}(Q)$, consider all $2^k$ states – if the NFA has $k$ states in $Q$;
- let $R \in Q'$ (i.e., $R \subseteq Q$),

$$
\Delta(R, a) = \bigcup_{r \in R} \delta(r, a) = \{ q : q \in \delta(r, a), \text{ for some } r \in R \}
$$

E.g.,

- $\Delta(\{p, q\}, 0) = \delta(p, 0) \cup \delta(q, 0) = \{p\} \cup \emptyset = \{p\}$
- $\Delta(\{p, q\}, 1) = \delta(p, 1) \cup \delta(q, 1) = \{p, q\} \cup \emptyset = \{p, q\}$

That is:
- $\Delta([p, q], 0) = [p]$, $\Delta([p, q], 1) = [p, q]$
1.2 Nondeterminism

Now include $\epsilon$-transitions. But again what does an $\epsilon$-transition mean?, e.g., $\delta(p, \epsilon) = \{q\}$ if a state can reach state $p$ on some string, it can also reach $q$ on the same string. This implies that the set of states that can be transited to should be extended through $\epsilon$-transitions:

$$E(R) = \{q : q \text{ is reachable from } R \text{ via zero or more } \epsilon-\text{transitions}\}$$

New transition:

$$\Delta(R, a) = \bigcup_{r \in R} E(\delta(r, a)) = \{q : q \in E(\delta(r, a)) \text{ for some } r \in R\}$$
1.2 Nondeterminism

Now include $\epsilon$-transitions

Now include $\epsilon$-transitions
Now include $\epsilon$-transitions

But again what does an $\epsilon$-transition mean?, e.g., $\delta(p, \epsilon) = \{q\}$
1.2 Nondeterminism

Now include $\epsilon$-transitions

But again what does an $\epsilon$-transition mean?, e.g., $\delta(p, \epsilon) = \{q\}$

if a state can reach state $p$ on some string, it can also reach $q$ on the same string.
1.2 Nondeterminism

Now include \( \epsilon \)-transitions

But again what does an \( \epsilon \)-transition mean?, e.g., \( \delta(p, \epsilon) = \{q\} \)

if a state can reach state \( p \) on some string, it can also reach \( q \) on the same string.

This implies that the set of states that can be transited to should be extended through \( \epsilon \)-transitions:
1.2 Nondeterminism

Now include $\epsilon$-transitions

But again what does an $\epsilon$-transition mean?, e.g., $\delta(p, \epsilon) = \{q\}$

if a state can reach state $p$ on some string, it can also reach $q$ on the same string.

This implies that the set of states that can be transited to should be extended through $\epsilon$-transitions:

$E(R) = \{q : q$ is reachable from $R$ via zero or more $\epsilon$-transitions$\}$
1.2 Nondeterminism

Now include $\epsilon$-transitions

But again what does an $\epsilon$-transition mean?, e.g., $\delta(p, \epsilon) = \{q\}$

if a state can reach state $p$ on some string, it can also reach $q$ on the same string.

This implies that the set of states that can be transited to should be extended through $\epsilon$-transitions:

$$E(R) = \{q : q \text{ is reachable from } R \text{ via zero or more } \epsilon-\text{transitions}\}$$

New transition:

$$\Delta(R, a) = \bigcup_{r \in R} E(\delta(r, a)) = \{q : q \in E(\delta(r, a)) \text{ for some } r \in R\}$$
1.2 Nondeterminism

Proof of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \( |Q| = k \)

We construct a DFA \( M = (Q', \Sigma, \Delta, q'_{0}, F') \) to recognize the same language \( L \).

1. \( Q' \) contains \( 2^k \) states, one for each subset of \( Q \).
2. Let \( p_R \in Q' \), where \( R \subseteq Q \) representing a subset of \( Q \) define \( \Delta(p_R, a) = p_S \), where \( S = \{ q : q \in \delta(r, a), \text{for some } r \in R \} \).
3. \( q'_{0} = p_{\{q_0\}} \).
4. \( F' = \{ p_R : R \text{ contains an accepting state of } N \} \).
1.2 Nondeterminism

Proof of Theorem 1.39
1.2 Nondeterminism

Proof of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \( |Q| = k \).
Proof of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \( |Q| = k \)

We construct a DFA \( M = (Q', \Sigma, \Delta, q'_0, F') \) to recognize the same language \( L \).
1.2 Nondeterminism

Proof of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \(|Q| = k\)
We construct a DFA \( M = (Q', \Sigma, \Delta, q'_0, F') \) to recognize the same language \( L \).

1. \( Q' \) contains \( 2^k \) states, one for each subset of \( Q \)
1.2 Nondeterminism

Proof of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \(|Q| = k\)

We construct a DFA \( M = (Q', \Sigma, \Delta, q'_0, F') \) to recognize the same language \( L \).

1. \( Q' \) contains \( 2^k \) states, one for each subset of \( Q \)
2. Let \( p_R \in Q' \), where \( R \subseteq Q \) representing a subset of \( Q \)
1.2 Nondeterminism

Proof of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \(|Q| = k\).

We construct a DFA \( M = (Q', \Sigma, \Delta, q'_0, F') \) to recognize the same language \( L \).

1. \( Q' \) contains \( 2^k \) states, one for each subset of \( Q \).
2. Let \( p_R \in Q' \), where \( R \subseteq Q \) representing a subset of \( Q \). Define \( \Delta(p_R, a) = p_S \), where
1.2 Nondeterminism

**Proof** of Theorem 1.39

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing language \( L \), where \( |Q| = k \).

We construct a DFA \( M = (Q', \Sigma, \Delta, q_0', F') \) to recognize the same language \( L \).

1. \( Q' \) contains \( 2^k \) states, one for each subset of \( Q \).
2. Let \( p_R \in Q' \), where \( R \subseteq Q \) representing a subset of \( Q \).
   
   Define \( \Delta(p_R, a) = p_S \), where
   
   \[ S = \{ q : q \in \delta(r, a), \text{ for some } r \in R \} \]
1.2 Nondeterminism

**Proof** of Theorem 1.39

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing language $L$, where $|Q| = k$

We construct a DFA $M = (Q', \Sigma, \Delta, q'_0, F')$ to recognize the same language $L$.

1. $Q'$ contains $2^k$ states, one for each subset of $Q$
2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
   $$S = \{ q : q \in \delta(r, a), \text{ for some } r \in R \}$$
3. $q'_0 = p\{q_0\}$
1.2 Nondeterminism

Proof of Theorem 1.39

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing language $L$, where $|Q| = k$

We construct a DFA $M = (Q', \Sigma, \Delta, q'_0, F')$ to recognize the same language $L$.

1. $Q'$ contains $2^k$ states, one for each subset of $Q$
2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
   $$S = \{ q : q \in \delta(r, a), \text{ for some } r \in R \}$$
3. $q'_0 = p_{\{q_0\}}$
4. $F' = \{ p_R : R \text{ contains an accepting state of } N \}$
1.2 Nondeterminism

Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$. Define $\Delta(p_R, a) = p_S$, where $S = \{q : q \in E(\delta(r, a)) \text{ for some } r \in R\}$.

$q_0' = p_{E(S)}$ where $E(S)$ includes all the states in $S$ and those reachable through the $\epsilon$ transitions from the states in $S$.

Apparently $M$ simulates the computation of $N$ on any input string and accepts it if and only if $N$ accepts it.
1.2 Nondeterminism

When \( \epsilon \) transitions are involved
1.2 Nondeterminism

When $\epsilon$ transitions are involved

2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$.
1.2 Nondeterminism

When $\epsilon$ transitions are involved

2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
1.2 Nondeterminism

When $\epsilon$ transitions are involved

2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
   $S = \{q : q \in E(\delta(r,a)), \text{ for some } r \in R\}$
1.2 Nondeterminism

When $\epsilon$ transitions are involved

2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
   $$S = \{q : q \in E(\delta(r, a)), \text{ for some } r \in R\}$$
3. $q'_0 = p_{E(\{q_0\})}$
1.2 Nondeterminism

When $\epsilon$ transitions are involved

2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
   \[ S = \{ q : q \in E(\delta(r, a)), \text{ for some } r \in R \} \]
3. $q_0' = p_{E(\{q_0\})}$

where $E(S)$ includes all the states in $S$ and those reachable through the $\epsilon$ transitions from the states in $S$. 
When $\epsilon$ transitions are involved

2. Let $p_R \in Q'$, where $R \subseteq Q$ representing a subset of $Q$
   define $\Delta(p_R, a) = p_S$, where
   
   $S = \{q : q \in E(\delta(r, a)), \text{ for some } r \in R\}$

3. $q'_0 = p_{E(\{q_0\})}$

where $E(S)$ includes all the states in $S$ and those reachable through the $\epsilon$
transitions from the states in $S$.

Apparently $M$ simulates the computation of $N$ on any input string and accepts
it if and only if $N$ accepts it.
1.2 Nondeterminism

• the DFA has states $\emptyset$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1, 2, 3\}$

• and the new transition function $\Delta$:
  $\Delta(\{1\}, a) = \emptyset$, $\Delta(\{1\}, b) = \{2\}$
  $\Delta(\{2\}, a) = \{2, 3\}$, $\Delta(\{2\}, b) = \{3\}$
  $\Delta(\{3\}, a) = E(\{1\}) = \{1, 3\}$, $\Delta(\{3\}, b) = \emptyset$
  $\Delta(\{1, 2\}, a) = \{2, 3\}$
  $\Delta(\{1, 2\}, b) = \{2, 3\}$
  $\Delta(\{1, 3\}, a) = E(\{1\}) = \{1, 3\}$
  $\Delta(\{1, 3\}, b) = \{2\}$
  $\Delta(\{2, 3\}, a) = E(\{1, 2, 3\}) = \{1, 2, 3\}$
  $\Delta(\{2, 3\}, b) = \{3\}$
  $\Delta(\{1, 2, 3\}, a) = E(\{1, 2, 3\}) = \{1, 2, 3\}$
  $\Delta(\{1, 2, 3\}, b) = \{2\}$
1.2 Nondeterminism

Example

```
1 2 3

\[ \alpha(\{1\}, a) = \emptyset, \alpha(\{1\}, b) = \{2\}, \alpha(\{2\}, a) = \{2, 3\}, \alpha(\{2\}, b) = \{3\}, \alpha(\{1, 2\}, a) = \{2, 3\}, \alpha(\{1, 2\}, b) = \{2, 3\}, \alpha(\{1, 3\}, a) = \{1, 3\}, \alpha(\{1, 3\}, b) = \emptyset, \alpha(\{2, 3\}, a) = \{1, 2, 3\}, \alpha(\{2, 3\}, b) = \{3\}, \alpha(\{1, 2, 3\}, a) = \{1, 2, 3\}, \alpha(\{1, 2, 3\}, b) = \{2, 3\} \]
```
1.2 Nondeterminism

Example

[Diagram of a nondeterministic finite automaton]
1.2 Nondeterminism

Example

- the DFA has states
1.2 Nondeterminism

Example

- the DFA has states $\emptyset$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1, 2, 3\}$
1.2 Nondeterminism

Example

- the DFA has states $\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$

- and the new transition function $\Delta$: 
1.2 Nondeterminism

Example

- the DFA has states \( \emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \)
- and the new transition function \( \Delta \):
  \[
  \Delta(\{1\}, a) = \emptyset, \quad \Delta(\{1\}, b) = \{2\}
  \]
1.2 Nondeterminism

Example

- the DFA has states $\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$

- and the new transition function $\Delta$:

  $\Delta(\{1\}, a) = \emptyset, \quad \Delta(\{1\}, b) = \{2\}$
  $\Delta(\{2\}, a) = \{2, 3\}, \quad \Delta(\{2\}, b) = \{3\}$
1.2 Nondeterminism

Example

- the DFA has states \( \emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \)

- and the new transition function \( \Delta \):

  \[
  \Delta(\{1\}, a) = \emptyset, \quad \Delta(\{1\}, b) = \{2\} \\
  \Delta(\{2\}, a) = \{2, 3\}, \quad \Delta(\{2\}, b) = \{3\} \\
  \Delta(\{3\}, a) = E(\{1\}) = \{1, 3\}, \quad \Delta(\{3\}, b) = \emptyset
  \]
1.2 Nondeterminism

Example

- the DFA has states \[ \emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \]

- and the new transition function \( \Delta \):

\[
\begin{align*}
\Delta(\{1\}, a) &= \emptyset, \quad \Delta(\{1\}, b) = \{2\} \\
\Delta(\{2\}, a) &= \{2, 3\}, \quad \Delta(\{2\}, b) = \{3\} \\
\Delta(\{3\}, a) &= E(\{1\}) = \{1, 3\}, \quad \Delta(\{3\}, b) = \emptyset \\
\Delta(\{1, 2\}, a) &= \{2, 3\}, \quad \Delta(\{1, 2\}, b) = \{2, 3\}
\end{align*}
\]
1.2 Nondeterminism

Example

- the DFA has states \( \emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \)

- and the new transition function \( \Delta \):

\[
\begin{align*}
\Delta(\{1\}, a) &= \emptyset, \quad \Delta(\{1\}, b) = \{2\} \\
\Delta(\{2\}, a) &= \{2, 3\}, \quad \Delta(\{2\}, b) = \{3\} \\
\Delta(\{3\}, a) &= E(\{1\}) = \{1, 3\}, \quad \Delta(\{3\}, b) = \emptyset \\
\Delta(\{1, 2\}, a) &= \{2, 3\}, \quad \Delta(\{1, 2\}, b) = \{2, 3\} \\
\Delta(\{1, 3\}, a) &= E(\{1\}) = \{1, 3\}, \quad \Delta(\{1, 3\}, b) = \{2\}
\end{align*}
\]
1.2 Nondeterminism

Example

- the DFA has states $\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$

- and the new transition function $\Delta$:

  $\Delta(\{1\}, a) = \emptyset, \hspace{1em} \Delta(\{1\}, b) = \{2\}$
  $\Delta(\{2\}, a) = \{2, 3\}, \hspace{1em} \Delta(\{2\}, b) = \{3\}$
  $\Delta(\{3\}, a) = \mathcal{E}(\{1\}) = \{1, 3\}, \hspace{1em} \Delta(\{3\}, b) = \emptyset$
  $\Delta(\{1, 2\}, a) = \{2, 3\}, \hspace{1em} \Delta(\{1, 2\}, b) = \{2, 3\}$
  $\Delta(\{1, 3\}, a) = \mathcal{E}(\{1\}) = \{1, 3\}, \hspace{1em} \Delta(\{1, 3\}, b) = \{2\}$
  $\Delta(\{2, 3\}, a) = \mathcal{E}(\{1, 2, 3\}) = \{1, 2, 3\}, \hspace{1em} \Delta(\{2, 3\}, b) = \{3\}$
1.2 Nondeterminism

Example

- the DFA has states \( \emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \)
- and the new transition function \( \Delta \):

\[
\begin{align*}
\Delta(\{1\}, a) &= \emptyset, & \Delta(\{1\}, b) &= \{2\} \\
\Delta(\{2\}, a) &= \{2, 3\}, & \Delta(\{2\}, b) &= \{3\} \\
\Delta(\{3\}, a) &= E(\{1\}) = \{1, 3\}, & \Delta(\{3\}, b) &= \emptyset \\
\Delta(\{1, 2\}, a) &= \{2, 3\}, & \Delta(\{1, 2\}, b) &= \{2, 3\} \\
\Delta(\{1, 3\}, a) &= E(\{1\}) = \{1, 3\}, & \Delta(\{1, 3\}, b) &= \{2\} \\
\Delta(\{2, 3\}, a) &= E(\{1, 2, 3\}) = \{1, 2, 3\}, & \Delta(\{2, 3\}, b) &= \{3\} \\
\Delta(\{1, 2, 3\}, a) &= E(\{1, 2, 3\}) = \{1, 2, 3\}, & \Delta(\{1, 2, 3\}, b) &= \{2, 3\}
\end{align*}
\]
1.2 Nondeterminism

Steps to convert an NFA to DFA:

• determine the states for the DFA
• determine the start and accepting states
• determine the transition function, pay attention to $\epsilon$-transitions
• simplify the DFA by removing unnecessary states

Corollary 1.40 (page 56)

A language is regular if and only if some NFA recognizes it.
We have established the equivalence between regular languages and NFAs.
1.2 Nondeterminism

Steps to convert an NFA to DFA:
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
- determine the start and accepting states
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
- determine the start and accepting states
- determine the transition function, pay attention to $\epsilon$-transitions
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
- determine the start and accepting states
- determine the transition function, pay attention to $\epsilon$-transitions
- simplify the DFA by removing unnecessary states
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
- determine the start and accepting states
- determine the transition function, pay attention to $\epsilon$-transitions
- simplify the DFA by removing unnecessary states

Corollary 1.40 (page 56)
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
- determine the start and accepting states
- determine the transition function, pay attention to $\epsilon$-transitions
- simplify the DFA by removing unnecessary states

**Corollary 1.40** (page 56)

A language is regular if and only if some NFA recognizes it.
1.2 Nondeterminism

Steps to convert an NFA to DFA:

- determine the states for the DFA
- determine the start and accepting states
- determine the transition function, pay attention to $\epsilon$-transitions
- simplify the DFA by removing unnecessary states

**Corollary 1.40** (page 56)

A language is regular if and only if some NFA recognizes it.

We have established the equivalence between regular languages and NFAs.
1.2 Nondeterminism

Theorem 1.45
The class of regular languages is closed under the union operation.
That is, if \( L_1 \) and \( L_2 \) are regular, so is \( L_1 \cup L_2 \).

Theorem 1.46
The class of regular languages is closed under the concatenation operation.
That is, if \( L_1 \) and \( L_2 \) are regular, so is \( L_1 L_2 \).

Theorem 1.49
The class of regular languages is closed under the star operation.
That is, if \( L \) is regular, so is \( L^* \).

Note: \( L^* = \{ \epsilon \} \cup L \cup LL \cup \ldots \)
1.2 Nondeterminism

Properties of regular languages under some operations
1.2 Nondeterminism

Properties of regular languages under some operations

**Theorem 1.45** (page 59) *The class of regular languages is closed under the union operation.*
1.2 Nondeterminism

Properties of regular languages under some operations

**Theorem 1.45** (page 59) *The class of regular languages is closed under the union operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1 \cup L_2$. 

**Note:** $L^* = \{\epsilon\} \cup L \cup LL \cup ...$
1.2 Nondeterminism

Properties of regular languages under some operations

**Theorem 1.45** (page 59) *The class of regular languages is closed under the union operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1 \cup L_2$.

**Theorem 1.46** (page 60) *The class of regular languages is closed under the concatenation operation.*
Properties of regular languages under some operations

**Theorem 1.45** (page 59) *The class of regular languages is closed under the union operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1 \cup L_2$.

**Theorem 1.46** (page 60) *The class of regular languages is closed under the concatenation operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1L_2$. 
1.2 Nondeterminism

Properties of regular languages under some operations

**Theorem 1.45** (page 59) *The class of regular languages is closed under the union operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1 \cup L_2$.

**Theorem 1.46** (page 60) *The class of regular languages is closed under the concatenation operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1L_2$.

**Theorem 1.49** (page 62) *The class of regular languages is closed under the star operation.*
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59) *The class of regular languages is closed under the union operation.*

That is, if \( L_1 \) and \( L_2 \) are regular, so is \( L_1 \cup L_2 \).

Theorem 1.46 (page 60) *The class of regular languages is closed under the concatenation operation.*

That is, if \( L_1 \) and \( L_2 \) are regular, so is \( L_1 L_2 \).

Theorem 1.49 (page 62) *The class of regular languages is closed under the star operation.*

That is, if \( L \) is regular, so is \( L^* \).
Properties of regular languages under some operations

**Theorem 1.45** (page 59) *The class of regular languages is closed under the union operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1 \cup L_2$.

**Theorem 1.46** (page 60) *The class of regular languages is closed under the concatenation operation.*

That is, if $L_1$ and $L_2$ are regular, so is $L_1L_2$.

**Theorem 1.49** (page 62) *The class of regular languages is closed under the star operation.*

That is, if $L$ is regular, so is $L^*$.

Note: $L^* = \{ \epsilon \} \cup L \cup LL \cup \ldots \}$
1.2 Nondeterminism

Theorem 1.45 (page 59)
The class of regular languages is closed under the union operation.

Proof:
construct an NFA to recognize $L_1 \cup L_2$.

So how to prove?
1.2 Nondeterminism

Properties of regular languages under some operations
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)

*The class of regular languages is closed under the union operation.*
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)

*The class of regular languages is closed under the union operation.*

(What does it mean that a class is closed under certain operation?)
Properties of regular languages under some operations

Theorem 1.45 (page 59)

The class of regular languages is closed under the union operation.

(What does it mean that a class is closed under certain operation?)

Proof:
construct an NFA to recognize $L_1 \cup L_2$. 
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)

*The class of regular languages is closed under the union operation.*

(What does it mean that a class is closed under certain operation?)

Proof:
construct an NFA to recognize $L_1 \cup L_2$. 
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)

The class of regular languages is closed under the union operation.

(What does it mean that a class is closed under certain operation?)

Proof:
construct an NFA to recognize $L_1 \cup L_2$. 

[Diagram of NFAs and their union]
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)

The class of regular languages is closed under the union operation.

(What does it mean that a class is closed under certain operation?)

Proof:
construct an NFA to recognize $L_1 \cup L_2$. 

but drawing pictures is not a formal proof
1.2 Nondeterminism

Properties of regular languages under some operations

Theorem 1.45 (page 59)

The class of regular languages is closed under the union operation.

(What does it mean that a class is closed under certain operation?)

Proof:

construct an NFA to recognize $L_1 \cup L_2$.

but drawing pictures is not a formal proof

So how to prove?
1.2 Nondeterminism

Let $w = w_1 w_2 ... w_n \in L_1 \cup L_2$, where $w_i \in \Sigma$. Then $w \in L_1$ or $w \in L_2$.

Assume $w \in L_1$ (the case of $w \in L_2$ can be proved similarly.)

Assume $N_1$ recognizes $L_1$; then $w$ is accepted by $N_1$.

There must be a sequence of states $r_0 r_1 ... r_n$, such that

- $r_0$ is the start state of $N_1$,
- $r_{i+1} \in \delta_1(r_i, w_{i+1})$, $i = 0, 1, ..., n-1$
- $r_n$ is an accept state of $N_1$.

On the combined NFA $N$, $w$ can be written as $\epsilon w_1 ... w_n$.

According to the construction of $N$, it is clear the sequence of states of $N_0 r_2 ... r_n$ accepts $w = \epsilon w_1 ... w_n$ because

- $r_0 \in \Delta(r, \epsilon)$,
- $r_{i+1} \in \Delta(r_i, w_{i+1})$, $i = 0, 1, ..., n-1$
- $r_n$ is an accept state of $N_1$.

So $N$ accepts $w$.

Is that all? What else are needed to be done?
1.2 Nondeterminism

1. Let \( w = w_1 w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_{\varepsilon} \)
1.2 Nondeterminism

1. Let $w = w_1 w_2 \ldots w_n \in L_1 \cup L_2$, where $w_i \in \Sigma_\epsilon$

2. Then $w \in L_1$ or $w \in L_2$. 
1.2 Nondeterminism

1. Let $w = w_1w_2 \ldots w_n \in L_1 \cup L_2$, where $w_i \in \Sigma_\epsilon$

2. Then $w \in L_1$ or $w \in L_2$.

3. Assume $w \in L_1$ (the case of $w \in L_2$ can be proved similarly.)
1.2 Nondeterminism

1. Let $w = w_1w_2 \ldots w_n \in L_1 \cup L_2$, where $w_i \in \Sigma_e$

2. Then $w \in L_1$ or $w \in L_2$.

3. Assume $w \in L_1$ (the case of $w \in L_2$ can be proved similarly.)

4. Assume $N_1$ recognizes $L_1$; then $w$ is accepted by $N_1$. 
1.2 Nondeterminism

1. Let $w = w_1w_2 \ldots w_n \in L_1 \cup L_2$, where $w_i \in \Sigma_e$

2. Then $w \in L_1$ or $w \in L_2$.

3. Assume $w \in L_1$ (the case of $w \in L_2$ can be proved similarly.)

4. Assume $N_1$ recognizes $L_1$; then $w$ is accepted by $N_1$.

5. There must be a sequence of states $r_0r_1 \ldots r_n$, such that
   - $r_0$ is the start state of $N_1$,
   - $r_{i+1} \in \delta_1(r_i, w_{i+1})$, $i = 0, 1, \ldots, n - 1$
   - $r_n$ is an accept state of $N_1$. 

1.2 Nondeterminism

1. Let \( w = w_1w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_\epsilon \)

2. Then \( w \in L_1 \) or \( w \in L_2 \).

3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)

4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).

5. There must be a sequence of states \( r_0r_1 \ldots r_n \), such that
   - \( r_0 \) is the start state of \( N_1 \),
   - \( r_{i+1} \in \delta_1(r_i, w_{i+1}) \), \( i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N_1 \).

6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).
1.2 Nondeterminism

1. Let \( w = w_1w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_\epsilon \)
2. Then \( w \in L_1 \) or \( w \in L_2 \).
3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)
4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).
5. There must be a sequence of states \( r_0r_1 \ldots r_n \), such that
   - \( r_0 \) is the start state of \( N_1 \),
   - \( r_{i+1} \in \delta_1(r_i, w_{i+1}) \), \( i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N_1 \).
6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).
7. According to the construction of \( N \), it is clear the sequence of states of \( N \)
   \( \bar{r}r_0r_2 \ldots r_n \) accepts \( w = \epsilon w_1 \ldots w_n \) because
1.2 Nondeterminism

1. Let \( w = w_1 w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_e \)

2. Then \( w \in L_1 \) or \( w \in L_2 \).

3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)

4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).

5. There must be a sequence of states \( r_0 r_1 \ldots r_n \), such that
   - \( r_0 \) is the start state of \( N_1 \),
   - \( r_{i+1} \in \delta_1(r_i, w_{i+1}) \), \( i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N_1 \).

6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).

7. According to the construction of \( N \), it is clear the sequence of states of \( N \)
   \( \overline{r} r_0 r_2 \ldots r_n \) accepts \( w = \epsilon w_1 \ldots w_n \) because
   - \( r_0 \in \Delta(\overline{r}, \epsilon) \),
1.2 Nondeterminism

1. Let \( w = w_1 w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_{\epsilon} \)

2. Then \( w \in L_1 \) or \( w \in L_2 \).

3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)

4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).

5. There must be a sequence of states \( r_0 r_1 \ldots r_n \), such that
   
   - \( r_0 \) is the start state of \( N_1 \),
   - \( r_{i+1} \in \delta_1(r_i, w_{i+1}), \ i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N_1 \).

6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).

7. According to the construction of \( N \), it is clear the sequence of states of \( N \)
   
   \( \bar{r} r_0 r_2 \ldots r_n \) accepts \( w = \epsilon w_1 \ldots w_n \) because
   
   - \( r_0 \in \Delta(\bar{r}, \epsilon) \), and
1.2 Nondeterminism

1. Let \( w = w_1 w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_e \)

2. Then \( w \in L_1 \) or \( w \in L_2 \).

3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)

4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).

5. There must be a sequence of states \( r_0 r_1 \ldots r_n \), such that
   - \( r_0 \) is the start state of \( N_1 \),
   - \( r_{i+1} \in \delta_1(r_i, w_{i+1}), i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N_1 \).

6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).

7. According to the construction of \( N \), it is clear the sequence of states of \( N_{\overline{r}} r_0 r_2 \ldots r_n \) accepts \( w = \epsilon w_1 \ldots w_n \) because
   - \( r_0 \in \Delta(\overline{r}, \epsilon) \), and
   - \( r_{i+1} \in \Delta(r_i, w_{i+1}), i = 0, 1, \ldots, n - 1 \)
1.2 Nondeterminism

1. Let \( w = w_1w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma_{\epsilon} \).
2. Then \( w \in L_1 \) or \( w \in L_2 \).
3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)
4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).
5. There must be a sequence of states \( r_0r_1 \ldots r_n \), such that
   - \( r_0 \) is the start state of \( N_1 \),
   - \( r_{i+1} \in \delta_1(r_i, w_{i+1}), i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N_1 \).
6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).
7. According to the construction of \( N \), it is clear the sequence of states of \( N \)
   \( \bar{r}r_0r_2 \ldots r_n \) accepts \( w = \epsilon w_1 \ldots w_n \) because
   - \( r_0 \in \Delta(r, \epsilon) \), and
   - \( r_{i+1} \in \Delta(r_i, w_{i+1}), i = 0, 1, \ldots, n - 1 \)
   - \( r_n \) is an accept state of \( N \).
1.2 Nondeterminism

1. Let $w = w_1w_2 \ldots w_n \in L_1 \cup L_2$, where $w_i \in \Sigma_\epsilon$

2. Then $w \in L_1$ or $w \in L_2$.

3. Assume $w \in L_1$ (the case of $w \in L_2$ can be proved similarly.)

4. Assume $N_1$ recognizes $L_1$; then $w$ is accepted by $N_1$.

5. There must be a sequence of states $r_0r_1 \ldots r_n$, such that
   - $r_0$ is the start state of $N_1$,
   - $r_{i+1} \in \delta_1(r_i, w_{i+1})$, $i = 0, 1, \ldots, n - 1$
   - $r_n$ is an accept state of $N_1$.

6. On the combined NFA $N$, $w$ can be written as $\epsilon w_1 \ldots w_n$.

7. According to the construction of $N$, it is clear the sequence of states of $N$
   $\overline{r}r_0r_2 \ldots r_n$ accepts $w = \epsilon w_1 \ldots w_n$ because
   - $r_0 \in \Delta(\overline{r}, \epsilon)$, and
   - $r_{i+1} \in \Delta(r_i, w_{i+1})$, $i = 0, 1, \ldots, n - 1$
   - $r_n$ is an accept state of $N$.

8. So $N$ accepts $w$. 

Is that all? What else are needed to be done?
1.2 Nondeterminism

1. Let \( w = w_1 w_2 \ldots w_n \in L_1 \cup L_2 \), where \( w_i \in \Sigma \epsilon \)
2. Then \( w \in L_1 \) or \( w \in L_2 \).
3. Assume \( w \in L_1 \) (the case of \( w \in L_2 \) can be proved similarly.)
4. Assume \( N_1 \) recognizes \( L_1 \); then \( w \) is accepted by \( N_1 \).
5. There must be a sequence of states \( r_0 r_1 \ldots r_n \), such that
   • \( r_0 \) is the start state of \( N_1 \),
   • \( r_{i+1} \in \delta_1(r_i, w_{i+1}) \), \( i = 0, 1, \ldots, n - 1 \)
   • \( r_n \) is an accept state of \( N_1 \).
6. On the combined NFA \( N \), \( w \) can be written as \( \epsilon w_1 \ldots w_n \).
7. According to the construction of \( N \), it is clear the sequence of states of \( N \)
   \( \bar{r} r_0 r_2 \ldots r_n \) accepts \( w = \epsilon w_1 \ldots w_n \) because
   • \( r_0 \in \Delta(\bar{r}, \epsilon) \), and
   • \( r_{i+1} \in \Delta(r_i, w_{i+1}) \), \( i = 0, 1, \ldots, n - 1 \)
   • \( r_n \) is an accept state of \( N \).
8. So \( N \) accepts \( w \).

Is that all? What else are needed to be done?
1.2 Nondeterminism

We actually need to prove:
for any \( w \in \Sigma^* \), \( w \in L_1 \cup L_2 \) if and only if \( N \) accepts \( w \). That is

(1) if \( w \in L_1 \cup L_2 \) then \( N \) accepts \( w \);

(2) if \( w \notin L_1 \cup L_2 \) then \( N \) does not accept \( w \);

We have just proved Part (1). Part (2) can be restated as:

(2) if \( N \) accepts \( w \), then \( w \in L_1 \cup L_2 \).
1.2 Nondeterminism

We actually need to prove:

for any $w \in \Sigma^*$, $w \in L_1 \cup L_2$ if and only if $N$ accepts $w$. That is

1) if $w \in L_1 \cup L_2$ then $N$ accepts $w$;
2) if $w \not\in L_1 \cup L_2$ then $N$ does not accept $w$.

We have just proved Part (1). Part (2) can be restated as:

2) if $N$ accepts $w$, then $w \in L_1 \cup L_2$.
1.2 Nondeterminism

We actually need to prove:

for any \( w \in \Sigma^* \), \( w \in L_1 \cup L_2 \) if and only if \( N \) accepts \( w \). That is
We actually need to prove:

for any $w \in \Sigma^*$, $w \in L_1 \cup L_2$ if and only if $N$ accepts $w$. That is

1. if $w \in L_1 \cup L_2$ then $N$ accepts $w$;
2. if $w \notin L_1 \cup L_2$ then $N$ does not accept $w$;
1.2 Nondeterminism

We actually need to prove:

for any $w \in \Sigma^*$, $w \in L_1 \cup L_2$ if and only if $N$ accepts $w$. That is

(1) if $w \in L_1 \cup L_2$ then $N$ accepts $w$;
(2) if $w \notin L_1 \cup L_2$ then $N$ does not accepts $w$;

We have just proved Part (1). Part (2) can be restated as:
We actually need to prove:

for any $w \in \Sigma^*$, $w \in L_1 \cup L_2$ if and only if $N$ accepts $w$. That is

(1) if $w \in L_1 \cup L_2$ then $N$ accepts $w$;
(2) if $w \notin L_1 \cup L_2$ then $N$ does not accepts $w$;

We have just proved Part (1). Part (2) can be restated as:

(2) if $N$ accepts $w$, then $w \in L_1 \cup L_2$. 


1.2 Nondeterminism

We actually need to prove:

for any \( w \in \Sigma^* \), \( w \in L_1 \cup L_2 \) if and only if \( N \) accepts \( w \). That is

(1) if \( w \in L_1 \cup L_2 \) then \( N \) accepts \( w \);
(2) if \( w \notin L_1 \cup L_2 \) then \( N \) does not accepts \( w \);

We have just proved Part (1). Part (2) can be restated as:

(2) if \( N \) accepts \( w \), then \( w \in L_1 \cup L_2 \).

Example of language union and combined NFA.
1.2 Nondeterminism

Theorem 1.46 (page 60)
The class of regular languages is closed under the concatenation operation.

Proof: construct an NFA to recognize $L_1L_2$.

How to prove?
Example of language concatenation and combined NFA.
1.2 Nondeterminism

**Theorem 1.46** (page 60)

The class of regular languages is closed under the concatenation operation. Proof: construct an NFA to recognize $L_1 L_2$. How to prove? Example of language concatenation and combined NFA.
1.2 Nondeterminism

**Theorem 1.46** (page 60)

*The class of regular languages is closed under the concatenation operation.*
1.2 Nondeterminism

**Theorem 1.46** (page 60)

*The class of regular languages is closed under the concatenation operation.*

**Proof:** construct an NFA to recognize $L_1 L_2$. 

![Diagram of an NFA to recognize $L_1 L_2$.]
1.2 Nondeterminism

**Theorem 1.46** (page 60)

*The class of regular languages is closed under the concatenation operation.*

**Proof:** construct an NFA to recognize $L_1L_2$.

![Diagram of NFAs](image)

How to prove?
1.2 Nondeterminism

**Theorem 1.46** (page 60)

*The class of regular languages is closed under the concatenation operation.*

**Proof**: construct an NFA to recognize $L_1 L_2$.

---

**How to prove?**

Example of language concatenation and combined NFA.
1.2 Nondeterminism

\[ L^* = \{ \epsilon \} \cup L \cup L^2 \cup L^3 \cup \ldots \]

where \( L_k = LL_{k-1} \), \( k \geq 1 \)

**Theorem 1.49** (page 62)
The class of regular languages is closed under the star operation.

**Proof**
Construct an NFA to recognize \( L^* \).

**How to prove?**
Math induction on \( n \) for \( w \in L^n \).

- When \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
- Assume for all \( w \in L^k \), \( N \) accepts \( w \);
- Now if \( w \in L^{k+1} \), \( w \) can be written as \( w = xy \) where \( x \in L^k \), \( y \in L \).
  - By assumption \( N \) accepts \( x \);
  - \( x \) arrives at some accept state \( q_{\text{acc}} \) in \( N \);
  - Because of \( \epsilon \)-transition from \( q_{\text{acc}} \) to the start state of \( N \), \( xy \) also arrives at some accept state of \( N \). So \( N \) accepts \( w \).

Example of language star and combined NFA.
1.2 Nondeterminism

\[ L^* = \{ \epsilon \} \cup L \cup L^2 \cup L^3 \cup \ldots \] where \( L^k \equiv LL^{k-1}, k \geq 1 \)
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where } L^k = LL^{k-1}, \ k \geq 1 \]

**Theorem 1.49** (page 62)
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where } L^k = LL^{k-1}, \quad k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where } L^k = LL^{k-1}, \ k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).
1.2 Nondeterminism

\[ L^* = \{ \epsilon \} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where } L^k = LL^{k-1}, \ k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

**How to prove ?**
1.2 Nondeterminism

\[ L^* = \{ \epsilon \} \cup L \cup L^2 \cup L^3 \cup \ldots \]  
where \( L^k = LL^{k-1}, \; k \geq 1 \)

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

How to prove? math induction on \( n \) for \( w \in L^n \)
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where} \quad L^k = LL^{k-1}, \quad k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

How to prove? math induction on \( n \) for \( w \in L^n \)

- when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where} \quad L^k = LL^{k-1}, \quad k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

How to prove? math induction on \( n \) for \( w \in L^n \)

- when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
- assume for all \( w \in L^k \), \( N \) accepts \( w \);
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where } L^k = LL^{k-1}, \ k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

- **How to prove?** math induction on \( n \) for \( w \in L^n \)
  - when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
  - assume for all \( w \in L^k \), \( N \) accepts \( w \);
  - now if \( w \in L^{k+1} \), \( w \) can be written as \( w = xy \) where \( x \in L^k \), \( y \in L \).
1.2 Nondeterminism

\[ L^* = \{ \epsilon \} \cup L \cup L^2 \cup L^3 \cup \ldots \]
where \( L^k = LL^{k-1} \), \( k \geq 1 \)

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

How to prove? math induction on \( n \) for \( w \in L^n \)

- when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
- assume for all \( w \in L^k \), \( N \) accepts \( w \);
- now if \( w \in L^{k+1} \), \( w \) can be written as \( w = xy \) where \( x \in L^k \), \( y \in L \).
- by assumption \( N \) accepts \( x \);
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where} \quad L^k = LL^{k-1}, \quad k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

How to prove? math induction on \( n \) for \( w \in L^n \)

- when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
- assume for all \( w \in L^k \), \( N \) accepts \( w \);
- now if \( w \in L^{k+1} \), \( w \) can be written as \( w = xy \) where \( x \in L^k \), \( y \in L \);
- by assumption \( N \) accepts \( x \); \( x \) arrives at some accept state \( q_{acc} \) in \( N \);
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where } L^k = LL^{k-1}, \ k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

How to prove? math induction on \( n \) for \( w \in L^n \)

- when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
- assume for all \( w \in L^k \), \( N \) accepts \( w \);
- now if \( w \in L^{k+1} \), \( w \) can be written as \( w = xy \) where \( x \in L^k, y \in L \).
- by assumption \( N \) accepts \( x \); \( x \) arrives at some accept state \( q_{acc} \) in \( N \);
- because of \( \epsilon \)-transition from \( q_{acc} \) to the start state of \( N_1 \), \( xy \) also arrives at some accept state of \( N \). So \( N \) accepts \( w \).
1.2 Nondeterminism

\[ L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots \quad \text{where} \quad L^k = LL^{k-1}, \quad k \geq 1 \]

**Theorem 1.49** (page 62)

*The class of regular languages is closed under the star operation.*

**Proof** construct an NFA to recognize \( L^* \).

**How to prove?** math induction on \( n \) for \( w \in L^n \)

- when \( n = 0 \), we have \( w = \epsilon \) and \( N \) accepts \( \epsilon \);
- assume for all \( w \in L^k \), \( N \) accepts \( w \);
- now if \( w \in L^{k+1} \), \( w \) can be written as \( w = xy \) where \( x \in L^k \), \( y \in L \).
- by assumption \( N \) accepts \( x \); \( x \) arrives at some accept state \( q_{acc} \) in \( N \);
- because of \( \epsilon \)-transition from \( q_{acc} \) to the start state of \( N_1 \),
  \( xy \) also arrives at some accept state of \( N \). So \( N \) accepts \( w \).

Example of language star and combined NFA.
1.3 Regular Expressions

- Natural languages are too vague for language definition.
- FAs are precise but not succinct for language definition.
- Regular expressions are used to define regular languages.
1.3 Regular Expressions

Regular expressions

• Natural languages are too vague for language definition.
1.3 Regular Expressions

**Regular expressions**

- Natural languages are too vague for language definition.
- FAs are precise but not succinct for language definition.
1.3 Regular Expressions

Regular expressions

- Natural languages are too vague for language definition.
- FAs are precise but not succinct for language definition.
- Regular expressions are used to define regular languages.
1.3 Regular Expressions

For example, a language in which each string contains either substring 11 or 001, i.e., each string can have "anything" as prefix, "anything" as suffix, and contain 11 or 001 in between such set of strings can be expressed as

\[(1 \cup 0)^* (11 \cup 001) (1 \cup 0)^*\]

Another example: a language whose strings have symbol a in the third last position.

\[(a \cup b)^* a (a \cup b)^* (a \cup b)^*\]
1.3 Regular Expressions

For example,
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such **set of strings** can be expressed as

\[(1 \cup 0)^*\]
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such set of strings can be expressed as

$$(1 \cup 0)^*(11 \cup 001)$$
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such set of strings can be expressed as

\[(1 \cup 0)^*(11 \cup 001)(1 \cup 0)^*\]
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such set of strings can be expressed as

$$(1 \cup 0)^*(11 \cup 001)(1 \cup 0)^*$$

Another example: a language whose strings have symbol $a$ in the third
last position.
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such set of strings can be expressed as

\[(1 \cup 0)^*(11 \cup 001)(1 \cup 0)^*\]

Another example: a language whose strings have symbol a in the third last position.

\[(a \cup b)^*\]
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such set of strings can be expressed as

\[(1 \cup 0)^* (11 \cup 001) (1 \cup 0)^*\]

Another example: a language whose strings have symbol a in the third last position.

\[(a \cup b)^* a\]
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix,
and contain 11 or 001 in between

such set of strings can be expressed as

\[(1 \cup 0)^* (11 \cup 001)(1 \cup 0)^*\]

Another example: a language whose strings have symbol a in the third
last position.

\[(a \cup b)^* a(a \cup b)\]
1.3 Regular Expressions

For example,

a language in which each string contains either substring 11 or 001
i.e., each string can have “anything” as prefix, “anything” as suffix, and contain 11 or 001 in between

such set of strings can be expressed as

\[(1 \cup 0)^*(11 \cup 001)(1 \cup 0)^*\]

Another example: a language whose strings have symbol \(a\) in the third last position.

\[(a \cup b)^*a(a \cup b)(a \cup b)\]
1.3 Regular Expressions

Notations for regular expressions

1. Use $0$ to represent $\{0\}$, and $1$ for $\{1\}$;
2. Use $\ast$ to represent "repeats", e.g., $1 \ast = \{\epsilon, 1, 11, 111, ...\}$;
3. Use $\cup$ for 'OR', e.g., $1 \cup 0$ represents $\{0, 1\}$;
4. Use $\circ$ for 'concatenation', e.g., $0 \circ 1 \ast$ represents $\{0, 01, 011, 0111, ...\}$, often $\circ$ can be removed: $01 \ast$;
5. Use '(', ')' whenever needed (as in arithmetic expressions).
1.3 Regular Expressions

Notations for regular expressions

1. Use $0$ to represent $\{0\}$, and $1$ for $\{1\}$;
2. Use $\ast$ to represent "repeats", e.g., $1 \ast = \{\epsilon, 1, 11, ...\}$;
3. Use $\cup$ for 'OR', e.g., $1 \cup 0$ represents $\{0, 1\}$;
4. Use $\circ$ for 'concatenation', e.g., $0 \circ 1 \ast$ represents $\{0, 01, 011, 0111, ...\}$, often $\circ$ can be removed: $01 \ast$;
5. Use '(', ')' whenever needed (as in arithmetic expressions).
1.3 Regular Expressions

Notations for regular expressions

1. Use 0 to represent \{0\}, and 1 for \{1\};
1.3 Regular Expressions

Notations for regular expressions

1. Use 0 to represent \{0\}, and 1 for \{1\};
2. Use \* to represent “repeats”, e.g., \1^* = \{\epsilon, 1, 11, \ldots \};
1.3 Regular Expressions

Notations for regular expressions

1. Use 0 to represent \{0\}, and 1 for \{1\};
2. Use * to represent “repeats”, e.g., 1* = \{\epsilon, 1, 11, \ldots \};
3. Use \cup for ‘OR’, e.g., 1 \cup 0 represents \{0, 1\}
1.3 Regular Expressions

Notations for regular expressions

1. Use 0 to represent \{0\}, and 1 for \{1\};
2. Use * to represent "repeats", e.g., 1* = \{ε, 1, 11, \ldots \};
3. Use ∪ for 'OR', e.g., 1 ∪ 0 represents \{0, 1\}
4. Use ◦ for 'concatenation',
   e.g., 0 ◦ 1* represents \{0, 01, 011, 0111, \ldots \},
1.3 Regular Expressions

Notations for regular expressions

1. Use 0 to represent \{0\}, and 1 for \{1\};
2. Use * to represent "repeats", e.g., 1* = \{\epsilon, 1, 11, \ldots \};
3. Use \cup for 'OR', e.g., 1 \cup 0 represents \{0, 1\}
4. Use \circ for 'concatenation',
   e.g., 0 \circ 1* represents \{0, 01, 011, 0111, \ldots \},
   often \circ can be removed: 01*
1.3 Regular Expressions

Notations for regular expressions

1. Use 0 to represent \{0\}, and 1 for \{1\};

2. Use \* to represent “repeats”, e.g., \(1^* = \{\epsilon, 1, 11, \ldots \}\);

3. Use \(\cup\) for ’OR’, e.g., \(1 \cup 0\) represents \{0, 1\}

4. Use \(\circ\) for ’concatenation’,
   e.g., \(0 \circ 1^*\) represents \{0, 01, 011, 0111, \ldots \},
   often \(\circ\) can be removed: \(01^*\)

5. Use ‘(’, ’)’ whenever needed (as in arithmetic expressions).
1.3 Regular Expressions

More examples:

\[(1 \cup 0)^*\]

represents language \[\{00, 01, 10, 11\}\]

\[(0 \cup 1)^*\] = \[\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}\] = \[\Sigma^*\] where \[\Sigma = \{0, 1\}\]

\[(0 \cup 1)^* \circ 1 \circ 1 \circ (1 \cup 0)^*\] , or simply \[\Sigma^* 111 \Sigma^* 0 \Sigma^* 0 \cup 1 \Sigma^* 1 \cup 1 \cup 0\]

\[(0 \cup \epsilon)^* 1^*\] = \[01^* \cup 1^*\]

\[(0 \cup \epsilon)^* 1^*\] = \[\emptyset\]

\[\emptyset^* = \emptyset\]
1.3 Regular Expressions

More examples:
1.3 Regular Expressions

More examples:

\[(1 \cup 0) \circ (1 \cup 0)\] represents language
1.3 Regular Expressions

More examples:

\[(1 \cup 0) \circ (1 \cup 0)\] represents language \(\{00, 01, 10, 11\}\)
1.3 Regular Expressions

More examples:

$(1 \cup 0) \circ (1 \cup 0)$ represents language $\{00, 01, 10, 11\}$

$(0 \cup 1)^*$
1.3 Regular Expressions

More examples:

\((1 \cup 0) \circ (1 \cup 0)\) represents language \( \{00, 01, 10, 11\}\)

\((0 \cup 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} = \Sigma^*\) where \(\Sigma = \{0, 1\}\)
1.3 Regular Expressions

More examples:

$$(1 \cup 0) \circ (1 \cup 0)$$ represents language \{00, 01, 10, 11\}

$$(0 \cup 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots \} = \Sigma^* \text{ where } \Sigma = \{0, 1\}$$

$$(0 \cup 1)^* \circ 1 \circ 1 \circ 1 \circ (1 \cup 0)^* \text{, or simply } \Sigma^*111\Sigma^*$$
1.3 Regular Expressions

More examples:

\((1 \cup 0) \circ (1 \cup 0)\) represents language \{00, 01, 10, 11\}

\((0 \cup 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} = \Sigma^*\) where \(\Sigma = \{0, 1\}\)

\((0 \cup 1)^* \circ 1 \circ 1 \circ 1 \circ (1 \cup 0)^*\), or simply \(\Sigma^*111\Sigma^*\)

\(0\Sigma^*0 \cup 1\Sigma^*1 \cup 1 \cup 0\)
1.3 Regular Expressions

More examples:

\((1 \cup 0) \circ (1 \cup 0)\) represents language \(\{00, 01, 10, 11\}\)

\((0 \cup 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}\) = \(\Sigma^*\) where \(\Sigma = \{0, 1\}\)

\((0 \cup 1)^* \circ 1 \circ 1 \circ 1 \circ (1 \cup 0)^*\), or simply \(\Sigma^*111\Sigma^*\)

\(0\Sigma^*0 \cup 1\Sigma^*1 \cup 1 \cup 0\)

\((0 \cup \epsilon)1^* = 01^* \cup 1^*\)
1.3 Regular Expressions

More examples:

$$(1 \cup 0) \circ (1 \cup 0)$$ represents language $$\{00, 01, 10, 11\}$$

$$(0 \cup 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} = \Sigma^*$$ where $$\Sigma = \{0, 1\}$$

$$(0 \cup 1)^* \circ 1 \circ 1 \circ 1 \circ (1 \cup 0)^*$$, or simply $$\Sigma^*111\Sigma^*$$

$$0\Sigma^*0 \cup 1\Sigma^*1 \cup 1 \cup 0$$

$$(0 \cup \epsilon)1^* = 01^* \cup 1^*$$

$$(0 \cup \epsilon)(1 \cup \epsilon) = \{\epsilon, 1, 0, 01\}$$
1.3 Regular Expressions

More examples:

\((1 \cup 0) \circ (1 \cup 0)\) represents language \(\{00, 01, 10, 11\}\)

\((0 \cup 1)^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} = \Sigma^*\) \(\text{where } \Sigma = \{0, 1\}\)

\((0 \cup 1)^* \circ 1 \circ 1 \circ 1 \circ (1 \cup 0)^*\), or simply \(\Sigma^*111\Sigma^*\)

\(0\Sigma^*0 \cup 1\Sigma^*1 \cup 1 \cup 0\)

\((0 \cup \varepsilon)1^* = 01^* \cup 1^*\)

\((0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 1, 0, 01\}\)

\(1^*\emptyset = \emptyset\)
More examples:

\((1 \cup 0) \circ (1 \cup 0)\) represents language \(\{00, 01, 10, 11\}\)

\((0 \cup 1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} = \Sigma^*\) where \(\Sigma = \{0, 1\}\)

\((0 \cup 1)^* \circ 1 \circ 1 \circ 1 \circ (1 \cup 0)^*\), or simply \(\Sigma^*111\Sigma^*\)

\(0\Sigma^*0 \cup 1\Sigma^*1 \cup 1 \cup 0\)

\((0 \cup \epsilon)1^* = 01^* \cup 1^*\)

\((0 \cup \epsilon)(1 \cup \epsilon) = \{\epsilon, 1, 0, 01\}\)

\(1^*\emptyset = \emptyset\)

\(\emptyset^* = ?\)
1.3 Regular Expressions

**Formal Definition of a Regular Expression**

**Definition 1.52**

$R$ is a regular expression if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.

Also we use $R^+$ for $R^* R$, for example, $1^+ = \{1, 11, 111, \ldots\}$.
Formal Definition of a Regular Expression
1.3 Regular Expressions

Formal Definition of a Regular Expression

Definition 1.52 (page 64)
1.3 Regular Expressions

Formal Definition of a Regular Expression

Definition 1.52 (page 64)

$R$ is a regular expression if $R$ is one of the followings:

1. A symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1)^*$, where $R_1$ is a regular expression.

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.

Also we use $R^+$ for $RR^*$, for example, $1^+ = \{1, 11, 111, \ldots\}$. 
1.3 Regular Expressions

Formal Definition of a Regular Expression

**Definition 1.52** (page 64)

*R* is a *regular expression* if *R* is one of the followings:

1. a symbol \( a \in \Sigma \) the alphabet
1.3 Regular Expressions

Formal Definition of a Regular Expression

**Definition 1.52** (page 64)

$R$ is a *regular expression* if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.

Also we use $R^+$ for $RR^*$, for example, $1^+ = \{1, 11, 111, ...\}$.
1.3 Regular Expressions

Formal Definition of a Regular Expression

Definition 1.52 (page 64)

$R$ is a regular expression if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.

Also we use $R^+$ for $R^*$, for example, $1^+$ = \{1, 11, 111, ...\}
1.3 Regular Expressions

Formal Definition of a Regular Expression

**Definition 1.52** (page 64)

$R$ is a *regular expression* if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union. Also we use $R^+$ for $RR^*$, for example, $1^+ = \{1, 11, 111, \ldots\}$.
1.3 Regular Expressions

Formal Definition of a Regular Expression

Definition 1.52 (page 64)

$R$ is a regular expression if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.

Also we use $R^+$ for $RR^*$, for example, $1^+ = \{1, 11, 111, \ldots\}$.
1.3 Regular Expressions

Formal Definition of a Regular Expression

Definition 1.52 (page 64)

$R$ is a regular expression if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.
1.3 Regular Expressions

Formal Definition of a Regular Expression

Definition 1.52 (page 64)

$R$ is a *regular expression* if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.
1.3 Regular Expressions

Formal Definition of a Regular Expression

**Definition 1.52** (page 64)

$R$ is a *regular expression* if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Without the parentheses, evaluation of a regular expression is done in the precedence order: star, then concatenation, then union.

Also we use $R^+$ for $RR^*$, for example, $1^+ = \{1, 11, 111, \ldots \}$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already:

- $R \cup \epsilon = R$
- $R \circ \epsilon = R$
- $R \cup \emptyset = R$
- $R \circ \emptyset = R$

More examples:

$\{w : w$ starts with $0$ and has even length or starts with $1$ and has odd length $\}$

Answer:

$0((1 \cup 0)(1 \cup 0))^* (1 \cup 0) \cup 1((1 \cup 0)(1 \cup 0))^*$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

\[
\begin{align*}
R \cup \epsilon &= R \\
R \circ \epsilon &= R \\
R \cup \emptyset &= R \\
R \circ \emptyset &= R
\end{align*}
\]

More examples:

\[
\{w : w \text{ starts with 0 and has even length or starts with 1 and has odd length}\}
\]

Answer:

\[
0 \left((1 \cup 0)(1 \cup 0)\right)^* (1 \cup 0) \cup 1 \left((1 \cup 0)(1 \cup 0)\right)^*
\]
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon = ?$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon = ?$

$R \circ \epsilon = ?$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$$R \cup \epsilon = ?$$

$$R \circ \epsilon = ?$$

$$R \cup \emptyset = ?$$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon = ?$

$R \circ \epsilon = ?$

$R \cup \emptyset = ?$

$R \circ \emptyset = ?$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon = ?$

$R \circ \epsilon = ?$

$R \cup \emptyset = ?$

$R \circ \emptyset = ?$

More examples:

$\{w : w$ starts with 0 and has even length or starts with 1 and has odd length$\}$
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon = ?$

$R \circ \epsilon = ?$

$R \cup \emptyset = ?$

$R \circ \emptyset = ?$

More examples:

$\{w : w \text{ starts with 0 and has even length or starts with 1 and has odd length}\}$

Answer:

0
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon =?$

$R \circ \epsilon =?$

$R \cup \emptyset =?$

$R \circ \emptyset =?$

More examples:

$\{w : w$ starts with 0 and has even length or starts with 1 and has odd length$\}$

Answer:

$0((1 \cup 0)(1 \cup 0))^*(1 \cup 0)$
1.3 Regular Expressions

Additional special cases, where \( R \) is a regular expression already

\[
R \cup \epsilon = ? \\
R \circ \epsilon = ? \\
R \cup \emptyset = ? \\
R \circ \emptyset = ?
\]

More examples:

\{ w : w \text{ starts with 0 and has even length or starts with 1 and has odd length} \}

Answer:

\[
0((1 \cup 0)(1 \cup 0))^*(1 \cup 0) \cup
\]
1.3 Regular Expressions

Additional special cases, where \( R \) is a regular expression already

\[
\begin{align*}
R \cup \epsilon &= ? \\
R \circ \epsilon &= ? \\
R \cup \emptyset &= ? \\
R \circ \emptyset &= ?
\end{align*}
\]

More examples:

\([w : w \text{ starts with 0 and has even length or starts with 1 and has odd length}]\)

Answer:

\[
0((1 \cup 0)(1 \cup 0))^*(1 \cup 0) \cup 1
\]
1.3 Regular Expressions

Additional special cases, where $R$ is a regular expression already

$R \cup \epsilon = ?$

$R \circ \epsilon = ?$

$R \cup \emptyset = ?$

$R \circ \emptyset = ?$

More examples:

$\{w : w \text{ starts with } 0 \text{ and has even length or starts with } 1 \text{ and has odd length} \}$

Answer:

$0((1 \cup 0)(1 \cup 0))^*(1 \cup 0) \cup 1((1 \cup 0)(1 \cup 0))^*$
1.3 Regular Expressions

Theorem 1.54

A language is regular if and only if some regular expression describes it. The theorem has two directions (stated in the following lemmas.)

Lemma 1.55

If a language is described by a regular expression, then it is regular.

Lemma 1.60

If a language is regular, then it is described by a regular expression.
Theorem 1.54 (page 66)
A language is regular if and only if some regular expression describes it.
1.3 Regular Expressions

**Theorem 1.54** (page 66)

A language is regular if and only if some regular expression describes it.

The theorem has two directions (stated in the following lemmas.)

**Lemma 1.55** (page 67)

If a language is described by a regular expression, then it is regular.
1.3 Regular Expressions

**Theorem 1.54** (page 66)

A language is regular if and only if some regular expression describes it.

The theorem has two directions (stated in the following lemmas.)

**Lemma 1.55** (page 67)

*If a language is described by a regular expression, then it is regular.*

**Lemma 1.60** (page 69)

*If a language is regular, then it is described by a regular expression.*
1.3 Regular Expressions

Lemma 1.55
If a language is described by a regular expression, then it is regular.

proof idea
• Convert a regular expression $R$ to an NFA;
• by following constructing rules of regular expressions;
• to assemble NFAs in a bottom up fashion.

First, try an example. Given regular expression:
$((ab)^* \cup (ba)^*)^*aaa^*$
build an NFA to recognize the described language.
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.
Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof idea
Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof idea

• Convert a regular expression $R$ to an NFA;
1.3 Regular Expressions

Lemma 1.55 (page 67)

If a language is described by a regular expression, then it is regular.

proof idea

- Convert a regular expression $R$ to an NFA;
- by following constructing rules of regular expressions;
Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof idea

• Convert a regular expression $R$ to an NFA;
• by following constructing rules of regular expressions;
• to assemble NFAs in a bottom up fashion.
Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof idea

- Convert a regular expression $R$ to an NFA;
- by following constructing rules of regular expressions;
- to assemble NFAs in a bottom up fashion.

First, try an example. Given regular expression:

$$(((ab)$$
Lemma 1.55 (page 67)

*If a language is described by a regular expression, then it is regular.*

**proof idea**

- Convert a regular expression $R$ to an NFA;
- by following constructing rules of regular expressions;
- to assemble NFAs in a bottom up fashion.

First, try an example. Given regular expression:

$$(((ab)^*)$$
Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof idea

- Convert a regular expression $R$ to an NFA;
- by following constructing rules of regular expressions;
- to assemble NFAs in a bottom up fashion.

First, try an example. Given regular expression:

$$(((ab)^* \cup (ba)^*)$$
Lemma 1.55 (page 67)
*If a language is described by a regular expression, then it is regular.*

**proof idea**
- Convert a regular expression \( R \) to an NFA;
- by following constructing rules of regular expressions;
- to assemble NFAs in a bottom up fashion.

First, try an example. Given regular expression:

\[
(((ab)^* \cup (ba)^*) (aaa)
\]
1.3 Regular Expressions

**Lemma 1.55** (page 67)  
*If a language is described by a regular expression, then it is regular.*

**proof idea**

- Convert a regular expression $R$ to an NFA;
- by following constructing rules of regular expressions;
- to assemble NFAs in a bottom up fashion.

First, try an example. Given regular expression:

$(((ab)^* \cup (ba)^*)(aaa))^*$

build an NFA to recognize the described language.
1.3 Regular Expressions

In general, given regular expression $R$,

- build NFAs for "atomic regular expressions" in $R$,
- assemble these NFAs into a larger NFA for $R$ according to how the rules used to "assemble" $R$.

That is

- prove all atomic regular expressions have corresponding NFAs;
- prove if all sub-regular expressions in $R$ have corresponding NFAs;
- then $R$ has a corresponding NFA.

Use structural induction
1.3 Regular Expressions

In general, given regular expression $R$, ...
1.3 Regular Expressions

In general, given regular expression $R$,

- build NFAs for “atomic regular expressions” in $R$, and
1.3 Regular Expressions

In general, given regular expression $R$,

- build NFAs for “atomic regular expressions” in $R$, and
- assemble these NFAs into a larger NFA for $R$
  according to how the rules used to “assemble” $R$. 
1.3 Regular Expressions

In general, given regular expression $R$,

- build NFAs for “atomic regular expressions” in $R$, and
- assemble these NFAs into a larger NFA for $R$ according to how the rules used to “assemble” $R$.

That is
1.3 Regular Expressions

In general, given regular expression $R$,

- build NFAs for “atomic regular expressions” in $R$, and
- assemble these NFAs into a larger NFA for $R$ according to how the rules used to “assemble” $R$.

That is

- prove all atomic regular expressions have corresponding NFAs;
In general, given regular expression $R$, 

- build NFAs for “atomic regular expressions” in $R$, and
- assemble these NFAs into a larger NFA for $R$ according to how the rules used to “assemble” $R$.

That is

- prove all atomic regular expressions have corresponding NFAs;
- prove if all sub-regular expressions in $R$ have corresponding NFAs; then $R$ has a corresponding NFA.
1.3 Regular Expressions

In general, given regular expression $R$,

- build NFAs for “atomic regular expressions” in $R$, and
- assemble these NFAs into a larger NFA for $R$
  according to how the rules used to “assemble” $R$.

That is

- prove all atomic regular expressions have corresponding NFAs;
- prove if all sub-regular expressions in $R$ have corresponding NFAs; then $R$ has a corresponding NFA.

Use structural induction
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, ..., \} \]

is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \).

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

if \( y = 1 \), \( 1 \leq 1 \); so \( y \leq y^2 \);

otherwise, there is \( x \in N \) such that \( y = x + 1 \)
assume \( x^2 \geq x \)

then \( y^2 = (x + 1)^2 = x^2 + 2x + 1 \geq x + 2x + 1 > x + 1 = y \)

Therefore the claim is true.
1.3 Regular Expressions

Re-examine mathematical induction
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?
1.3 Regular Expressions

Re-examine mathematical induction

$N = \{1, 2, \ldots, \}$ is the set of natural numbers.

But how do you define $N$ more rigorously?

Define elements in $N$ inductively:

1. $1 \in N$;
2. For any $x \in N$, $x + 1 \in N$;
3. Only elements generated with rules (1) and (2) belong to $N$.

Claim: for every $y \in N$, $y \leq y^2$.

Prove using structural induction (mathematical induction for $N$).

Let $y$ be any element in $N$.

If $y = 1$, $1 \leq 1^2$; so $y \leq y^2$;

Otherwise, there is $x \in N$ such that $y = x + 1$; assume $x^2 \geq x$ then $y^2 = (x + 1)^2 = x^2 + 2x + 1 \geq x + 2x + 1 > x + 1 = y$.

Therefore the claim is true.
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

(1) \( 1 \in N \);
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. (1) \( 1 \in N \);
2. (2) for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \).

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

If \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);

Otherwise, there is \( x \in N \) such that \( y = x + 1 \).

Assume \( x^2 \geq x \) then \( y^2 = (x + 1)^2 = x^2 + 2x + 1 \geq x + 2x + 1 > x + 1 = y \).

Therefore the claim is true.
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

If \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);

otherwise, there is \( x \in N \) such that \( y = x + 1 \)

Assume \( x^2 \geq x \) then \( y^2 = (x + 1)^2 = x^2 + 2x + 1 \geq x + 2x + 1 > x + 1 = y \)

Therefore the claim is true.
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

(1) \( 1 \in N \);
(2) for any \( x \in N \), \( x + 1 \in N \);
(3) only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

\[ \text{Let } y \text{ be any element in } N. \]
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

If \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

if \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);
otherwise, there is \( x \in N \) such that \( y = x + 1 \)
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

if \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);
otherwise, there is \( x \in N \) such that \( y = x + 1 \)

assumed \( x^2 \geq x \)
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

1. \( 1 \in N \);
2. for any \( x \in N \), \( x + 1 \in N \);
3. only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

if \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);
otherwise, there is \( x \in N \) such that \( y = x + 1 \)

assume \( x^2 \geq x \)
then \( y^2 = (x + 1)^2 = x^2 + 2x + 1 \geq x + 2x + 1 > x + 1 = y \)
1.3 Regular Expressions

Re-examine mathematical induction

\[ N = \{1, 2, \ldots, \} \] is the set of natural numbers.

But how do you define \( N \) more rigorously?

Define elements in \( N \) inductively:

(1) \( 1 \in N \);
(2) for any \( x \in N \), \( x + 1 \in N \);
(3) only elements generated with rules (1) and (2) belong to \( N \)

Claim: for every \( y \in N \), \( y \leq y^2 \).

Prove using structural induction (mathematical induction for \( N \)).

Let \( y \) be any element in \( N \).

if \( y = 1 \), \( 1 \leq 1^2 \); so \( y \leq y^2 \);
otherwise, there is \( x \in N \) such that \( y = x + 1 \)

\[ \text{assume } x^2 \geq x \]

\[ \text{then } y^2 = (x + 1)^2 = x^2 + 2x + 1 \geq x + 2x + 1 > x + 1 = y \]

Therefore the claim is true.
1.3 Regular Expressions

Example: Let set $T$ be defined with the following (inductive rules):

1. $3 \in T$;
2. for any $x \in T$, $3x \in T$;
3. only elements generated with rules (1) and (2) belong to $T$.

What is $T$?

Powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3^x$ for some $x \in T$.

Assume $x$ is a power of 3, i.e., $x = 3^k$ for some integer $k$; then $y = 3^x = 3^{3^k} = 3^{3^k+1}$, also a power of 3.

Therefore, all elements in $T$ are powers of 3.
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$
2. For any $x \in T$, $3x \in T$
3. Only elements generated with rules (1) and (2) belong to $T$


Claim: Every $x \in T$ is a power of 3.

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$ for some $x \in T$.

Assume $x$ is a power of 3, i.e., $x = 3^k$ for some integer $k$;
then $y = 3x = 3^{k+1}$ also a power of 3.

Therefore, all elements in $T$ are powers of 3.
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. For any $x \in T$, $3x \in T$;
3. Only elements generated with rules (1) and (2) belong to $T$.

What is $T$?

Claim: Every $x \in T$ is a power of 3.

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$ for some $x \in T$.

Assume $x$ is a power of 3, i.e., $x = 3^k$ for some integer $k$;
then $y = 3x = 3^{k+1}$ also a power of 3.

Therefore, all elements in $T$ are powers of 3.
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. for any $x \in T$, $3x \in T$;
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. For any $x \in T$, $3x \in T$;
3. Only elements generated with rules (1) and (2) belong to $T$.
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. For any $x \in T$, $3x \in T$;
3. Only elements generated with rules (1) and (2) belong to $T$.

What is $T$?
Example: Let set $T$ be defined with following (inductive rules)

(1) $3 \in T$;
(2) for any $x \in T$, $3x \in T$;
(3) only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.
1.3 Regular Expressions

Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. for any $x \in T$, $3x \in T$;
3. only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?
Example: Let set $T$ be defined with following (inductive rules)

(1) $3 \in T$;
(2) for any $x \in T$, $3x \in T$;
(3) only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. for any $x \in T$, $3x \in T$;
3. only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$. 
Example: Let set $T$ be defined with following (inductive rules)

(1) $3 \in T$;
(2) for any $x \in T$, $3x \in T$;
(3) only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.
Example: Let set $T$ be defined with following (inductive rules)

(1) $3 \in T$;
(2) for any $x \in T$, $3x \in T$;
(3) only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$
1.3 Regular Expressions

Example: Let set $T$ be defined with following (inductive rules)

(1) $3 \in T$;
(2) for any $x \in T$, $3x \in T$;
(3) only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$ for some $x \in T$. 
1.3 Regular Expressions

Example: Let set $T$ be defined with following (inductive rules)

(1) $3 \in T$;
(2) for any $x \in T$, $3x \in T$;
(3) only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$ for some $x \in T$.

Assume $x$ is a power of 3, i.e, $x = 3^k$ for some integer $k$;
Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. for any $x \in T$, $3x \in T$;
3. only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$ for some $x \in T$.
Assume $x$ is a power of 3, i.e., $x = 3^k$ for some integer $k$;
then $y = 3x = 33^k = 3^{k+1}$ also power of 3.
1.3 Regular Expressions

Example: Let set $T$ be defined with following (inductive rules)

1. $3 \in T$;
2. for any $x \in T$, $3x \in T$;
3. only elements generated with rules (1) and (2) belong to $T$

What is $T$? powers of 3.

Claim: every $x \in T$ is a power of 3?

We prove the claim by following how each element in $T$ was created.

Let $y$ be any element in $T$.

If $y = 3$, then $y = 3^1$, a power of 3.

Otherwise, $y = 3x$ for some $x \in T$.
Assume $x$ is a power of 3, i.e, $x = 3^k$ for some integer $k$;
then $y = 3x = 33^k = 3^{k+1}$ also power of 3.

Therefore, all elements in $T$ are powers of 3.
1.3 Regular Expressions

Where is the "structure"? this is about the structure of how an element in the set is created:

\[ T = \{ 3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \]

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time a new element was included to the set, it has the property \( P \). So the idea of structural induction proof is to proceed in the same inductive way that \( T \) was defined.

\[ \begin{align*}
&\text{e.g., for } 3 \times 3 \times 3 \text{ • it belongs to } T \text{ because } 3 \times 3 \text{ already belongs to } T \\
&\text{• since } 3 \times 3 \text{ belongs to } T, \text{ it already have property } P, \\
&\text{• to prove } 3 \times 3 \times 3 \text{ also has the same property.}
\end{align*} \]
1.3 Regular Expressions

Where is the “structure”?
1.3 Regular Expressions

Where is the “structure”? 

this is about the structure of how an element in the set is created:

e.g., $T = \{ 3, 3 \times 3, 3 \times 3 \times 3, \ldots \}$

So to prove certain property $P$ holds for every element in $T$, it suffices to show that every time an new element was included to the set, it has the property $P$. 

So the idea of structural induction proof is to proceed in the same inductive way that $T$ was defined.

e.g., for $3 \times 3 \times 3$ it belongs to $T$ because $3 \times 3$ already belongs to $T$.

since $3 \times 3$ belongs to $T$, it already have property $P$,

• to prove $3 \times 3 \times 3$ also has the same property.
1.3 Regular Expressions

Where is the “structure”? 

this is about the structure of how an element in the set is created:

e.g., \( T = \{ 3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \)
Where is the “structure”? 

this is about the structure of how an element in the set is created:

\( T = \{3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \)

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time an new element was included to the set, it has the property \( P \).
Where is the “structure”? 

this is about the structure of how an element in the set is created: 

\[ T = \{3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \]

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time an new element was included to the set, it has the property \( P \).

So the idea of structural induction proof is to proceed in the same inductive way that \( T \) was defined.
1.3 Regular Expressions

Where is the “structure”?

this is about the structure of how an element in the set is created:

\[ T = \{3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \]

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time a new element was included to the set, it has the property \( P \).

So the idea of \textit{structural induction proof} is to proceed in the same inductive way that \( T \) was defined.

\[ \text{e.g., for } 3 \times 3 \times 3 \]
1.3 Regular Expressions

Where is the “structure”?  
this is about the structure of how an element in the set is created:

e.g., \( T = \{3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \)

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time a new element was included to the set, it has the property \( P \).

So the idea of structural induction proof is to proceed in the same inductive way that \( T \) was defined.

\hspace{0.5cm} \hspace{1cm} \hspace{1cm} e.g., for \( 3 \times 3 \times 3 \)

\hspace{0.5cm} \hspace{2cm} \hspace{1cm} \bull \hspace{0.5cm} \text{it belongs to } T \text{ because } 3 \times 3 \text{ already belongs to } T
1.3 Regular Expressions

Where is the “structure”? this is about the structure of how an element in the set is created:

\[ T = \{3, 3 \times 3, 3 \times 3 \times 3, \ldots\} \]

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time an new element was included to the set, it has the property \( P \).

So the idea of structural induction proof is to proceed in the same inductive way that \( T \) was defined.

e.g., for \( 3 \times 3 \times 3 \)

- it belongs to \( T \) because \( 3 \times 3 \) already belongs to \( T \)
- since \( 3 \times 3 \) belongs to \( T \), it already have property \( P \),
1.3 Regular Expressions

Where is the “structure”? 

this is about the structure of how an element in the set is created:

\[ T = \{ 3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \]

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time an new element was included to the set, it has the property \( P \).

So the idea of structural induction proof is to proceed in the same inductive way that \( T \) was defined.

- e.g., for \( 3 \times 3 \times 3 \)
  - it belongs to \( T \) because \( 3 \times 3 \) already belongs to \( T \)
  - since \( 3 \times 3 \) belongs to \( T \), it already have property \( P \),
  - to prove \( 3 \times 3 \times 3 \) also has the same property.
1.3 Regular Expressions

Where is the “structure”?

this is about the structure of how an element in the set is created:

\[ T = \{3, 3 \times 3, 3 \times 3 \times 3, \ldots \} \]

So to prove certain property \( P \) holds for every element in \( T \), it suffices to show that every time an new element was included to the set, it has the property \( P \).

So the idea of structural induction proof is to proceed in the same inductive way that \( T \) was defined.

\[
\text{e.g., for } 3 \times 3 \times 3
\]

- it belongs to \( T \) because \( 3 \times 3 \) already belongs to \( T \)
- since \( 3 \times 3 \) belongs to \( T \), it already have property \( P \),
- to prove \( 3 \times 3 \times 3 \) also has the same property.
1.3 Regular Expressions

Here is the definition for regular expressions:

1. A symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2$), where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2$), where $R_1, R_2$ are regular expressions
6. $(R_1^*$), where $R_1$ is a regular expression.

We can redefine it in a slightly different way:

The set $R$ of regular expressions is defined inductively:

1. For any $a \in \Sigma$, $a \in R$;
2. $\epsilon \in R$;
3. $\emptyset \in R$;
4. If $R_1, R_2 \in R$, then $(R_1 \cup R_2) \in R$;
5. If $R_1, R_2 \in R$, then $(R_1 \circ R_2) \in R$;
6. If $R_1 \in R$, then $(R_1^*) \in R$;
Here is the definition for regular expressions: the following are regular expressions:

1. A symbol \( a \in \Sigma \) where \( \Sigma \) is the alphabet.
2. \( \epsilon \) (the empty string).
3. \( \emptyset \) (the empty set).
4. \( (R_1 \cup R_2) \) where \( R_1 \) and \( R_2 \) are regular expressions.
5. \( (R_1 \circ R_2) \) where \( R_1 \) and \( R_2 \) are regular expressions.
6. \( (R_1)^* \) where \( R_1 \) is a regular expression.
Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1$, $R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1$, $R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1$, $R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1$, $R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a slightly different way:
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a **slightly different way**:

The set $\mathcal{R}$ of regular expressions is defined inductively:
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1$, $R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1$, $R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a **slightly different way**:

The set $\mathcal{R}$ of regular expressions is defined inductively:

1. for any $a \in \Sigma$, $a \in \mathcal{R}$;
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a slightly different way:

The set $\mathcal{R}$ of regular expressions is defined inductively:

1. for any $a \in \Sigma$, $a \in \mathcal{R}$;
2. $\epsilon \in \mathcal{R}$;
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a slightly different way:

The set $\mathcal{R}$ of regular expressions is defined inductively:

1. for any $a \in \Sigma$, $a \in \mathcal{R}$;
2. $\epsilon \in \mathcal{R}$;
3. $\emptyset \in \mathcal{R}$;
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a slightly different way:

The set $\mathcal{R}$ of regular expressions is defined inductively:

1. for any $a \in \Sigma$, $a \in \mathcal{R}$;
2. $\epsilon \in \mathcal{R}$;
3. $\emptyset \in \mathcal{R}$;
4. if $R_1, R_2 \in \mathcal{R}$, then $(R_1 \cup R_2) \in \mathcal{R}$;
1.3 Regular Expressions

Here is the definition for regular expressions: the following are regular expressions:

1. A symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1$, $R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1$, $R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

We can redefine it in a slightly different way:

The set $\mathcal{R}$ of regular expressions is defined inductively:

1. For any $a \in \Sigma$, $a \in \mathcal{R}$;
2. $\epsilon \in \mathcal{R}$;
3. $\emptyset \in \mathcal{R}$;
4. If $R_1, R_2 \in \mathcal{R}$, then $(R_1 \cup R_2) \in \mathcal{R}$;
5. If $R_1, R_2 \in \mathcal{R}$, then $(R_1 R_2) \in \mathcal{R}$;
Here is the definition for regular expressions: the following are regular expressions

1. a symbol \( a \in \Sigma \) the alphabet
2. \( \epsilon \)
3. \( \emptyset \)
4. \((R_1 \cup R_2)\), where \( R_1, R_2 \) are regular expressions
5. \((R_1 \circ R_2)\), where \( R_1, R_2 \) are regular expressions
6. \((R_1^*)\), where \( R_1 \) is a regular expression.

We can redefine it in a slightly different way:

The set \( \mathcal{R} \) of regular expressions is defined inductively:

1. for any \( a \in \Sigma \), \( a \in \mathcal{R} \);
2. \( \epsilon \in \mathcal{R} \);
3. \( \emptyset \in \mathcal{R} \);
4. if \( R_1, R_2 \in \mathcal{R} \), then \((R_1 \cup R_2) \in \mathcal{R} \);
5. if \( R_1, R_2 \in \mathcal{R} \), then \((R_1 R_2) \in \mathcal{R} \);
6. if \( R_1 \in \mathcal{R} \), then \((R_1^*) \in \mathcal{R} \);
1.3 Regular Expressions

Lemma 1.55

If a language is described by a regular expression, then it is regular.
proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

• If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$.

• Similar arguments for $R = \epsilon$, $R = \emptyset$; ...

• If $R = (R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions.
Assume $R_1, R_2$ describe regular languages $L_1, L_2$, respectively.
$(R_1 \cup R_2)$ describes $L_1 \cup L_2$ that is a regular language by Theorem 1.45.
Thus, $R$ describes a regular language.

• Similar arguments for cases 5 and 6; ...

by Theorems 1.46 and 1.49.
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)
1.3 Regular Expressions

**Lemma 1.55** (page 67)
*If a language is described by a regular expression, then it is regular.*

**proof:** *(Using structural induction)*

Recall
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6 $\left( R_1^* \right)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$. 
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\varepsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, 
1.3 Regular Expressions

**Lemma 1.55** (page 67)

*If a language is described by a regular expression, then it is regular.*

**proof:** (Using structural induction)

Recall \( R \) is a regular expression if \( R \) is one of the followings:

1. a symbol \( a \in \Sigma \) the alphabet
2. \( \epsilon \)
3. \( \emptyset \)
4. \( (R_1 \cup R_2) \), where \( R_1, R_2 \) are regular expressions
5. \( (R_1 \circ R_2) \), where \( R_1, R_2 \) are regular expressions
6. \( (R_1^*) \), where \( R_1 \) is a regular expression.

Let language \( L \) be described by regular expression \( R \).

- If \( R = a \) for some \( a \in \Sigma \), we can construct an NFA to recognize exactly strings described by \( a \).
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. . . . .
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. ... So $L$ is regular language.
1.3 Regular Expressions

**Lemma 1.55** (page 67)
*If a language is described by a regular expression, then it is regular.*

**proof:** *(Using structural induction)*

Recall *$R$* is a *regular expression* if *$R$* is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. .... So $L$ is regular language.

- Similar arguments for $R = \epsilon$, $R = \emptyset$; ....
1.3 Regular Expressions

**Lemma 1.55** (page 67)

*If a language is described by a regular expression, then it is regular.*

**proof:** *(Using structural induction)*

Recall *$R$* is a *regular expression* if $R$ is one of the followings:

1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. . . . So $L$ is regular language.

- Similar arguments for $R = \epsilon$, $R = \emptyset$; . . .

- If $R = (R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions.
Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. . . . So $L$ is regular language.

- Similar arguments for $R = \epsilon$, $R = \emptyset$; . . .

- If $R = (R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions. Assume $R_1, R_2$ describe regular languages $L_1, L_2$, respectively
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6 $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. . . . So $L$ is regular language.

- Similar arguments for $R = \epsilon$, $R = \emptyset$; . . .

- If $R = (R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions.
  Assume $R_1, R_2$ describe regular languages $L_1, L_2$, respectively 
  $(R_1 \cup R_2)$ describes $L_1 \cup L_2$
1.3 Regular Expressions

Lemma 1.55 (page 67)
If a language is described by a regular expression, then it is regular.

proof: (Using structural induction)

Recall $R$ is a regular expression if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. . . . So $L$ is regular language.

- Similar arguments for $R = \epsilon, R = \emptyset$; . . .

- If $R = (R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions. Assume $R_1, R_2$ describe regular languages $L_1, L_2$, respectively $(R_1 \cup R_2)$ describes $L_1 \cup L_2$ that is a regular language
1.3 Regular Expressions

**Lemma 1.55** (page 67)

*If a language is described by a regular expression, then it is regular.*

**proof:** *(Using structural induction)*

Recall *R* is a *regular expression* if *R* is one of the followings:

1. a symbol *a* ∈ Σ the alphabet
2. ε
3. ∅
4. (*R*₁ ∪ *R*₂), where *R*₁, *R*₂ are regular expressions
5. (*R*₁ ∩ *R*₂), where *R*₁, *R*₂ are regular expressions
6. (*R*₁*), where *R*₁ is a regular expression.

Let language *L* be described by regular expression *R*.

- If *R* = *a* for some *a* ∈ Σ, we can construct an NFA to recognize exactly strings described by *a*. . . . . So *L* is regular language.

- Similar arguments for *R* = ε, *R* = ∅; . . . .

- If *R* = (*R*₁ ∪ *R*₂), where *R*₁, *R*₂ are regular expressions.
  Assume *R*₁, *R*₂ describe regular languages *L*₁, *L*₂, respectively
  (*R*₁ ∪ *R*₂) describes *L*₁ ∪ *L*₂ that is a regular language by Theorem 1.45.
1.3 Regular Expressions

**Lemma 1.55** (page 67)
*If a language is described by a regular expression, then it is regular.*

**proof:** (Using structural induction)

Recall $R$ is a *regular expression* if $R$ is one of the followings:
1. a symbol $a \in \Sigma$ the alphabet
2. $\epsilon$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions
5. $(R_1 \circ R_2)$, where $R_1, R_2$ are regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.

Let language $L$ be described by regular expression $R$.

- If $R = a$ for some $a \in \Sigma$, we can construct an NFA to recognize exactly strings described by $a$. . . . So $L$ is regular language.

- Similar arguments for $R = \epsilon$, $R = \emptyset$; . . .

- If $R = (R_1 \cup R_2)$, where $R_1, R_2$ are regular expressions.
  Assume $R_1, R_2$ describe regular languages $L_1, L_2$, respectively
  $(R_1 \cup R_2)$ describes $L_1 \cup L_2$ that is a regular language by Theorem 1.45.
  Thus, $R$ describes a regular language.
1.3 Regular Expressions

**Lemma 1.55** (page 67)
*If a language is described by a regular expression, then it is regular.*

**proof:** *(Using structural induction)*

Recall *R* is a *regular expression* if *R* is one of the followings:
1. a symbol *a* ∈ Σ the alphabet
2. ε
3. ∅
4. (*R*₁ ∪ *R*₂), where *R*₁, *R*₂ are regular expressions
5. (*R*₁ ∩ *R*₂), where *R*₁, *R*₂ are regular expressions
6 (*R*₁*), where *R*₁ is a regular expression.

Let language *L* be described by regular expression *R*.

- If *R* = *a* for some *a* ∈ Σ, we can construct an NFA to recognize exactly strings described by *a*. . . . So *L* is regular language.

- Similar arguments for *R* = ε, *R* = ∅; . . .

- If *R* = (*R*₁ ∪ *R*₂), where *R*₁, *R*₂ are regular expressions. Assume *R*₁, *R*₂ describe regular languages *L*₁, *L*₂, respectively. (*R*₁ ∪ *R*₂) describes *L*₁ ∪ *L*₂ that is a regular language by Theorem 1.45. Thus, *R* describes a regular language.

- Similar arguments for cases 5 and 6; . . . by Theorems 1.46 and 1.49.
1.3 Regular Expressions

More examples of building NFAs from regular expressions

Building an NFA for regular expression \((a \cup b)^* aba\), (Figure 1.59, page 69)
to determine how the expression is composed,
then to construct NFAs bottom-up
- NFA for \(a\);  
- NFA for \(b\);  
- NFA for \((a \cup b)\);  
- NFA for \((a \cup b)^*\);  
- NFA for \(ab\);  
- NFA for \(aba\);  
- NFA for \((a \cup b)^* aba\).
1.3 Regular Expressions

More examples of building NFAs from regular expressions

Building an NFA for regular expression

\((a \cup b)^*aba\), (Figure 1.59, page 69)
More examples of building NFAs from regular expressions

Building an NFA for regular expression

\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed,
1.3 Regular Expressions

More examples of building NFAs from regular expressions

Building an NFA for regular expression
\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed,
then to construct NFAs bottom-up
More examples of building NFAs from regular expressions

Building an NFA for regular expression
\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed,
then to construct NFAs bottom-up

● NFA for \(a\);
More examples of building NFAs from regular expressions

Building an NFA for regular expression

\[(a \cup b)^*aba\], (Figure 1.59, page 69)

to determine how the expression is composed, then to construct NFAs bottom-up

- NFA for \(a\);
- NFA for \(b\);
More examples of building NFAs from regular expressions

Building an NFA for regular expression

\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed, then to construct NFAs bottom-up

- NFA for \(a\);
- NFA for \(b\);
- NFA for \((a \cup b)\);
1.3 Regular Expressions

More examples of building NFAs from regular expressions

Building an NFA for regular expression
\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed, then to construct NFAs bottom-up

- NFA for \(a\);
- NFA for \(b\);
- NFA for \((a \cup b)\);
- NFA for \((a \cup b)^*\);
1.3 Regular Expressions

More examples of building NFAs from regular expressions

Building an NFA for regular expression

\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed,
then to construct NFAs bottom-up

- NFA for \(a\);
- NFA for \(b\);
- NFA for \((a \cup b)\);
- NFA for \((a \cup b)^*\);
- NFA for \(ab\);
1.3 Regular Expressions

More examples of building NFAs from regular expressions

Building an NFA for regular expression

\[(a \cup b)^*aba\], (Figure 1.59, page 69)

to determine how the expression is composed, then to construct NFAs bottom-up

- NFA for \(a\);
- NFA for \(b\);
- NFA for \((a \cup b)\);
- NFA for \((a \cup b)^*\);
- NFA for \(ab\);
- NFA for \(aba\);
More examples of building NFAs from regular expressions

Building an NFA for regular expression

\((a \cup b)^*aba\), (Figure 1.59, page 69)

to determine how the expression is composed,
then to construct NFAs bottom-up

- NFA for \(a\);
- NFA for \(b\);
- NFA for \((a \cup b)\);
- NFA for \((a \cup b)^*\);
- NFA for \(ab\);
- NFA for \(aba\);
- NFA for \((a \cup b)^*aba\)
1.3 Regular Expressions

Lemma 1.60

If a language is regular, it is described by a regular expression.

proof idea:

• a regular language is recognized by some DFA;
• convert the DFA to a generalized NFA (GNFA);
• repeatedly simplify the GNFA until it contains only two states;
• the two-state GNFA is actually a regular expression.
1.3 Regular Expressions

How about the other direction?
How about the other direction?

**Lemma 1.60** (page 69)

*If a language is regular, it is described by a regular expression.*
1.3 Regular Expressions

How about the other direction?

**Lemma 1.60** (page 69)

*If a language is regular, it is described by a regular expression.*

**proof idea:**
- a regular language is recognized by some DFA;
1.3 Regular Expressions

How about the other direction?

**Lemma 1.60** (page 69)

*If a language is regular, it is described by a regular expression.*

**proof idea:**

- a regular language is recognized by some DFA;
- convert the DFA to a generalized NFA (GNFA);
1.3 Regular Expressions

How about the other direction?

**Lemma 1.60** (page 69)

*If a language is regular, it is described by a regular expression.*

**proof idea:**

- a regular language is recognized by some DFA;
- convert the DFA to a generalized NFA (GNFA);
- repeatedly simplify the GNFA until it contains only two states
1.3 Regular Expressions

How about the other direction?

Lemma 1.60 (page 69)

> If a language is regular, it is described by a regular expression.

proof idea:

- a regular language is recognized by some DFA;
- convert the DFA to a generalized NFA (GNFA);
- repeatedly simplify the GNFA until it contains only two states
- the two-state GNFA is actually a regular expression
1.3 Regular Expressions

• a single accept state;
• only out-going transitions from start state;
• only incoming transitions to accept state;
• one transition from one state to every other state, and from one to itself, excluding start and accept states;
• start and accept states are not the same.
• transition edges can have regular expressions, instead of single symbol
1.3 Regular Expressions

GNFA
1.3 Regular Expressions

- a single accept state;
1.3 Regular Expressions

- a single accept state;
- only out-going transitions from start state;

GNFA
1.3 Regular Expressions

- a single accept state;
- only out-going transitions from start state;
- only incoming transitions to accept state;
1.3 Regular Expressions

- a single accept state;
- only out-going transitions from start state;
- only incoming transitions to accept state;
- one transition from one state to every other state, and from one to itself, excluding start and accept states;
1.3 Regular Expressions

- a single **accept state**;
- only out-going transitions from **start state**;
- only incoming transitions to **accept state**;
- one transition from one state to every other state, and from one to itself, **excluding start and accept states**;
- **start** and **accept** states are not the same.
1.3 Regular Expressions

- a single accept state;
- only out-going transitions from start state;
- only incoming transitions to accept state;
- one transition from one state to every other state, and from one to itself, excluding start and accept states;
- start and accept states are not the same.
- transition edges can have regular expressions, instead of single symbol
1.3 Regular Expressions

- a single accept state;
- only out-going transitions from start state;
- only incoming transitions to accept state;
- one transition from one state to every other state, and from one to itself, excluding start and accept states;
- start and accept states are not the same.
- transition edges can have regular expressions, instead of single symbol.
1.3 Regular Expressions

- add a new start state \( S_{\text{start}} \);
- add a new accept state \( S_{\text{accept}} \);
- remove state \( S_2 \) and compensate it with regular expression \( 01^*0 \) on a new transition from \( S_1 \) to \( S_1 \);
- on the self-loop, combine \( 1 \) with \( 01^*0 \), resulting in \( 1 \cup 01^*0 \);
- remove state \( S_1 \) and compensate it with regular expression \( \epsilon(1 \cup 01^*0)^*\epsilon \);
- only \( S_{\text{start}} \) and \( S_{\text{accept}} \) remain, with \( \epsilon(1 \cup 01^*0)^*\epsilon \) on the transition;
- simplify \( \epsilon(1 \cup 01^*0)^*\epsilon \) as \( (1 \cup 01^*0)^* \), the desired regular expression.
1.3 Regular Expressions

- add a new start state $S_{\text{start}}$;

![DFA Diagram]

- add a new accept state $S_{\text{accept}}$;
- remove state $S_2$ and compensate it with regular expression $01^*0$ on a new transition from $S_1$ to $S_1$;
- on the self-loop, combine $1$ with $01^*0$, resulting in $1 \cup 01^*0$;
- remove state $S_1$ and compensate it with regular expression $\epsilon(1 \cup 01^*0)^*\epsilon$;
- only $S_{\text{start}}$ and $S_{\text{accept}}$ remain, with $\epsilon(1 \cup 01^*0)^*\epsilon$ on the transition;
- simplify $\epsilon(1 \cup 01^*0)^*\epsilon$ as $(1 \cup 01^*0)^*$, the desired regular expression.
1.3 Regular Expressions

- add a new start state $S'_{\text{start}}$;
- add a new accept state $S'_{\text{accept}}$;

![DFA Diagram](image-url)
1.3 Regular Expressions

- add a new start state $S_{\text{start}}$;
- add a new accept state $S_{\text{accept}}$;
- remove state $S_2$ and compensate it with regular expression $01^*0$
on a new transition from $S_1$ to $S_1$;
1.3 Regular Expressions

- add a new start state $S'_{\text{start}}$;
- add a new accept state $S'_{\text{accept}}$;
- remove state $S_2$ and compensate it with regular expression $01^*0$
  on a new transition from $S_1$ to $S_1$;
- on the self-loop, combine 1 with $01^*0$, resulting in $1 \cup 01^*0$;
1.3 Regular Expressions

- add a new start state $S_{\text{start}}$;
- add a new accept state $S_{\text{accept}}$;
- remove state $S_2$ and compensate it with regular expression $01^*0$
  on a new transition from $S_1$ to $S_1$;
- on the self-loop, combine 1 with $01^*0$, resulting in $1 \cup 01^*0$;
- remove state $S_1$ and compensate it with regular expression $\epsilon(1 \cup 01^*0)^*\epsilon$;
1.3 Regular Expressions

- add a new start state $S'_{\text{start}}$;
- add a new accept state $S'_{\text{accept}}$;
- remove state $S_2$ and compensate it with regular expression $01^*0$ on a new transition from $S_1$ to $S_1$;
- on the self-loop, combine 1 with $01^*0$, resulting in $1 \cup 01^*0$;
- remove state $S_1$ and compensate it with regular expression $\epsilon(1 \cup 01^*0)^*\epsilon$;
- only $S'_{\text{start}}$ and $S'_{\text{accept}}$ remain, with $\epsilon(1 \cup 01^*0)^*\epsilon$ on the transition;
1.3 Regular Expressions

- add a new start state $S_{\text{start}}$;
- add a new accept state $S_{\text{accept}}$;
- remove state $S_2$ and compensate it with regular expression $01^*0$ on a new transition from $S_1$ to $S_1$;
- on the self-loop, combine 1 with $01^*0$, resulting in $1 \cup 01^*0$;
- remove state $S_1$ and compensate it with regular expression $\varepsilon(1 \cup 01^*0)^*\varepsilon$;
- only $S_{\text{start}}$ and $S_{\text{accept}}$ remain, with $\varepsilon(1 \cup 01^*0)^*\varepsilon$ on the transition;
- simplify $\varepsilon(1 \cup 01^*0)^*\varepsilon$ as $(1 \cup 01^*0)^*$, the desired regular expression.
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

$DFA\ M \Rightarrow GNFA$ with $k$ states$
\Rightarrow GNFA$ with $k - 1$ states

$\Rightarrow \ldots$

$\Rightarrow GNFA$ with $2$ states

$\Rightarrow$ regular expression

The critical step is to construct an equivalent GNFA with one fewer state when $k > 2$.

where $q_{rip}$ is removed and compensate for it with regular expression $(R_1)(R_2)^* (R_3) \cup (R_4)$
1.3 Regular Expressions

General steps (given regular language \( L \) that is recognized by DFA \( M \))
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M$
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M \Rightarrow$ GNFA with $k$ states

The critical step is to construct an equivalent GNFA with one fewer state when $k > 2$.

where $q_{\text{rip}}$ is removed and compensate for it with regular expression $(R_1)(R_2)^* (R_3) \cup (R_4)$
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M \Rightarrow$ GNFA with $k$ states $\Rightarrow$ GNFA with $k - 1$ states

The critical step is to construct an equivalent GNFA with one fewer state when $k > 2$.

where $q_{rip}$ is removed and compensate for it with regular expression $(R_1)(R_2)^* (R_3) \cup (R_4)$.
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M \Rightarrow$ GNFA with $k$ states $\Rightarrow$ GNFA with $k - 1$ states $\Rightarrow$ \ldots
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M \Rightarrow$ GNFA with $k$ states $\Rightarrow$ GNFA with $k - 1$ states
$\Rightarrow \ldots$ GNFA with 2 states
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

$$DFA \ M \Rightarrow \ GNFA \ with \ k \ states \Rightarrow \ GNFA \ with \ k - 1 \ states$$

$$\Rightarrow \ldots \ GNFA \ with \ 2 \ states \Rightarrow \ regular \ expression$$
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M \Rightarrow$ GNFA with $k$ states $\Rightarrow$ GNFA with $k - 1$ states

$\Rightarrow \ldots$ GNFA with 2 states $\Rightarrow$ regular expression

The critical step is to construct an equivalent GNFA with one fewer state when $k > 2$. 
1.3 Regular Expressions

General steps (given regular language $L$ that is recognized by DFA $M$)

DFA $M \Rightarrow$ GNFA with $k$ states $\Rightarrow$ GNFA with $k - 1$ states
$\Rightarrow \ldots$ GNFA with 2 states $\Rightarrow$ regular expression

The critical step is to construct an equivalent GNFA with one fewer state when $k > 2$.

\[
\begin{align*}
q_i \xrightarrow{R_1} q_{rip} \xrightarrow{R_2} q_i \quad \text{(before)} \quad \text{and} \quad q_i \xrightarrow{(R_1)(R_2)^* (R_3) \cup (R_4)} q_j \quad \text{(after)}
\end{align*}
\]

**Figure 1.63**
Constructing an equivalent GNFA with one fewer state

where $q_{rip}$ is removed and compensate for it with regular expression

\[(R_1)(R_2)^* (R_3) \cup (R_4)\]
1.3 Regular Expressions

Actually accomplished in the following steps:

• remove $q_{\text{rip}}$ and drop transitions from $q_i$ to $q_{\text{rip}}$ and from $q_{\text{rip}}$ to $q_j$;

• compensate with a transition from $q_i$ to $q_j$ with regular expression

\[(R_1)(R_2)^* (R_3)\];

• combine the two transitions from $q_i$ to $q_j$, resulting in

\[(R_1)(R_2)^* (R_3) \cup (R_4).\]
1.3 Regular Expressions

Actually accomplished in the following steps:

Figure 1.63
Constructing an equivalent GNFA with one fewer state
1.3 Regular Expressions

Actually accomplished in the following steps:

- remove $q_{rip}$ and drop transitions from $q_i$ to $q_{rip}$ and from $q_{rip}$ to $q_j$;

\[
\text{before} \quad q_i \xrightarrow{R_1} q_{rip} \xrightarrow{R_2} q_j \text{ after} \quad q_i \xrightarrow{(R_1)(R_2)^* (R_3) \cup (R_4)} q_j
\]
1.3 Regular Expressions

Actually accomplished in the following steps:

- remove $q_{rip}$ and drop transitions from $q_i$ to $q_{rip}$ and from $q_{rip}$ to $q_j$;
- compensate with a transition from $q_i$ to $q_j$ with regular expression $(R_1)(R_2)^*(R_3)$;
1.3 Regular Expressions

Actually accomplished in the following steps:

- remove $q_{rip}$ and drop transitions from $q_i$ to $q_{rip}$ and from $q_{rip}$ to $q_j$;
- compensate with a transition from $q_i$ to $q_j$ with regular expression $(R_1)(R_2)^*(R_3)$;
- combine the two transitions from $q_i$ to $q_j$, resulting in $(R_1)(R_2)^*(R_3) \cup (R_4)$.
1.3 Regular Expressions
1.3 Regular Expressions

Alternative way to view removal of a state from a GNFA
1.3 Regular Expressions

Alternative way to view removal of a state from a GNFA
1.3 Regular Expressions

Alternative way to view removal of a state from a GNFA
1.3 Regular Expressions

To show that every regular language $L$ can be described with a regular expression, we prove it by construction.

- a formal definition of GNFA;
- an algorithm to derive from GNFA a regular expression that can describe the given language $L$. 
1.3 Regular Expressions

To show that every regular language $L$ can be described with a regular expression,
1.3 Regular Expressions

To show that every regular language $L$ can be described with a regular expression, we prove it by construction.
To show that every regular language $L$ can be described with a regular expression, we prove it \textit{by construction}.

So we need
1.3 Regular Expressions

To show that every regular language $L$ can be described with a regular expression, we prove it by construction.

So we need

- a formal definition of GNFA; and
1.3 Regular Expressions

To show that every regular language $L$ can be described with a regular expression, we prove it by construction.

So we need

- a formal definition of GNFA; and
- an algorithm to derive from GNFA a regular expression that can describe the given language $L$. 
1.3 Regular Expressions

A generalized NFA is a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function: \((Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \rightarrow \mathbb{R}\);
4. \(q_{\text{start}}\) is the start state;
5. \(q_{\text{accept}}\) is the accept state.

where \(\mathbb{R}\) is the set of regular expressions.

The GNFA accepts a string \(w = w_1 w_2 \ldots w_k\), where \(w_i \in \Sigma^*\) if there is a sequence of states \(q_0 q_1 \ldots q_k\) such that

1. \(q_0 = q_{\text{start}}\)
2. \(q_k = q_{\text{accept}}\)
3. \(w_i \in \text{set of strings represented by regular expression } R_i\)

where \(R_i = \delta(q_{i-1}, q_i)\), for all \(i = 1, 2, \ldots, k\).
1.3 Regular Expressions

**Definition 1.64** (page 73)
A generalized NFA is a 5-tuple \((Q, \Sigma, \delta, q_{start}, q_{accept})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function:
   \(\delta: (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow R\);
4. \(q_{start}\) is the start state;
5. \(q_{accept}\) is the accept state.

where \(R\) is the set of regular expressions.

The GNFA accepts a string \(w = w_1w_2...w_k\), where \(w_i \in \Sigma^*\) if there is a sequence of states \(q_0q_1...q_k\) such that

1. \(q_0 = q_{start}\)
2. \(q_k = q_{accept}\)
3. \(w_i \in \text{set of strings represented by regular expression } R_i\), where
   \(R_i = \delta(q_{i-1}, q_i)\), for all \(i = 1, 2, ..., k\).
Definition 1.64 (page 73)
A generalized NFA is a 5-tuple \((Q, \Sigma, \delta, q_{start}, q_{accept})\), where

1. \(Q\) is the finite set of states;
1.3 Regular Expressions

Definition 1.64 (page 73)
A generalized NFA is a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
1.3 Regular Expressions

**Definition 1.64** (page 73)
A *generalized NFA* is a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function: \((Q - \{q_{\text{accept}}\}) \times (Q - \{Q_{\text{start}}\}) \rightarrow \mathcal{R}\);
1.3 Regular Expressions

Definition 1.64 (page 73)
A generalized NFA is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. $Q$ is the finite set of states;
2. $\Sigma$ is the input alphabet;
3. $\delta$ is transition function: $(Q - \{q_{\text{accept}}\}) \times (Q - \{Q_{\text{start}}\}) \rightarrow R$;
4. $q_{\text{start}}$ is the start state;
1.3 Regular Expressions

**Definition 1.64 (page 73)**

A *generalized NFA* is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. $Q$ is the finite set of states;
2. $\Sigma$ is the input alphabet;
3. $\delta$ is transition function: $(Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \to \mathcal{R}$;
4. $q_{\text{start}}$ is the start state;
5. $q_{\text{accept}}$ is the accept state.

where $\mathcal{R}$ is the set of regular expressions.
1.3 Regular Expressions

Definition 1.64 (page 73)
A generalized NFA is a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function: \((Q - \{q_{\text{accept}}\}) \times (Q - \{Q_{\text{start}}\}) \rightarrow \mathcal{R}\);
4. \(q_{\text{start}}\) is the start state;
5. \(q_{\text{accept}}\) is the accept state.

where \(\mathcal{R}\) is the set of regular expressions.

The GNFA accepts a string \(w = w_1w_2 \ldots w_k\), where \(w_i \in \Sigma^*\) if there is a sequence of states \(q_0q_1 \ldots q_k\) such that
1.3 Regular Expressions

**Definition 1.64 (page 73)**
A generalized NFA is a 5-tuple \((Q, \Sigma, \delta, q_{start}, q_{accept})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function: \((Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{R}\);
4. \(q_{start}\) is the start state;
5. \(q_{accept}\) is the accept state.

where \(\mathcal{R}\) is the set of regular expressions.

The GNFA accepts a string \(w = w_1w_2 \ldots w_k\), where \(w_i \in \Sigma^*\) if there is a sequence of states \(q_0q_1 \ldots q_k\) such that
1.3 Regular Expressions

**Definition 1.64 (page 73)**
A *generalized NFA* is a 5-tuple \( (Q, \Sigma, \delta, q_{start}, q_{accept}) \), where

1. \( Q \) is the finite set of states;
2. \( \Sigma \) is the input alphabet;
3. \( \delta \) is transition function: \( (Q - \{q_{accept}\}) \times (Q - \{Q_{start}\}) \rightarrow R \);
4. \( q_{start} \) is the start state;
5. \( q_{accept} \) is the accept state.

where \( R \) is the set of regular expressions.

The GNFA *accepts* a string \( w = w_1w_2 \ldots w_k \), where \( w_i \in \Sigma^* \) if there is a sequence of states \( q_0q_1 \ldots q_k \) such that

\[
(1) \quad q_0 = q_{start}
\]
1.3 Regular Expressions

**Definition 1.64 (page 73)**
A *generalized NFA* is a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function: \((Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \mathcal{R};\)
4. \(q_{\text{start}}\) is the start state;
5. \(q_{\text{accept}}\) is the accept state.

where \(\mathcal{R}\) is the set of regular expressions.

The GNFA *accepts* a string \(w = w_1w_2 \ldots w_k\), where \(w_i \in \Sigma^*\) if there is a sequence of states \(q_0q_1 \ldots q_k\) such that

1. \(q_0 = q_{\text{start}}\)
2. \(q_k = q_{\text{accept}}\)
Definition 1.64 (page 73)
A *generalized NFA* is a 5-tuple \((Q, \Sigma, \delta, q_{start}, q_{accept})\), where

1. \(Q\) is the finite set of states;
2. \(\Sigma\) is the input alphabet;
3. \(\delta\) is transition function: \((Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \mathcal{R}\);
4. \(q_{start}\) is the start state;
5. \(q_{accept}\) is the accept state.

where \(\mathcal{R}\) is the set of regular expressions.

The GNFA *accepts* a string \(w = w_1w_2\ldots w_k\), where \(w_i \in \Sigma^*\) if there is a sequence of states \(q_0q_1\ldots q_k\) such that

1. \(q_0 = q_{start}\)
2. \(q_k = q_{accept}\)
3. \(w_i \in \text{set of strings represented by regular expression } R_i\),
1.3 Regular Expressions

**Definition 1.64 (page 73)**

A *generalized NFA* is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. $Q$ is the finite set of states;
2. $\Sigma$ is the input alphabet;
3. $\delta$ is transition function: $(Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \mathcal{R}$;
4. $q_{\text{start}}$ is the start state;
5. $q_{\text{accept}}$ is the accept state.

where $\mathcal{R}$ is the set of regular expressions.

The GNFA accepts a string $w = w_1w_2 \ldots w_k$, where $w_i \in \Sigma^*$ if there is a sequence of states $q_0q_1 \ldots q_k$ such that

1. $q_0 = q_{\text{start}}$
2. $q_k = q_{\text{accept}}$
3. $w_i \in$ set of strings represented by regular expression $R_i$,
   where $R_i = \delta(q_{i-1}, q_i)$, for all $i = 1, 2, \ldots, k$. 
1.3 Regular Expressions

Formally define this GNFA

- \( Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \} \)
- \( \delta(q_{\text{start}}, q_{\text{up}}) = \text{ab}^* \), \( \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset \), \( \delta(q_{\text{start}}, q_{\text{accept}}) = b \),
- \( \delta(q_{\text{up}}, q_{\text{up}}) = \text{aa} \), \( \delta(q_{\text{up}}, q_{\text{accept}}) = \text{ab} \cup \text{ba} \), \( \delta(q_{\text{up}}, q_{\text{down}}) = \text{a}^* \)

Is string \( w = \text{abbbaaaaabbbbb} \) accepted?

Rewrite \( w = \text{abbbaaaaabbbbb} \)
- \( w_1 = \text{abbb} \in \delta(q_{\text{start}}, q_{\text{up}}) \),
- \( w_2 = \text{aa} \in \delta(q_{\text{up}}, q_{\text{up}}) \),
- \( w_3 = \text{aaa} \in \delta(q_{\text{up}}, q_{\text{down}}) \),
- \( w_4 = \text{bbbbb} \in \delta(q_{\text{down}}, q_{\text{accept}}) \)

Sequence of states \( q_{\text{start}} q_{\text{up}} q_{\text{up}} q_{\text{down}} q_{\text{accept}} \) allows \( w \) to be accepted.
1.3 Regular Expressions

Formally define this GNFA

- $Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$,
1.3 Regular Expressions

Formally define this GNFA

- $Q = \{q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}}\}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$, $\delta(q_{\text{start}}, q_{\text{down}}) = \emptyset$,

Is string $w = \text{abbbaaaaabbbbb}$ accepted?

Rewrite $w = \text{abbbaaaaabbbbb}$

- $w_1 = \text{abbb} \in \delta(q_{\text{start}}, q_{\text{up}})$
- $w_2 = \text{aa} \in \delta(q_{\text{up}}, q_{\text{up}})$
- $w_3 = \text{aaa} \in \delta(q_{\text{up}}, q_{\text{down}})$
- $w_4 = \text{bbbb} \in \delta(q_{\text{down}}, q_{\text{accept}})$

Sequence of states $q_{\text{start}} q_{\text{up}} q_{\text{up}} q_{\text{down}} q_{\text{accept}}$ allows $w$ to be accepted.
1.3 Regular Expressions

Formally define this GNFA

- \( Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \} \)
- \( \delta(q_{\text{start}}, q_{\text{up}}) = ab^* \), \( \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset \), \( \delta(q_{\text{start}}, q_{\text{accept}}) = b \),

Is string \( w = \text{abbbaaaaabbbbb} \) accepted?

Rewrite \( w = \text{abbbaaaaabbbbb} \)

\( w_1 = \text{abbb} \in \delta(q_{\text{start}}, q_{\text{up}}) \),

\( w_2 = \text{aa} \in \delta(q_{\text{up}}, q_{\text{up}}) \),

\( w_3 = \text{aaa} \in \delta(q_{\text{up}}, q_{\text{down}}) \),

\( w_4 = \text{bbbb} \in \delta(q_{\text{down}}, q_{\text{accept}}) \)

Sequence of states \( q_{\text{start}} \rightarrow q_{\text{up}} \rightarrow q_{\text{up}} \rightarrow q_{\text{down}} \rightarrow q_{\text{accept}} \) allows \( w \) to be accepted.
Formally define this GNFA

- $Q = \{q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}}\}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$, $\delta(q_{\text{start}}, q_{\text{down}}) = \emptyset$, $\delta(q_{\text{start}}, q_{\text{accept}}) = b$, $\delta(q_{\text{up}}, q_{\text{up}}) = aa$,
1.3 Regular Expressions

Formally define this GNFA

- \( Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \} \)
- \( \delta(q_{\text{start}}, q_{\text{up}}) = ab^* \), \( \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset \), \( \delta(q_{\text{start}}, q_{\text{accept}}) = b \),
- \( \delta(q_{\text{up}}, q_{\text{up}}) = aa \), \( \delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba \),

Is string \( w = \text{abbbaaaaabbbbb} \) accepted?
1.3 Regular Expressions

Formally define this GNFA

- \( Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \} \)
- \( \delta(q_{\text{start}}, q_{\text{up}}) = ab^* \), \( \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset \), \( \delta(q_{\text{start}}, q_{\text{accept}}) = b \),
- \( \delta(q_{\text{up}}, q_{\text{up}}) = aa \), \( \delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba \), \( \delta(q_{\text{up}}, q_{\text{down}}) = a^* \)

Is string \( w = abbbaaaaabbbbb \) accepted?

Rewrite \( w = abbbaaaaabbbbb \)

- \( w_1 = abbb \in \delta(q_{\text{start}}, q_{\text{up}}) \)
- \( w_2 = aa \in \delta(q_{\text{up}}, q_{\text{up}}) \)
- \( w_3 = aaa \in \delta(q_{\text{up}}, q_{\text{down}}) \)
- \( w_4 = bbbb \in \delta(q_{\text{down}}, q_{\text{accept}}) \)

Sequence of states \( q_{\text{start}} q_{\text{up}} q_{\text{up}} q_{\text{down}} q_{\text{accept}} \) allows \( w \) to be accepted.
1.3 Regular Expressions

Formally define this GNFA

- \( Q = \{ q_{start}, q_{accept}, q_{up}, q_{down} \} \)
- \( \delta(q_{start}, q_{up}) = ab^* \), \( \delta(q_{start}, q_{down}) = \emptyset \), \( \delta(q_{start}, q_{accept}) = b \),
  \( \delta(q_{up}, q_{up}) = aa \), \( \delta(q_{up}, q_{accept}) = ab \cup ba \), \( \delta(q_{up}, q_{down}) = a^* \)

\ldots
1.3 Regular Expressions

Formally define this GNFA

- $Q = \{q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}}\}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$, $\delta(q_{\text{start}}, q_{\text{down}}) = \emptyset$, $\delta(q_{\text{start}}, q_{\text{accept}}) = b$, $\delta(q_{\text{up}}, q_{\text{up}}) = aa$, $\delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba$, $\delta(q_{\text{up}}, q_{\text{down}}) = a^*$

Is string $w = abbaaaaaabbbbbbb$ accepted?
1.3 Regular Expressions

Formally define this GNFA

- \( Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \} \)
- \( \delta(q_{\text{start}}, q_{\text{up}}) = ab^* \), \( \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset \), \( \delta(q_{\text{start}}, q_{\text{accept}}) = b \), \( \delta(q_{\text{up}}, q_{\text{up}}) = aa \), \( \delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba \), \( \delta(q_{\text{up}}, q_{\text{down}}) = a^* \)

Is string \( w = abbbaaaaaabbbbbb \) accepted? Rewrite \( w = abbbaaaaaabbbbbb \)

\( w_1 = abbb \in \delta(q_{\text{start}}, q_{\text{up}}) \),
Formally define this GNFA

- $Q = \{ q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}} \}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$, $\delta(q_{\text{start}}, q_{\text{down}}) = \emptyset$, $\delta(q_{\text{start}}, q_{\text{accept}}) = b$
- $\delta(q_{\text{up}}, q_{\text{up}}) = aa$, $\delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba$, $\delta(q_{\text{up}}, q_{\text{down}}) = a^*$

Is string $w = abbbaaaaabbbbb$ accepted? Rewrite $w = abbbaaaaaabbbbb$

$w_1 = abbb \in \delta(q_{\text{start}}, q_{\text{up}})$, $w_2 = aa \in \delta(q_{\text{up}}, q_{\text{up}})$,
Formally define this GNFA

\[ Q = \{q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}}\} \]

\[ \delta(q_{\text{start}}, q_{\text{up}}) = ab^*, \quad \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset, \quad \delta(q_{\text{start}}, q_{\text{accept}}) = b, \]

\[ \delta(q_{\text{up}}, q_{\text{up}}) = aa, \quad \delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba, \quad \delta(q_{\text{up}}, q_{\text{down}}) = a^* \]

Is string \( w = abbbaaaaabbbbb \) accepted?

Rewrite \( w = abbbaaaaaaabbbbb \)

\( w_1 = abbb \in \delta(q_{\text{start}}, q_{\text{up}}), \quad w_2 = aa \in \delta(q_{\text{up}}, q_{\text{up}}), \quad w_3 = aaa \in \delta(q_{\text{up}}, q_{\text{down}}), \)
1.3 Regular Expressions

Formally define this GNFA

- $Q = \{q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}}\}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$, $\delta(q_{\text{start}}, q_{\text{down}}) = \emptyset$, $\delta(q_{\text{start}}, q_{\text{accept}}) = b$
- $\delta(q_{\text{up}}, q_{\text{up}}) = aa$, $\delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba$, $\delta(q_{\text{up}}, q_{\text{down}}) = a^*$

Is string $w = abbbaaaaabbbba$ accepted? Rewrite $w = abbbaaaaabbbba$

$w_1 = abbb \in \delta(q_{\text{start}}, q_{\text{up}})$, $w_2 = aa \in \delta(q_{\text{up}}, q_{\text{up}})$, $w_3 = aaaa \in \delta(q_{\text{up}}, q_{\text{down}})$, $w_4 = bbbba \in \delta(q_{\text{down}}, q_{\text{accept}})$
1.3 Regular Expressions

Formally define this GNFA

- $Q = \{q_{\text{start}}, q_{\text{accept}}, q_{\text{up}}, q_{\text{down}}\}$
- $\delta(q_{\text{start}}, q_{\text{up}}) = ab^*$, \hspace{1em} \delta(q_{\text{start}}, q_{\text{down}}) = \emptyset$, \hspace{1em} $\delta(q_{\text{start}}, q_{\text{accept}}) = b$,
- $\delta(q_{\text{up}}, q_{\text{up}}) = aa$, \hspace{1em} $\delta(q_{\text{up}}, q_{\text{accept}}) = ab \cup ba$, \hspace{1em} $\delta(q_{\text{up}}, q_{\text{down}}) = a^*$

Is string $w = abbbaaaaabbbbabbbbb$ accepted? Rewrite $w = abbbaaaaabbbbabbbbb$

- $w_1 = abbb \in \delta(q_{\text{start}}, q_{\text{up}})$, \hspace{1em} $w_2 = aa \in \delta(q_{\text{up}}, q_{\text{up}})$, \hspace{1em} $w_3 = aaa \in \delta(q_{\text{up}}, q_{\text{down}})$, \hspace{1em} $w_4 = bbbbb \in \delta(q_{\text{down}}, q_{\text{accept}})$

Sequence of states $q_{\text{start}}q_{\text{up}}q_{\text{up}}q_{\text{down}}q_{\text{accept}}$ allows $w$ to be accepted.
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- Add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$.
- Call the procedure $\text{Convert}(G)$ recursively.

$\text{Convert}(G)$:

1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{\text{start}}$ to $q_{\text{accept}}$ with $R$ return $(R)$ and stop.
3. Select a state $q_{\text{rip}}$, for every pair of states $q_i, q_j$ that are not $q_{\text{rip}}$, or start, or accept state, set $\delta'(q_i, q_j) = \delta_1 \delta_2 \ast \delta_3 \cup \delta_4$, where $\delta_1 = \delta(q_i, q_{\text{rip}})$, $\delta_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $\delta_3 = \delta(q_{\text{rip}}, q_j)$, $\delta_4 = \delta(q_i, q_j)$. $\delta'$ is the same as $\delta$ for the remaining parts.
4. Let $G' = (Q - \{q_{\text{rip}}\}, \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$.
5. Call $\text{Convert}(G')$. 
1.3 Regular Expressions

**Algorithm** to convert DFA $M$ to a 2-state GNFA.
1.3 Regular Expressions

**Algorithm** to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure \textsc{Convert}(G) recursively.

\textsc{Convert}(G):
1. Let $k$ be the number of states in $G$;
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.

$\text{CONVERT}(G)$:

1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{\text{start}}$ to $q_{\text{accept}}$ with $R$
   return $(R)$ and stop.
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.

$\text{CONVERT}(G)$:
1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{\text{start}}$ to $q_{\text{accept}}$ with $R$
   \hspace{1cm} return $\langle R \rangle$ and stop.
3. Select a state $q_{\text{rip}}$, for every pair of states $q_i, q_j$ that
   \hspace{1cm} are not $q_{\text{rip}}$, or start, or accept state,
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.

$\text{CONVERT}(G)$:

1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{\text{start}}$ to $q_{\text{accept}}$ with $R$
   return $(R)$ and stop.
3. Select a state $q_{\text{rip}}$, for every pair of states $q_i, q_j$ that
   are not $q_{\text{rip}}$, or start, or accept state,
   set $\delta'(q_i, q_j) = R_1 R_2 \ast R_3 \cup R_4$,
1.3 Regular Expressions

**Algorithm** to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.

$\text{CONVERT}(G)$:

1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{\text{start}}$ to $q_{\text{accept}}$ with $R$
   return $(R)$ and stop.
3. Select a state $q_{\text{rip}}$, for every pair of states $q_i, q_j$ that
   are not $q_{\text{rip}}$, or start, or accept state,
   set $\delta'(q_i, q_j) = R_1 R_2 * R_3 \cup R_4$,
   where $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$,
   $R_3 = \delta(q_{\text{rip}}, q_j)$, $R_4 = \delta(q_i, q_j)$
1.3 Regular Expressions

**Algorithm** to convert DFA $M$ to a 2-state GNFA.

- add $q_{start}$ and $q_{accept}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.

$\text{CONVERT}(G)$:
1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{start}$ to $q_{accept}$ with $R$
   \hspace{1cm} return $(R)$ and stop.
3. Select a state $q_{rip}$, for every pair of states $q_i, q_j$ that
   \hspace{1cm} are not $q_{rip}$, or start, or accept state,
   \hspace{1cm} set $\delta'(q_i, q_j) = R_1 R_2 \ast R_3 \cup R_4$,
   \hspace{1cm} where $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$,
   \hspace{1.2cm} $R_3 = \delta(q_{rip}, q_j)$, $R_4 = \delta(q_i, q_j)$
   \hspace{1cm} $\delta'$ is the same as $\delta$ for the remaining parts
1.3 Regular Expressions

Algorithm to convert DFA $M$ to a 2-state GNFA.

- add $q_{\text{start}}$ and $q_{\text{accept}}$ to $M$, resulting in GNFA $G$
- call the procedure $\text{CONVERT}(G)$ recursively.

$\text{CONVERT}(G)$:

1. Let $k$ be the number of states in $G$;
2. If $k = 2$, a single transition from $q_{\text{start}}$ to $q_{\text{accept}}$ with $R$
   return $(R)$ and stop.
3. Select a state $q_{\text{rip}}$, for every pair of states $q_i, q_j$ that
   are not $q_{\text{rip}}$, or start, or accept state,
   set $\delta'(q_i, q_j) = R_1 R_2 \ast R_3 \cup R_4$,
   where $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$,
   $R_3 = \delta(q_{\text{rip}}, q_j)$, $R_4 = \delta(q_i, q_j)$
   $\delta'$ is the same as $\delta$ for the remaining parts
4. Let $G' = (Q - \{q_{\text{rip}}\}, \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$
5. Call $\text{CONVERT}(G')$. 
1.3 Regular Expressions

Claim: for any \( G \), \( \text{Convert}(G) \) returns a regular expression that describes the language \( L \).

Proof: We prove the claim by induction on \( k \), the number of states in \( G \):

**basis:** \( k = 2 \). The GNAF is equivalent to the DFA \( M \); the regular expression returned by \( G \) describes \( L \). The claim is true.

**assumption:** \( \text{Convert}(G) \) returns a regular expression that describes language \( L \) when \( G \) has \( k - 1 \) states.

**induction:** \( G \) has \( k \) states. We show that \( G' \), the GNFA as the result of removing some state \( q \) from \( G \), recognizes the same language as \( G \) does. By assumption, \( \text{Convert}(G') \) returns a regular expression that describes language \( L \); thus, \( \text{Convert}(G) \) returns a regular expression that describes language \( L \). The claim is proved.
Claim: for any $G$, $\text{Convert}(G)$ returns a regular expression that describes the language $L$. 
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:
1.3 Regular Expressions

**Claim**: for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof**: We prove the claim by induction on $k$, the number of states in $G$:

**basis**: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** $\text{CONVERT}(G)$ returns a regular expression that describes language $L$ when $G$ has $k - 1$ states.
1.3 Regular Expressions

**Claim**: for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof**: We prove the claim by induction on $k$, the number of states in $G$:

- **basis**: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

- **assumption**: $\text{CONVERT}(G)$ returns a regular expression that describes language $L$ when $G$ has $k - 1$ states.

- **induction**: $G$ has $k$ states. We show that $G'$, the GNFA as the result of removing some state $q_{rip}$ from $G$, recognizes the same language as $G$ does.
1.3 Regular Expressions

**Claim:** for any $G$, $\text{Convert}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** $\text{Convert}(G)$ returns a regular expression that describes language $L$ when $G$ has $k - 1$ states.

**induction:** $G$ has $k$ states. We show that $G'$, the GNFA as the result of removing some state $q_{\text{rip}}$ from $G$, recognizes the same language as $G$ does.
1.3 Regular Expressions

Claim: for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

Proof: We prove the claim by induction on $k$, the number of states in $G$:

basis: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

assumption: $\text{CONVERT}(G)$ returns a regular expression that describes language $L$ when $G$ has $k-1$ states.

induction: $G$ has $k$ states. We show that $G'$, the GNFA as the result of removing some state $q_{rip}$ from $G$, recognizes the same language as $G$ does.

By assumption, $\text{CONVERT}(G')$ returns a regular expression that describes language $L$;
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

*Proof:* We prove the claim by induction on $k$, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** $\text{CONVERT}(G)$ returns a regular expression that describes language $L$ when $G$ has $k - 1$ states.

**induction:** $G$ has $k$ states. We show that $G'$, the GNFA as the result of removing some state $q_{\text{rip}}$ from $G$, recognizes the same language as $G$ does.

By assumption, $\text{CONVERT}(G')$ returns a regular expression that describes language $L$;
thus, $\text{CONVERT}(G)$ returns a regular expression that describes language $L$. The claim is proved.
1.3 Regular Expressions

Claim: for any $G$, $\text{Convert}(G)$ returns a regular expression that describes the language $L$.

Proof: We prove the claim by induction on $k$, the number of states in $G$:

basis: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

assumption: the claim is true when $G$ has $k - 1$ states.

induction: $G$ has $k$ states. We show removing state $q_{rip}$ still allows the recognition of the same language. Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{start}, q_1, q_2, ..., q_{accept}$.

Case 1. $q_{rip}$ does not occur in the path. The claim is true.

Case 2. $q_{rip}$ occurred in the path. Let $q_i = q_{rip}$; $\text{Convert}(G)$ step 3 guarantees $w$ can be accepted by the reduced GNFA $G'$ of $k - 1$ states, with the path $q_{start}, q_1, ..., q_{i-1}, q_{i+1}, ..., q_{accept}$. So $G$ and $G'$ accepts the same set of strings. By assumption, $\text{Convert}(G')$ returns a regular expression that describes language $L$; thus, $\text{Convert}(G)$ returns a regular expression that describes language $L$. The claim is proved.
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$. 

Proof: We prove the claim by induction on $k$, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption**: the claim is true when $G$ has $k−1$ states.

**induction**: $G$ has $k$ states. We show removing state $q_{rip}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$-state GNFA $G$, with path $q_{start}, q_1, q_2, ..., q_{accept}$

**Case 1.** $q_{rip}$ does not occur in the path. The claim is true.

**Case 2.** $q_{rip}$ occurred in the path. Let $q_i = q_{rip}$.

$\text{CONVERT}(G)$ step 3 guarantees $w$ can be accepted by the reduced GNFA $G'$ of $k−1$ states, with the path $q_{start}, q_1, ..., q_i−1, q_i+1, ..., q_{accept}$.

So $G$ and $G'$ accepts the same set of strings. By assumption, $\text{CONVERT}(G')$ returns a regular expression that describes language $L$; thus, $\text{CONVERT}(G)$ returns a regular expression that describes language $L$. The claim is proved.
1.3 Regular Expressions

Claim: for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

Proof: We prove the claim by induction on $k$, the number of states in $G$:
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:

- **basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

- **induction:** $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.

  - **Case 1.** $q_{\text{rip}}$ does not occur in the path. The claim is true.

  - **Case 2.** $q_{\text{rip}}$ occurred in the path. Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$.

    - **step 3** guarantees $w$ can be accepted by the reduced GNFA $G'$ of $k - 1$ states, with the path $q_{\text{start}}, q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_{\text{accept}}$.

    So $G$ and $G'$ accepts the same set of strings. By assumption, $\text{CONVERT}(G')$ returns a regular expression that describes language $L$; thus, $\text{CONVERT}(G)$ returns a regular expression that describes language $L$. The claim is proved.
1.3 Regular Expressions

Claim: for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

Proof: We prove the claim by induction on $k$, the number of states in $G$:

basis: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

assumption: the claim is true when $G$ has $k - 1$ states.
1.3 Regular Expressions

**Claim**: for any $G$, \textsc{Convert}(G) returns a regular expression that describes the language $L$.

**Proof**: We prove the claim by induction on $k$, the number of states in $G$:

**basis**: $k = 2$. The GNFA is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption**: the claim is true when $G$ has $k - 1$ states.

**induction**: $G$ has $k$ states. We show removing state $q_{rip}$ still allows the recognition of the same language.
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by **induction on** $k$, the number of states in $G$:

**basis:** $k = 2$. The GNFA is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** the claim is true when $G$ has $k - 1$ states.

**induction:** $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by **induction on $k$**, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** the claim is true when $G$ has $k - 1$ states.

**induction:** $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$

**Case 1.** $q_{\text{rip}}$ does not occur in the path. The claim is true.
1.3 Regular Expressions

Claim: for any $G$, \textsc{Convert}(G) returns a regular expression that describes the language $L$.

Proof: We prove the claim by induction on $k$, the number of states in $G$:

basis: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

assumption: the claim is true when $G$ has $k - 1$ states.

induction: $G$ has $k$ states. We show removing state $q_{rip}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{start}, q_1, q_2, \ldots, q_{accept}$

Case 1. $q_{rip}$ does not occur in the path. The claim is true.

Case 2. $q_{rip}$ occurred in the path.
1.3 Regular Expressions

Claim: for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

Proof: We prove the claim by induction on $k$, the number of states in $G$:

basis: $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

assumption: the claim is true when $G$ has $k - 1$ states.

induction: $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$

Case 1. $q_{\text{rip}}$ does not occur in the path. The claim is true.

Case 2. $q_{\text{rip}}$ occurred in the path. Let $q_i = q_{\text{rip}}$
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** the claim is true when $G$ has $k - 1$ states.

**induction:** $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$

*Case 1.* $q_{\text{rip}}$ does not occur in the path. The claim is true.

*Case 2.* $q_{\text{rip}}$ occurred in the path. Let $q_i = q_{\text{rip}}$

$\text{CONVERT}(G')$ step 3 guarantees $w$ can be accepted by the reduced GNFA $G'$ of $k - 1$ states, with the path $q_{\text{start}}, q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_{\text{accept}}$
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:

- **basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

- **assumption:** the claim is true when $G$ has $k - 1$ states.

- **induction:** $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.
  
  Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \cdots, q_{\text{accept}}$

  **Case 1.** $q_{\text{rip}}$ does not occur in the path. The claim is true.

  **Case 2.** $q_{\text{rip}}$ occurred in the path. Let $q_i = q_{\text{rip}}$

  $\text{CONVERT}(G')$ step 3 guarantees $w$ can be accepted by the reduced GNFA $G'$ of $k - 1$ states, with the path $q_{\text{start}}, q_1, \cdots, q_{i-1}, q_{i+1}, \cdots, q_{\text{accept}}$

  So $G$ and $G'$ accepts the same set of strings
1.3 Regular Expressions

Claim: for any \( G \), \( \text{CONVERT}(G) \) returns a regular expression that describes the language \( L \).

Proof: We prove the claim by induction on \( k \), the number of states in \( G \):

- **basis**: \( k = 2 \). The GNAF is equivalent to the DFA \( M \); the regular expression returned by \( G \) describes \( L \). The claim is true.

- **assumption**: the claim is true when \( G \) has \( k - 1 \) states.

- **induction**: \( G \) has \( k \) states. We show removing state \( q_{\text{rip}} \) still allows the recognition of the same language.

  Let \( w \) be accepted by the \( k \) state GNFA \( G \), with path \( q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}} \)

  - **Case 1.** \( q_{\text{rip}} \) does not occur in the path. The claim is true.
  - **Case 2.** \( q_{\text{rip}} \) occurred in the path. Let \( q_i = q_{\text{rip}} \)

    \( \text{CONVERT}(G) \) step 3 guarantees \( w \) can be accepted by the reduced GNFA \( G' \) of \( k - 1 \) states, with the path \( q_{\text{start}}, q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_{\text{accept}} \)

  So \( G \) and \( G' \) accepts the same set of strings.

  By assumption, \( \text{CONVERT}(G') \) returns a regular expression that describes language \( L \);
1.3 Regular Expressions

**Claim:** for any $G$, $\text{CONVERT}(G)$ returns a regular expression that describes the language $L$.

**Proof:** We prove the claim by induction on $k$, the number of states in $G$:

**basis:** $k = 2$. The GNAF is equivalent to the DFA $M$; the regular expression returned by $G$ describes $L$. The claim is true.

**assumption:** the claim is true when $G$ has $k - 1$ states.

**induction:** $G$ has $k$ states. We show removing state $q_{\text{rip}}$ still allows the recognition of the same language.

Let $w$ be accepted by the $k$ state GNFA $G$, with path $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$

**Case 1.** $q_{\text{rip}}$ does not occur in the path. The claim is true.

**Case 2.** $q_{\text{rip}}$ occurred in the path. Let $q_i = q_{\text{rip}}$

$\text{CONVERT}(G')$ step 3 guarantees $w$ can be accepted by the reduced GNFA $G'$ of $k - 1$ states, with the path $q_{\text{start}}, q_1, \ldots, q_{i-1}, q_i+1, \ldots, q_{\text{accept}}$

So $G$ and $G'$ accepts the same set of strings. By assumption, $\text{CONVERT}(G')$ returns a regular expression that describes language $L$; thus, $\text{CONVERT}(G)$ returns a regular expression that describes language $L$. The claim is proved.
1.3 Regular Expressions

Examples:
• Convert the 2-state DFA to an equivalent regular expression.
• Convert the 3-state DFA to a regular expression.
1.3 Regular Expressions

Examples:
1.3 Regular Expressions

Examples:

- Convert the 2-state DFA to an equivalent regular expression.
1.3 Regular Expressions

Examples:

- Convert the 2-state DFA to an equivalent regular expression.

![2-state DFA diagram](image)

- Convert the 3-state DFA to an equivalent regular expression.

![3-state DFA diagram](image)
1.4 Non-Regular Languages

To answer the following related questions:
• How powerful is a DFA (NFA)?
• What problems may (not) be defined as regular languages?
• What makes an FA (not) powerful?
• What languages cannot be recognized by FAs?
Non-regular languages

1.4 Non-Regular Languages
1.4 Non-Regular Languages

Non regular languages

To answer the following related questions:
1.4 Non-Regular Languages

Non regular languages

To answer the following related questions:

• How powerful is a DFA (NFA)?
1.4 Non-Regular Languages

Non regular languages

To answer the following related questions:

- How powerful is a DFA (NFA)?
- What problems may (not) be defined as regular languages?
1.4 Non-Regular Languages

Non regular languages

To answer the following related questions:

- How powerful is a DFA (NFA)?
- What problems may (not) be defined as regular languages?
- What makes an FA (not) powerful?
1.4 Non-Regular Languages

Non regular languages

To answer the following related questions:

- How powerful is a DFA (NFA) ?
- What problems may (not) be defined as regular languages?
- What makes an FA (not) powerful?
- What languages cannot be recognized by FAs?
1.4 Non-Regular Languages

Revisit the case of matching parentheses in arithmetic expressions. FAs do not seem to be able to do matching.

More examples:

$L_1 = \{ w : w \text{ contains the same number of } 1 \text{ and } 0 \}$

$L_2 = \{ 0^k 1^k : k \geq 0 \}$

$L_3 = \{ w : w \in \Sigma^* \text{ and } w \text{ is a palindrome} \}$

If FAs cannot recognize these languages, they are not regular languages.

What makes these languages non-regular?
1.4 Non-Regular Languages

Revisit the case of matching parentheses in arithmetic expressions
1.4 Non-Regular Languages

Revisit the case of matching parentheses in arithmetic expressions

FAs do not seem to be able to do matching
1.4 Non-Regular Languages

Revisit the case of matching parentheses in arithmetic expressions

FAs do not seem to be able to do matching

More examples:

$L_1 = \{w : w \text{ contains the same number of 1 and 0}\}$. 
1.4 Non-Regular Languages

Revisit the case of matching parentheses in arithmetic expressions

FAs do not seem to be able to do matching

More examples:

$L_1 = \{ w : w \text{ contains the same number of 1 and 0} \}.$

$L_2 = \{ 0^k 1^k : k \geq 0 \}$
Revisit the case of matching parentheses in arithmetic expressions

FAs do not seem to be able to do matching

More examples:

$L_1 = \{w : w \text{ contains the same number of 1 and 0}\}$.

$L_2 = \{0^k1^k : k \geq 0\}$

$L_3 = \{w : w \in \Sigma^* \text{ and } w \text{ is a palindrome}\}$

If FAs cannot recognize these languages, they are not regular languages.
Revisit the case of matching parentheses in arithmetic expressions

FAs do not seem to be able to do matching

More examples:

$L_1 = \{ w : w \text{ contains the same number of 1 and 0} \}.$

$L_2 = \{ 0^k 1^k : k \geq 0 \}$

$L_3 = \{ w : w \in \Sigma^* \text{ and } w \text{ is a palindrome} \}$

If FAs cannot recognize these languages, they are not regular languages.

What makes these languages non-regular?
1.4 Non-Regular Languages

Consider $L_2 = \{0^k 1^k : k \geq 0\}$.
Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:

{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^5 1^5$.
$w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, ..., q_{10}$ because $|w| = 10$.
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
{blue, black, green, red, pink, gray, violet, orange}
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states:
\( \{\text{blue, black, green, red, pink, gray, violet, orange}\} \)

Also assume string \( w = 0^51^5 \).
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10.$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: {blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$$w = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$$w = \begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}$$
$q_0$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k 1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
{$\text{blue, black, green, red, pink, gray, violet, orange}$}

Also assume string $w = 0^5 1^5$. $w$ is accepted by $M$ with 
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[w = 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\]

$q_0 \quad q_1$
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of \( 11 \) states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
\begin{align*}
w &= 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\
q_0 \ q_1 \ q_2 \ q_3
\end{align*}
\]
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 

{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$w = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5$
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states:
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$$w = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1$$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
$\{\text{blue, black, green, red, pink, gray, violet, orange}\}$

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with 
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$w = 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1$

\[q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\]
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{align*}
w &= 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
q_0 & q_1 \; q_2 \; q_3 \; q_4 \; q_5 \; q_6 \; q_7 \; q_8 \; q_9
\end{align*}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\begin{align*}
w &= 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \\
q_0 & \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \quad q_9 \quad q_{10}
\end{align*}
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \( |Q| = 8 \) states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \( |w| = 10 \).

\[
\begin{align*}
  w & = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\
  q_0 & q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \\
  q_0 &
\end{align*}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
w = \begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10}
\end{array}
\]

\[
q_0 \quad q_1
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
{
blue, black, green, red, pink, gray, violet, orange
}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$w = 0 
0 
0 
0 
0 
1 
1 
1 
1 
1$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0 \ q_1 \ q_2$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$$w = \begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10}
\end{array}$$

$$\begin{array}{cccc}
q_0 & q_1 & q_2 & q_3
\end{array}$$
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \( |Q| = 8 \) states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with 
a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \( |w| = 10 \).

\[
\begin{align*}
    w &= 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\
    q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_{10} \\
    q_0 \ q_1 \ q_2 \ q_3 & \qquad q_4
\end{align*}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
\{	ext{blue, black, green, red, pink, gray, violet, orange}\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$. 

$$w = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 $$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ $
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0 \} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with 
\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8 \rightarrow q_9 \rightarrow q_{10} \]

\[ w = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \]
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: \{\text{blue, black, green, red, pink, gray, violet, orange}\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{array}{cccccccccccc}
\text{w} & = & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 \\
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
\begin{align*}
  w &= 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\
  q_0 & q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \\
  q_0 & q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8
\end{align*}
\]
1.4 Non-Regular Languages

Consider \( L_2 = \{ 0^k1^k : k \geq 0 \} \)

Assume some NFA \( M \) recognizes the language. It has \( |Q| = 8 \) states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with 
a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \( |w| = 10 \).

\[
\begin{array}{ccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9
\end{array}
\]
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{array}{cccccccccc}
& 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states:
$\{\text{blue, black, green, red, pink, gray, violet, orange}\}$

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$$w = \begin{array}{ccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array}$$

$$q_0 \quad q_0 \quad q_0 \quad q_0 \quad q_0 \quad q_0 \quad q_0 \quad q_0 \quad q_0$$
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
\begin{array}{ccccccccccc}
& 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 \\
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k 1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^5 1^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$. 

$w = \begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 
\end{array}$
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \( |Q| = 8 \) states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \( |w| = 10 \).

\[
\begin{array}{cccccccccc}
| & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\downarrow & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
\downarrow & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
\downarrow & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
\begin{array}{cccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 &  &  &  &  &  & \\
\end{array}
\]
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states:
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with
a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{align*}
w &= 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \\
q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \\
q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \\
q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5
\end{align*}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
{blue, black, green, red, pink, gray, violet, orange}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with 
a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q \\
\end{array}
\]
1.4 Non-Regular Languages

Consider \( L_2 = \{ 0^k1^k : k \geq 0 \} \)

Assume some NFA \( M \) recognizes the language. It has \(|Q| = 8\) states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with 
a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \(|w| = 10\).

\[
\begin{array}{cccccccc}
  w &=& 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
  q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
  q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
  q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q & q & q_7
\end{array}
\]
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \( |Q| = 8 \) states:
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \( |w| = 10 \).

\[
\begin{align*}
q_0 & \quad q_1 & \quad q_2 & \quad q_3 & \quad q_4 & \quad q_5 & \quad q_6 & \quad q_7 & \quad q_8 & \quad q_9 & \quad q_{10} \\
q_0 & \quad q_1 & \quad q_2 & \quad q_3 & \quad q_4 & \quad q_5 & \quad q_6 & \quad q_7 & \quad q_8 & \quad q_9 & \quad q_{10} \\
q_0 & \quad q_1 & \quad q_2 & \quad q_3 & \quad q_4 & \quad q_5 & \quad q & \quad q_7 & \quad q
\end{align*}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: 
\{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q & q_7 & q & q_9
\end{array}
\]
1.4 Non-Regular Languages

Consider $L_2 = \{0^k1^k : k \geq 0\}$

Assume some NFA $M$ recognizes the language. It has $|Q| = 8$ states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string $w = 0^51^5$. $w$ is accepted by $M$ with a sequence of 11 states $q_0, q_1, \ldots, q_{10}$ because $|w| = 10$.

$w = \begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q & q_7 & q & q_{10}
\end{array}$
1.4 Non-Regular Languages

Consider \( L_2 = \{0^k1^k : k \geq 0\} \)

Assume some NFA \( M \) recognizes the language. It has \( |Q| = 8 \) states: \{blue, black, green, red, pink, gray, violet, orange\}

Also assume string \( w = 0^51^5 \). \( w \) is accepted by \( M \) with a sequence of 11 states \( q_0, q_1, \ldots, q_{10} \) because \( |w| = 10 \).

\[
\begin{align*}
&0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
& q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{align*}
\]

A portion of NFA \( M \)
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8 \rightarrow 000001111$ 

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8 \rightarrow q_9 \rightarrow 0000011111$

More accept paths?

By the way, can you give a regular expression for these strings?

So $M$ accepts strings like $w' = 0517, 0519, 0513$, etc.

$M$ does not exactly recognize $L_2$. Contradicts.
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. 
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0 \rightarrow q_1$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0 \quad q_1 \quad q_2$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$  $q_1$  $q_2$  $q_3$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5$
From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$  $q_5$  $q_6$  $q_7$  $q_8$
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}$
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \ 0000011111 \]
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$  $q_5$  $q_6$  $q_7$  $q_8$  $q_9$  $q_{10}$  0000011111

$q_0$
From this portion of $M$, we can derive **some other strings accepted by $M$**. We find accept paths:

$$q_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \quad q_9 \quad q_{10} \quad 00000111111$$

$$q_0 \quad q_1$$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \quad q_9 \quad q_{10} \quad 0000011111$

$q_0 \quad q_1 \quad q_2 \quad q_3$
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$
From this portion of $M$, we can derive \textit{some other strings accepted by} $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_9 \ q_10 \ 0000011111$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \ 0000011111$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_7$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$. We find accept paths:

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$  $q_5$  $q_6$  $q_7$  $q_8$  $q_9$  $q_{10}$  0000011111

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$  $q_5$  $q_6$  $q_7$  $q_7$  $q_8$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$\begin{align*}
q_0 & \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_9 \rightarrow q_{10} \rightarrow 0000011111 \\
q_0 & \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_7 \rightarrow q_8 \rightarrow q_9
\end{align*}$
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$  $q_5$  $q_6$  $q_7$  $q_8$  $q_9$  $q_{10}$  0000011111

$q_0$  $q_1$  $q_2$  $q_3$  $q_4$  $q_5$  $q_6$  $q_7$  $q_7$  $q_8$  $q_9$  $q_{10}$
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_7$ $q_8$ $q_9$ $q_{10}$ 000001111111
From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_9\ q_{10}\ 0000011111$

$q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_7\ q_8\ q_9\ q_{10}\ 000001111111$

More accept paths?
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$$q_0 \ x \ q_2 \ x \ q_3 \ x \ q_4 \ x \ q_5 \ x \ q_6 \ x \ q_7 \ x \ q_8 \ x \ q_9 \ x \ q_{10} \ x \ 0000011111$$

$$q_0 \ x \ q_2 \ x \ q_3 \ x \ q_4 \ x \ q_5 \ x \ q_6 \ x \ q_7 \ x \ q_7 \ x \ q_8 \ x \ q_9 \ x \ q_{10} \ x \ 000001111111$$

More accept paths?

By the way, can you give a regular expression for these strings?
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \quad q_9 \quad q_{10} \quad 0000011111$

$q_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \quad q_7 \quad q_8 \quad q_9 \quad q_{10} \quad 000001111111$

More accept paths?

By the way, can you give a regular expression for these strings?

So $M$ accepts strings like $w' = 0^51^7$, $0^51^9$, $0^51^3$, etc.
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$ 0000011111

$q_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_7$ $q_8$ $q_9$ $q_{10}$ 000001111111

More accept paths?

By the way, can you give a regular expression for these strings?

So $M$ accepts strings like $w' = 0^51^7$, $0^51^9$, $0^51^3$, etc.

$M$ does not exactly recognize $L_2$. Contradicts.
1.4 Non-Regular Languages

From this portion of $M$, we can derive some other strings accepted by $M$.

We find accept paths:

\[
\begin{array}{cccccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} & 0000011111 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} & 0000011111
\end{array}
\]

More accept paths?

By the way, can you give a regular expression for these strings?

So $M$ accepts strings like $w' = 0^51^7$, $0^51^9$, $0^51^3$, etc.

$M$ does not exactly recognize $L_2$. Contradicts.
1.4 Non-Regular Languages

We conclude the following:

1. If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$; the accept path for $w$ in $M$ contains a pair of same state.

2. The substring on the transitions between this pair of states can be skipped or repeated multiple times; (pumped).

3. Strings as result of pumping are also accepted by $M$.

4. However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We conclude the following

• If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$, the accept path for $w$ in $M$ contains a pair of same state.

• The substring on the transitions between this pair of states can be skipped or repeated multiple times; (pumped).

• Strings as result of pumping are also accepted by $M$.

• However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
- The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped");
- Strings as result of pumping are also accepted by $M$;
- However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We conclude the following

• If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;

  the accept path for $w$ in $M$ contains a pair of same state.
We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
  the accept path for $w$ in $M$ contains a pair of same state.
- The substring on the transitions between this pair of states can be skipped or repeated multiple times;
- Strings as result of pumping are also accepted by $M$.
- However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
  
  the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be skipped or repeated multiple times; (“pumped”).

- Strings as result of pumping are also accepted by $M$.

- However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$, the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

000001
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
  the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be **skipped or repeated multiple times**; ("pumped").

  00000111
1.4 Non-Regular Languages

We conclude the following

- If \( M \) recognizes language \( L \) and accepts a string \( w \in L \), where \( |w| \geq |Q| \);
  the accept path for \( w \) in \( M \) contains a pair of same state.
- The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

  0000011111
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
  the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be \textit{skipped or repeated multiple times}; ("pumped").

- $000001111111$
1.4 Non-Regular Languages

We conclude the following

• If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
  the accept path for $w$ in $M$ contains a pair of same state.

• The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

  0000011111111
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$; the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

00000111111111111
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$; the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

  0000011111111111

- Strings as result of pumping are also accepted by $M$. 
1.4 Non-Regular Languages

We conclude the following

- If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$;
  the accept path for $w$ in $M$ contains a pair of same state.

- The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

  000001111111111

- Strings as result of pumping are also accepted by $M$.

- However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We conclude the following

• If $M$ recognizes language $L$ and accepts a string $w \in L$, where $|w| \geq |Q|$; the accept path for $w$ in $M$ contains a pair of same state.

• The substring on the transitions between this pair of states can be skipped or repeated multiple times; ("pumped").

  0000011111111111

• Strings as result of pumping are also accepted by $M$.

• However, if we can show pumped strings are NOT desired by $L$, we show $M$ does NOT recognize exactly $L$, a contradiction.
1.4 Non-Regular Languages

We consider another example $L_1 = \{ w \mid w \text{ contains the same number of 1's and 0's} \}$. Is $L_1$ regular? We now try to use the same trick to disprove.

- Assume NFA $M$ recognizes $L_1$.
- Pick a string $w \in L_1$ which has length $\geq |Q|$.
- $M$ accepts $w$ with a path of states, including a pair of same states.
- We hope $w$ can be pumped, resulted in $w'$, such that
  1. the $w'$ is also accepted by $M$,
  2. but $w'$ does not contain the same number of 0's and 1's.

However, statement (2) may not always be true.
1.4 Non-Regular Languages

We consider another example

\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's}\}. \]
1.4 Non-Regular Languages

We consider another example

\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \} \]

Is \( L_1 \) regular?
1.4 Non-Regular Languages

We consider another example

\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \} \]

Is \( L_1 \) regular? We now try to use the same trick to disprove.
1.4 Non-Regular Languages

We consider another example
\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \} \].

Is \( L_1 \) regular? We now try to use the same trick to disprove.

- Assume NFA \( M \) recognizes \( L_1 \).

\[ w \in L_1 \text{ which has length } \geq |Q|, \]
\[ M \text{ accepts } w \text{ with a path of states, including a pair of same states} \]

We hope \( w \) can be pumped, resulted in \( w' \), such that

1. \( w' \) is also accepted by \( M \),
2. but \( w' \) does not contain the same number of 0's and 1's

However, statement (2) may not always be true.
1.4 Non-Regular Languages

We consider another example

$L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \}$.

Is $L_1$ regular? We now try to use the same trick to disprove.

- Assume NFA $M$ recognizes $L_1$.
- Pick a string $w \in L_1$ which has length $\geq |Q|$,

However, statement (2) may not always be true.
1.4 Non-Regular Languages

We consider another example

\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \} \]

Is \( L_1 \) regular? We now try to use the same trick to disprove.

- Assume NFA \( M \) recognizes \( L_1 \).
- Pick a string \( w \in L_1 \) which has length \( \geq |Q| \),
- \( M \) accepts \( w \) with a path of states, including a pair of same states.
1.4 Non-Regular Languages

We consider another example
\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \} \]

Is \( L_1 \) regular? We now try to use the same trick to disprove.

- Assume NFA \( M \) recognizes \( L_1 \).
- Pick a string \( w \in L_1 \) which has length \( \geq |Q| \),
- \( M \) accepts \( w \) with a path of states, including a pair of same states
- We hope \( w \) can be pumped, resulted in \( w' \), such that

  (1) the \( w' \) is also accepted by \( M \),
  (2) but \( w' \) does not contain the same number of 0's and 1's

However, statement (2) may not always be true
1.4 Non-Regular Languages

We consider another example
\[ L_1 = \{ w : w \text{ contains the same number of 1's and 0's} \} \].

Is \( L_1 \) regular? We now try to use the same trick to disprove.

- Assume NFA \( M \) recognizes \( L_1 \).
- Pick a string \( w \in L_1 \) which has length \( \geq |Q| \),
- \( M \) accepts \( w \) with a path of states, including a pair of same states
- We hope \( w \) can be pumped, resulted in \( w' \), such that
  (1) the \( w' \) is also accepted by \( M \),
1.4 Non-Regular Languages

We consider another example

\[ L_1 = \{ w : \text{w contains the same number of 1's and 0's} \} \]

Is \( L_1 \) regular? We now try to use the same trick to disprove.

- Assume NFA \( M \) recognizes \( L_1 \).
- Pick a string \( w \in L_1 \) which has length \( \geq |Q| \),
- \( M \) accepts \( w \) with a path of states, including a pair of same states
- We hope \( w \) can be pumped, resulted in \( w' \), such that

1. the \( w' \) is also accepted by \( M \),
2. but \( w' \) does not contain the same number of 0's and 1's
1.4 Non-Regular Languages

We consider another example
\[ L_1 = \{w : w \text{ contains the same number of 1’s and 0’s}\} \].

Is \( L_1 \) regular? We now try to use the same trick to disprove.

- Assume NFA \( M \) recognizes \( L_1 \).
- Pick a string \( w \in L_1 \) which has length \( \geq |Q| \),
- \( M \) accepts \( w \) with a path of states, including a pair of same states
- We hope \( w \) can be pumped, resulted in \( w' \), such that
  
  (1) the \( w' \) is also accepted by \( M \),
  
  (2) but \( w' \) does not contain the same number of 0’s and 1’s

However, statement (2) may not always be true
1.4 Non-Regular Languages

conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

Conclusion: $M$ accepts strings $001(1010)^*$ \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10,$

$w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10,$

\[ w = \begin{array}{cccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
q_0 & & & & & & & 1
\end{array} \]
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10$,

$w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$

$q_0 \ q_1$

Conclusion: $M$ accepts strings $001010101010101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \]
1.4 Non-Regular Languages

\[ |Q| = 8, \quad |w| = 10, \]

\[ w = \begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3
\end{array} \]

Conclusion: \( M \) accepts strings \( 001(1010) \) \( \subseteq L \), No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

$w = \begin{array}{cccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4
\end{array}$

Conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \]

Conclusion: \( M \) accepts strings \( 001(1010)^* 101 \) \( \subseteq L \), no contradiction!
1.4 Non-Regular Languages

|Q| = 8, |w| = 10,

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \]

\[
\text{conclusion: } M \text{ accepts strings } 0011010101101 \subseteq L_1, \text{ No contradiction!}
\]
1.4 Non-Regular Languages

\[ |Q| = 8, \; |w| = 10, \]

\[ w = 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \]

\[ q_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \]

Conclusion: \( M \) accepts strings \( 001(1010)^*101 \) \( \subseteq L_1 \), No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

\[
w = \begin{array}{ccccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8
\end{array}
\]

Conclusion: $M$ accepts strings $001(1010)^{\ast}101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \]

Conclusion: \( M \) accepts strings \( 001(1010) \). No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

$w = \begin{array}{ccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}$
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

$w = \begin{array}{cccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}$

Conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L$, No contradiction!
1.4 Non-Regular Languages

\[|Q| = 8, \ |w| = 10,\]

\[w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1\]

\[q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}\]

\[q_0 \ q_1\]
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10$,

$w = \begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2
\end{array}$

conclusion: $M$ accepts strings $001(1010)^*101$ $\subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$$|Q| = 8, \ |w| = 10,$$

$$w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$$

$$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$$

$$q_0 \ q_1 \ q_2 \ q_3$$

conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = \begin{array}{ccccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 \end{array} \]

Conclusion: \( M \) accepts strings \( 001(1010)^*101 \subseteq L_1 \), No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \]

Conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10,$

$w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6$
1.4 Non-Regular Languages

\[ |Q| = 8, \; |w| = 10, \]

\[ w = 0 \; 0 \; 1 \; 1 \; 0 \; 1 \; 0 \; 1 \; 0 \; 1 \]

q_0 \; q_1 \; q_2 \; q_3 \; q_4 \; q_5 \; q_6 \; q_7 \; q_8 \; q_9 \; q_{10}

q_0 \; q_1 \; q_2 \; q_3 \; q_4 \; q_5 \; q_6 \; q_7
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[
\begin{array}{cccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array}
\]
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10$,

$$w = \begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
\end{array}$$

$\conclusion: M$ accepts strings $0011010101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

$w = \ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1$

$q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_9\ q_{10}$

Conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10,$

$w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0$
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

\[ w = 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ \]

\[ q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_9\ q_{10} \]

\[ q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ q_9\ q_{10} \]

\[ q_0\ q_1 \]

conclusion: $M$ accepts strings $001(1010)^*101 \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10$, 

$$w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10}$

$q_0 \ q_1 \ q_2$
1.4 Non-Regular Languages

$|Q| = 8$, $|w| = 10$,

$w = \begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q \\
\end{array}$

Conclusion: $M$ accepts strings $001(1010) \subseteq L_1$, No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \quad |w| = 10,$

\[
w = \begin{array}{ccccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
q_0 & q_1 & q_2 & q & q_4
\end{array}
\]
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

Conclusion: \( M \) accepts strings \( 001(1010)^*101 \subseteq L_1 \), No contradiction!
1.4 Non-Regular Languages

\[ |Q| = 8, |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

\[ q_0 \ q_1 \ q_2 \ q \ q_4 \ q_5 \ q_6 \]

Conclusion:
\[ M \text{ accepts strings } 001(1010)^* 101 \subseteq L_1, \text{ No contradiction!} \]
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

- \( q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \)
- \( q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \)
- \( q_0 \ q_1 \ q_2 \ q \ q_4 \ q_5 \ q_6 \ q \)
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = \begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array} \]

\[ q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \]
\[ q_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8 \]

Conclusion: \( M \) accepts strings \( 001(1010)^*101 \) \( \subseteq L \), No contradiction!
1.4 Non-Regular Languages

$|Q| = 8, \ |w| = 10,$

\[
\begin{array}{ccccccccccc}
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \quad
\end{array}
\]
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

Conclusion: \( M \) accepts strings \( 001(1010)^*101 \subseteq L_1 \), No contradiction!
1.4 Non-Regular Languages

\[ |Q| = 8, \ |w| = 10, \]

\[ w = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

\[ q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]

\[ q_0 \ q_1 \ q_2 \ q \ q_4 \ q_5 \ q_6 \ q \ q_8 \ q_9 \ q_{10} \]

Conclusion: \( M \) accepts strings 001(1010)**101
1.4 Non-Regular Languages

|Q| = 8, |w| = 10,

w = 0 0 1 1 0 1 0 1 0 1

q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10
q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10
q0 q1 q2 q q4 q5 q6 q q8 q9 q10

conclusion: \( M \) accepts strings 001(1010)*101 \( \subseteq L_1 \), No contradiction!
We will have to choose a different word, but any word in $L_1$ may seem difficult. Strings described by regular expression $(111010001010)^*$ are all in $L_1$. What should we do? Actually, the previous trick should apply not just the whole string $w$ but to any long substring as well!
1.4 Non-Regular Languages

We will have to choose a different word, but any word in $L_1$ may seem difficult

0101011010010101010100010111010001010101011110
1.4 Non-Regular Languages

We will have to choose a different word, but any word in $L_1$ may seem difficult

```
01010110100101010100010111010001010101011110
01010110100101010100010111010001010101011110
```

```
01010110100101010100010111010001010101011110
01010110100101010100010111010001010101011110
```
1.4 Non-Regular Languages

We will have to choose a different word, but any word in $L_1$ may seem difficult

\[
\begin{align*}
0101011010010101010100010111010001010101011110 \\
0101011010010101010100010111010001010101011110
\end{align*}
\]

strings described by regular expression

\[
010101101001010101010100010111010001010101011110
\]

are all in $L_1$. 
We will have to choose a different word, but any word in $L_1$ may seem difficult.

$$0101011010010101010100010111010001010101011110$$
$$0101011010010101010100010(111010001010)\ast 111010001010101011110$$

strings described by regular expression
$$0101011010010101010101010100010(111010001010)\ast 101011110$$

are all in $L_1$.

What should we do?
1.4 Non-Regular Languages

We will have to choose a different word, but any word in $L_1$ may seem difficult

\[01010110100101010100010111010001010101011110\]
\[01010110100101010100010111010001010101011110\]

strings described by regular expression

\[01010110100101010100010111010001010101011110(111010001010)^*101011110\]

are all in $L_1$.

What should we do?

Actually, the previous trick should apply not just the whole string $w$ but to any long substring as well!
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0's and 1's). and substring $1111111111$ in $w$ (of length 10).

Therefore, strings described by regular expression $0000000000111(1111)^*$ are also in $L_1$! But numbers of 1's and 0's are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).
and substring $1111111111$ in $w$ (of length 10)

Therefore, strings described by regular expression
$000000000111(1111)^*111$
are also in $L_1$.
But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

1 1 1 1 1 1 1 1 1 1
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000011111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

1 1 1 1 1 1 1 1 1 1

Therefore, strings described by regular expression $000000000111(1111)^\ast 111$ are also in $L_1$!
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{ccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1
\end{array}
\]
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 000000000011111111111111111$ ∈ $L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

1 1 1 1 1 1 1 1 1 1

$p_0$ $q_1$ $q_2$
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3
\end{array}
\]

Therefore, strings described by regular expression $0000000000111(1111)^*111$ are also in $L_1$!
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4
\end{array}$

Therefore, strings described by regular expression

$000000000111(1111)^*111$

are also in $L_1$.

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{ccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 \\
\end{array}
\]
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 0000000000111111111111111111111111\in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6
\end{array}
\]

Therefore, strings described by regular expression $0000000000111(1111)^*111$ are also in $L_1$.

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[ p_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \]
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$ 
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[ \begin{array}{ccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8
\end{array} \]

Therefore, strings described by regular expression

00000000001111111111

are also in $L_1$!

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring $1111111111$ in $w$ (of length 10)

\[
p_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9
\]

Therefore, strings described by regular expression $000000000111(1111)^{*}111$ are also in $L_1$.
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{ccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10}
\end{array}
\]

Therefore, strings described by regular expression $000000000111(1111) \ast 111$ are also in $L_1$.
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
Assume $|Q| = 8$.
Consider string $w = 000000000111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{align*}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 &
\end{align*}
\]
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000011111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
0 & q_1
\end{array}
\]

Therefore, strings described by regular expression $000000000111(1111)^*111$ are also in $L_1$!
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000011111111111$ ∈ $L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

```
1 1 1 1 1 1 1 1 1 1
p0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10
p0 q1 q2
```
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

1 1 1 1 1 1 1 1 1 1
$p_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$

$p_0$ $q_1$ $q_2$ $q_3$

Therefore, strings described by regular expression $000000000111(1111)^*111$ are also in $L_1$.
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 0000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$$
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array}
$$

Therefore, strings described by regular expression $000000000111111^{*} \in L_1$ are also in $L_1$.

But numbers of 1’s and 0’s are different. Contradicts. So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_{1}$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5
\end{array}
\]

Therefore, strings described by regular expression

$0000000000111(1111)^*111$

are also in $L_{1}$

But numbers of 1’s and 0’s are different, Contradicts.

So $L_{1}$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6
\end{array}
\]

Therefore, strings described by regular expression $0000000000111(1111)^*111$ are also in $L_1$! But numbers of 1’s and 0’s are different, Contradicts. So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & & & \\
\end{array}
\]

Therefore, strings described by regular expression

\[
0000000000111(1111) \ast 111
\]

are also in $L_1$!

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 000000000011111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

1 1 1 1 1 1 1 1 1 1
$p_0$ $q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$ $q_7$ $q_8$ $q_9$ $q_{10}$

Therefore, strings described by regular expression
00000000001111(1111)$^*$111
are also in $L_1$!

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111 \in L_1$ (because it contains the same number of 0's and 1's).
and substring $1111111111$ in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9
\end{array}
\]

Therefore, strings described by regular expression $000000000111(1111)^*111$ are also in $L_1$.
But numbers of 1's and 0's are different, \textit{contradicts}.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 0000000001111111111$ ∈ $L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$$
\begin{array}{ccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array}
$$

Therefore, strings described by regular expression $000000000111(1111)^*$ are also in $L_1$!
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000011111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
 p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
 p_0 & \\
\end{array}
\]

Therefore, strings described by regular expression

$000000000111(1111)^*111$

are also in $L_1$.

But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000011111111111 \in L_1$ (because it contains the same number of 0's and 1's).

and substring 1111111111 in $w$ (of length 10)

\begin{align*}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{align*}

Therefore, strings described by regular expression $000000000111(1111)^*111$ are also in $L_1$!

But numbers of 1's and 0's are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume \( |Q| = 8 \).
Consider string \( w = 0000000000111111111 \in L_1 \) (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in \( w \) (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10}
\end{array}
\]

Therefore, strings described by regular expression

\[
0000000000111(1111)\]

are also in \( L_1 \)!
But numbers of 1’s and 0’s are different, Contradicts.
So \( L_1 \) is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 000000000111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$$
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q \\
\end{array}
$$

Therefore, strings described by regular expression $000000000111(1111)^\ast$ are also in $L_1$! But numbers of 1’s and 0’s are different, Contradicts. So $L_1$ is not regular language.
Assume $|Q| = 8$.
Consider string $w = 00000000011111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4
\end{array}
\]

Therefore, strings described by regular expression

$0000000001111(1111) \ast 111$

are also in $L_1$.

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

Therefore, strings described by regular expression $0000000000111(1111)^*$ are also in $L_1$

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 00000000011111111111 \in L_1$ (because it contains the same number of 0's and 1's).

and substring 1111111111 in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 \\
\end{array}
\]

Therefore, strings described by regular expression $00000000011(1111)^*111$ are also in $L_1$.

But numbers of 1's and 0's are different, Contradicts. So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$$p_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6 \quad q_7 \quad q_8 \quad q_9 \quad q_{10}$$

Therefore, strings described by regular expression $0000000000111(1111)\ast111$ are also in $L_1$! But numbers of 1’s and 0’s are different, Contradicts. So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 000000000011111111111 \in L_1$
(because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\begin{align*}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8
\end{align*}

Therefore, strings described by regular expression $0000000000111(1111)^*111$
are also in $L_1$!
But numbers of 1’s and 0’s are different, Contradicts.
So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.

Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

\[ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \]
\[ p_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]
\[ p_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 \ q_{10} \]
\[ p_0 \ q_1 \ q_2 \q \ q_4 \ q_5 \ q_6 \ q \ q_8 \ q_9 \]

Therefore, strings described by regular expression $0000000000111(1111)^*111$ are also in $L_1$.

But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0's and 1's).

and substring $1111111111$ in $w$ (of length 10)

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8 & q_9 & q_{10} \\
\end{array}
\]

Therefore, strings described by regular expression $0000000000111(1111)^*111$ are also in $L_1$!

But numbers of 1's and 0's are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 00000000011111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$\begin{array}{ccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8 & q_9 & q_{10}
\end{array}$

Therefore, strings described by regular expression

$0000000000111111111(1111)^*111$

are also in $L_1$!
1.4 Non-Regular Languages

Assume $|Q| = 8$.
Consider string $w = 000000000011111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

Therefore, strings described by regular expression

$$0000000000111(1111)^*111$$

are also in $L_1$! But numbers of 1’s and 0’s are different, Contradicts.
### 1.4 Non-Regular Languages

Assume $|Q| = 8$. Consider string $w = 00000000001111111111 \in L_1$ (because it contains the same number of 0’s and 1’s).

and substring 1111111111 in $w$ (of length 10)

$$\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8 & q_9 & q_{10} \\
p_0 & q_1 & q_2 & q & q_4 & q_5 & q_6 & q & q_8 & q_9 & q_{10} \\
\end{array}$$

Therefore, strings described by regular expression

$$00000000001111(1111)^*111$$

are also in $L_1$! But numbers of 1’s and 0’s are different, Contradicts.

So $L_1$ is not regular language.
1.4 Non-Regular Languages

• The trick we used takes advantage of an important property of regular languages.

• Such a property is formally summarized into a Pumping Lemma (for regular languages).

• Once we define and prove the Pumping lemma, we can use it e.g., to prove some languages are not regular languages.
1.4 Non-Regular Languages

- The trick we used takes advantage of an important property of regular languages.
1.4 Non-Regular Languages

- The trick we used takes advantage of an important property of regular languages.
- Such a property is formally summarized into a Pumping Lemma (for regular languages).
1.4 Non-Regular Languages

- The trick we used takes advantage of an important property of regular languages.
- Such a property is formally summarized into a Pumping Lemma (for regular languages).
- Once we define and prove the Pumping lemma, we can use it.
1.4 Non-Regular Languages

- The trick we used takes advantage of an important property of regular languages.
- Such a property is formally summarized into a Pumping Lemma (for regular languages).
- Once we define and prove the Pumping lemma, we can use it e.g., to prove some languages are not regular languages.
1.4 Non-Regular Languages

The Pumping Lemma For Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)

If \( A \) is a regular language, then there is a number \( p \) (called pumping length) where, for any string \( s \in A \) of \(|s| \geq p\), \( s \) can be written as

\[ s = xyz \]

such that

1. for each \( i \geq 0 \), \( xy^i z \in A \),
2. \(|y| > 0\), and
3. \(|xy| \leq p\),

Note: string \( y^i \) is concatenation of \( i \) copies of \( y \)'s.
The Pumping Lemma For Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xy^iz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$
The Pumping Lemma For Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$, 

Note: string $y^i$ is concatenation of $i$ copies of $y$'s
The Pumping Lemma For Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
1.4 Non-Regular Languages

The Pumping Lemma For Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**

If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$
1.4 Non-Regular Languages

The Pumping Lemma For Regular Languages

**Theorem 1.70 (Pumping lemma)*** (page 78)
If $A$ is a regular language, then there is a number $p$ (called **pumping length**) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

Note: string $y^i$ is concatenation of $i$ copies of $y$'s
1.4 Non-Regular Languages

Pumping lemma illustrated:

\[ s \in L \]

\[ s = xyz \in L \]

\[ xy^3z \in L \]
1.4 Non-Regular Languages

Pumping lemma illustrated:

\[ s = 010101001000001001001001010010010010 \]

\[ s \in L \]
1.4 Non-Regular Languages

Pumping lemma illustrated:

\[ s \in L \]

\[ s = \overbrace{01010100100000100100100101010010}^{= p} \]

\[ s = \overbrace{01010100100000100100100101010010}^{= p} \]

\[ q \quad q \]
1.4 Non-Regular Languages

Pumping lemma illustrated:

\[ s = 0101010010000010010010101010101010010 \]

\[ s \in L \]

\[ s = 0101010010010010101010101010010 \]

\[ s \in L \]

\[ s = x y z \in L \]
1.4 Non-Regular Languages

Pumping lemma illustrated:

\[ s \in L \]

\[ s = \textbf{xyz} \in L \]

\[ xy^3z \in L \]
Theorem 1.70 (Pumping lemma) (page 78)

If \( A \) is a regular language, then there is a number \( p \) (called pumping length) where, for any string \( s \in A \) of \(|s| \geq p\), \( s \) can be written as \( s = xyz \), such that

1. for each \( i \geq 0 \), \( xy^iz \in A \),
2. \(|y| > 0\), and
3. \(|xy| \leq p\).

Proof idea

• Assume some DFA accepts language \( A \), \( p = |Q| \);
• \(|s| \geq p \), at least \( p + 1 \) states in the accepting sequence
• the first \( p + 1 \) states must contain a repetition (Pigeonhole principle)
• name the substring "between" repetition of the state
• denote substring before \( y \) with \( x \) and substring after \( y \) with \( z \)
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**proof idea**
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**proof idea**
- assume some DFA accepts language $A$, $p = |Q|$;
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**

If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**proof idea**

- assume some DFA accepts language $A$, $p = |Q|$;
- $|s| \geq p$, at least $p + 1$ states in the accepting sequence
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**proof idea**

- assume some DFA accepts language $A$, $p = |Q|$;
- $|s| \geq p$, at least $p + 1$ states in the accepting sequence
- the first $p + 1$ states must contain a repetition (Pigeonhole principle)
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**proof idea**

- assume some DFA accepts language $A$, $p = |Q|$;
- $|s| \geq p$, at least $p + 1$ states in the accepting sequence
- the first $p + 1$ states must contain a repetition (**Pigeonhole principle**)
- name the substring “between” repetition of the state $y$
1.4 Non-Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

proof idea

- assume some DFA accepts language $A$, $p = |Q|$;
- $|s| \geq p$, at least $p + 1$ states in the accepting sequence
- the first $p + 1$ states must contain a repetition (Pigeonhole principle)
- name the substring “between” repetition of the state $y$
- denote substring before $y$ with $x$ and substring after $y$ with $z$
1.4 Non-Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)

If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$.

Let $s = s_1s_2...s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$.

Let $q_0, q_1, ..., q_n$ be the states on the path to accept $s$.

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$.

$s_1s_2...s_js_{j+1}...s_l...s_p...s_n$ $q_0$ $q_1$...$q_j-1$ $q_j$...$q_l-1$ $q_l$...$q_n$ $M$ accepts $xy^iz$ for every $i \geq 0$; (proved by induction, later)

(2) $|y| > 0$ because $j < l$;

(3) $|xy| \leq p$ because $xy = s_1...s_l$, $l \leq p$. 
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**

If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**

If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**
Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

Proof:
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. 

1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$
1.4 Non-Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

Proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1 s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$.

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$.

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$. 
Theorem 1.70 (Pumping lemma) (page 78)
If \( A \) is a regular language, then there is a number \( p \) (called pumping length) where, for any string \( s \in A \) of \(|s| \geq p\), \( s \) can be written as \( s = xyz \), such that

1. for each \( i \geq 0 \), \( xy^i z \in A \),
2. \(|y| > 0\), and
3. \(|xy| \leq p\)

Proof:

Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA that recognizes \( A \) and \( p = |Q| \). Let \( s = s_1s_2 \ldots s_n \), \( n \geq p \); where \( s_i \in \Sigma \), \( 1 \leq i \leq n \)

Let \( q_0, q_1, \ldots, q_n \) be the states on the path to accept \( s \)

Among the first \( p + 1 \) states, two must be the same: \( q_j = q_l, j < l \leq p \)
1.4 Non-Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

Proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$$
1.4 Non-Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

Proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|.$ Let $s = s_1s_2\ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$
Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

\[
\begin{array}{cccccccc}
    s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
    q_0 & q_1 & \ldots & q_j & q_{j+1} & \ldots & q_l & \ldots & q_p & \ldots & q_n \\
\end{array}
\]
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$s_1$ $s_2$ $\ldots$ $s_j$ $s_{j+1}$ $\ldots$ $s_l$ $\ldots$ $s_p$ $\ldots$ $s_n$
$q_0$ $q_1$ $\ldots$ $q_{j-1}$ $q_j$ $\ldots$ $q_{l-1}$ $q_l$
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**

If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$$
\begin{array}{cccccccccc}
    s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
    q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n
\end{array}
$$
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$
Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

\[
\begin{array}{cccccccccccc}
& s_1 & s_2 & \cdots & s_j & s_{j+1} & \cdots & s_l & \cdots & s_p & \cdots & s_n \\
q_0 & q_1 & \cdots & q_{j-1} & q_j & \cdots & q_{l-1} & q_l & \cdots & q_p & \cdots & q_n \\
\end{array}
\]

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$. 
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1 s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$
Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$$
\begin{array}{cccccccccc}
  s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
  q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n \\
\end{array}
$$

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

So (1) $M$ accepts $xy^i z$ for every $i \geq 0$;
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$
$q_0 \ q_1 \ \ldots \ q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n$

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

So (1) $M$ accepts $xy^iz$ for every $i \geq 0$; *(proved by induction, later)*
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1 s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$
Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$
$q_0 \ q_1 \ \ldots \ q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n$

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

So (1) $M$ accepts $xy^i z$ for every $i \geq 0$; (proved by induction, later)
(2) $|y| > 0$
1.4 Non-Regular Languages

Theorem 1.70 (Pumping lemma) (page 78)
If \( A \) is a regular language, then there is a number \( p \) (called pumping length) where, for any string \( s \in A \) of \( |s| \geq p \), \( s \) can be written as \( s = xyz \), such that

1. for each \( i \geq 0 \), \( xy^i z \in A \),
2. \( |y| > 0 \), and
3. \( |xy| \leq p \)

Proof:
Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA that recognizes \( A \) and \( p = |Q| \).
Let \( s = s_1 s_2 \ldots s_n \), \( n \geq p \); where \( s_i \in \Sigma, 1 \leq i \leq n \)

Let \( q_0, q_1, \ldots, q_n \) be the states on the path to accept \( s \)
Among the first \( p + 1 \) states, two must be the same: \( q_j = q_l, j < l \leq p \)

\[
\begin{align*}
& s_1 \ s_2 \ \cdots \ s_j \ s_{j+1} \ \cdots \ s_l \ \cdots \ s_p \ \cdots \ s_n \\
& q_0 \ q_1 \ \cdots \ q_{j-1} \ q_j \ \cdots \ q_{l-1} \ q_l \ \cdots \ q_p \ \cdots \ q_n
\end{align*}
\]

Let \( x = s_1 \ldots s_j \), \( y = s_{j+1} \ldots s_l \), and \( z = s_{l+1} \ldots s_n \).

So (1) \( M \) accepts \( xy^i z \) for every \( i \geq 0 \); (proved by induction, later)
(2) \( |y| > 0 \) because \( j < l \);
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$\begin{align*}
s_1 & \ s_2 & \ldots & \ s_j & \ s_{j+1} & \ldots & \ s_l & \ldots & \ s_p & \ldots & \ s_n \\
q_0 & \ q_1 & \ldots & \ q_{j–1} & \ q_j & \ldots & \ q_{l–1} & \ q_l & \ldots & \ q_p & \ldots & \ q_n \\
\end{align*}$

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

So (1) $M$ accepts $xy^iz$ for every $i \geq 0$; (proved by induction, later)
(2) $|y| > 0$ because $j < l$; (3) $|xy| \leq p$
1.4 Non-Regular Languages

**Theorem 1.70 (Pumping lemma) (page 78)**
If $A$ is a regular language, then there is a number $p$ (called pumping length) where, for any string $s \in A$ of $|s| \geq p$, $s$ can be written as $s = xyz$, such that

1. for each $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$

**Proof:**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes $A$ and $p = |Q|$. Let $s = s_1s_2 \ldots s_n$, $n \geq p$; where $s_i \in \Sigma$, $1 \leq i \leq n$

Let $q_0, q_1, \ldots, q_n$ be the states on the path to accept $s$

Among the first $p + 1$ states, two must be the same: $q_j = q_l$, $j < l \leq p$

$s_1 \ s_2 \ \cdots \ s_j \ s_{j+1} \ \cdots \ s_l \ \cdots \ s_p \ \cdots \ s_n$
$q_0 \ q_1 \ \cdots \ q_{j-1} \ q_j \ \cdots \ q_{l-1} \ q_l \ \cdots \ q_p \ \cdots \ q_n$

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

So (1) $M$ accepts $xy^i z$ for every $i \geq 0$; (proved by induction, later)
(2) $|y| > 0$ because $j < l$; (3) $|xy| \leq p$ because $xy = s_1 \ldots s_l$, $l \leq p$. 
To prove (1), \( M \) accepts \( xy^i z \) for every \( i \geq 0 \), we restate it as

Claim: For every \( i \geq 0 \), there is an accept path for \( xy^i z \) in which substring \( y^i \) invokes transitions from \( q_j \) to the same state \( q_l \).

Proof (by induction on \( i \))

Basis \( i = 0 \), \( xy^0 z = xz \) that is accepted with path \( q_0 q_1 \ldots q_{j-1} q_j q_{j+1} \ldots q_p \ldots q_n \). The claim is true.

Basis \( i = 1 \), \( xy^1 z = xyz \) accepted with path \( q_0 q_1 \ldots q_{j-1} q_j q_{j+1} \ldots q_p \ldots q_n \). The claim is true.

Assumption: assume that for \( i = k \geq 1 \), the claim is true.

Induction: \( xy^{k+1} z = xy^k yz \); by the assumption, substring \( y^k \) invokes transitions from \( y_j \) to \( y_l \), the following \( y \) will invoke transitions from \( y_l \) (which is \( y_j \)) to \( y_l \), and continue to the accept state \( q_n \). So the claim is proved.
1.4 Non-Regular Languages

To prove (1) \( M \) accepts \( xy^i z \) for every \( i \geq 0 \),
1.4 Non-Regular Languages

To prove (1) $M$ accepts $x y^i z$ for every $i \geq 0$, we restate it as:

Claim: For every $i \geq 0$, there is an accept path for $x y^i z$.
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$
in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$
$q_0 \ q_1 \ \ldots \ q_{j-1}$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

\[ s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n \]

\[ q_0 \ q_1 \ \ldots q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \]
1.4 Non-Regular Languages

To prove (1) $M$ accepts $x y^i z$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $x y^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

```
s_1  s_2  \ldots  s_j  s_{j+1}  \ldots  s_l  \ldots  s_p  \ldots  s_n
q_0  q_1  \ldots q_{j-1}  q_j  \ldots  q_{l-1}  q_l  \ldots  q_{p-1}  q_p  \ldots  q_n
```

So the claim is proved.
To prove (1) $M$ accepts $xy^i z$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$$

$$q_0 \ q_1 \ \ldots q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n$$

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$. 
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

\[
\begin{array}{cccccccc}
  s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
  q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n \\
\end{array}
\]

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

Basis $i = 0$, $xy^iz$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

\[
\begin{array}{cccccccc}
  s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
  q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n \\
\end{array}
\]

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

Basis $i = 0$, $xy^iz = xy^0z$
1.4 Non-Regular Languages

To prove (1) \( M \) accepts \( xy^i z \) for every \( i \geq 0 \), we restate it as

Claim: For every \( i \geq 0 \), there is an accept path for \( xy^i z \) in which substring \( y^i \) invokes transitions from \( q_j \) to the same state \( q_l \).

Proof (by induction on \( i \))

\[
\begin{align*}
\begin{array}{cccccccccccc}
s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n
\end{array}
\end{align*}
\]

\( x = s_1 \ldots s_j, \ y = s_{j+1} \ldots s_l, \) and \( z = s_{l+1} \ldots s_n \).

Basis \( i = 0 \), \( xy^0 z = xy^0 z = xz \)
1.4 Non-Regular Languages

To prove (1) \( M \) accepts \( xy^i z \) for every \( i \geq 0 \), we restate it as

Claim: For every \( i \geq 0 \), there is an accept path for \( xy^i z \) in which substring \( y^i \) invokes transitions from \( q_j \) to the same state \( q_l \).

Proof (by induction on \( i \))

\[
s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n
q_0 \ q_1 \ \ldots q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n
\]

\( x = s_1 \ldots s_j, \ y = s_{j+1} \ldots s_l, \) and \( z = s_{l+1} \ldots s_n. \)

Basis \( i = 0, \ xy^0 z = xy^0 z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n \)
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^i z$ for every $i \geq 0$, we restate it as

**Claim:** For every $i \geq 0$, there is an accept path for $xy^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

**Proof (by induction on $i$)**

$$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$$

$$q_0 \ q_1 \ \ldots \ q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n$$

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

**Basis** $i = 0$, $xy^i z = xy^0 z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n$

that is accepted with path $q_0 q_1 \ldots q_{j-1}$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

\[
\begin{array}{cccccccc}
& s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n \\
\end{array}
\]

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

Basis $i = 0$, $xy^i z = xy^0 z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n$
that is accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. 
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^i z$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$$x = s_1 \ldots s_j, \quad y = s_{j+1} \ldots s_l, \quad \text{and} \quad z = s_{l+1} \ldots s_n.$$ 

Basis $i = 0$, $xy^0 z = xy^0 z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n$ that is accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. The claim is true.
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$
in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

Basis $i = 0$, $xy^0z = xz = s_1 \ldots s_js_{l+1} \ldots s_n$
that is accepted with path $q_0q_1 \ldots q_{j-1}q_jq_{l+1} \ldots q_p \ldots q_n$.
The claim is true.

Basis $i = 1$, $xy^i z$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$\begin{align*}
&\begin{array}{cccccccc}
s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n
\end{array} \\
x = s_1 \ldots s_j, \ y = s_{j+1} \ldots s_l, \text{ and } z = s_{l+1} \ldots s_n.
\end{align*}$

Basis $i = 0$, $xy^iz = xy^0z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n$ that is accepted with path $q_0q_1 \ldots q_{j-1}q_jq_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Basis $i = 1$, $xy^iz = xy^1z$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

\[
\begin{array}{cccccccccccc}
  s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
  q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n \\
\end{array}
\]

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

**Basis** $i = 0$, $xy^0z = xy^0z = xz = s_1 \ldots sjs_{l+1} \ldots s_n$

that is accepted with path $q_0q_1 \ldots q_{j-1}q_jq_{l+1} \ldots q_p \ldots q_n$.

The claim is true.

**Basis** $i = 1$, $xy^1z = xy^1z = xyz = s$
1.4 Non-Regular Languages

To prove (1) \( M \) accepts \( xy^i z \) for every \( i \geq 0 \), we restate it as

Claim: For every \( i \geq 0 \), there is an accept path for \( xy^i z \) in which substring \( y^i \) invokes transitions from \( q_j \) to the same state \( q_l \).

Proof (by induction on \( i \))

\[
\begin{array}{cccccccccc}
  s_1 & s_2 & \ldots & s_j & s_{j+1} & \ldots & s_l & \ldots & s_p & \ldots & s_n \\
  q_0 & q_1 & \ldots & q_{j-1} & q_j & \ldots & q_{l-1} & q_l & \ldots & q_p & \ldots & q_n
\end{array}
\]

\( x = s_1 \ldots s_j, \ y = s_{j+1} \ldots s_l, \) and \( z = s_{l+1} \ldots s_n. \)

Basis \( i = 0 \), \( xy^i z = xy^0 z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n \)

that is accepted with path \( q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n \).

The claim is true.

Basis \( i = 1 \), \( xy^i z = xy^1 z = xy z = s \in A \), accepted with path \( q_0 q_1 \ldots q_{j-1} \)
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^i z$ for every $i \geq 0$, we restate it as

**Claim:** For every $i \geq 0$, there is an accept path for $xy^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

**Proof (by induction on $i$)**

\[
s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n
\]

\[
q_0 \ q_1 \ \ldots q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n
\]

Let $x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

**Basis** $i = 0$, $xy^0z = x \in A$, accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. The claim is true.

**Basis** $i = 1$, $xy^1z = xyz = s \in A$, accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. The claim is true.
To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

Basis $i = 0$, $xy^0z = xy^0z = xz = s_1 \ldots s_js_{l+1} \ldots s_n$ that is accepted with path $q_0q_1 \ldots q_{j-1}q_jq_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Basis $i = 1$, $xy^1z = xy^1z = xyz = s \in A$, accepted with path $q_0q_1 \ldots q_{j-1}q_jq_{j+1} \ldots q_lq_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Assumption: assume that for $i = k \geq 1$, the claim is true.
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^i z$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

Assumption: assume that for $i = k \geq 1$, the claim is true.

Induction: $xy^{k+1} z$
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^i z$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^i z$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$s_1 \ s_2 \ \ldots \ s_j \ s_{j+1} \ \ldots \ s_l \ \ldots \ s_p \ \ldots \ s_n$
$q_0 \ q_1 \ \ldots \ q_{j-1} \ q_j \ \ldots \ q_{l-1} \ q_l \ \ldots \ q_p \ \ldots \ q_n$

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

Basis $i = 0$, $xy^i z = xy^0 z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n$

that is accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Basis $i = 1$, $xy^i z = xy^1 z = xyz = s \in A$, accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{j+1} \ldots q_l q_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Assumption: assume that for $i = k \geq 1$, the claim is true.

Induction: $xy^{k+1} z = xy^k yz$; by the assumption, substring $y^k$ invokes transitions from $y_j$ to $y_l$, 

1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$
in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

$x = s_1 \ldots s_j$, $y = s_{j+1} \ldots s_l$, and $z = s_{l+1} \ldots s_n$.

Basis $i = 0$, $xy^i z = xy^0 z = xz = s_1 \ldots s_{j}s_{l+1} \ldots s_n$
that is accepted with path $q_0q_1 \ldots q_jq_{j+1} \ldots q_{l}q_{l+1} \ldots q_{p} \ldots q_{n}$. The claim is true.

Basis $i = 1$, $xy^i z = xy^1 z = xyz = s \in A$, accepted with path
$q_0q_1 \ldots q_{j-1}q_{j}q_{j+1} \ldots q_{l}q_{l+1} \ldots q_{p} \ldots q_{n}$. The claim is true.

Assumption: assume that for $i = k \geq 1$, the claim is true.

Induction: $xy^{k+1}z = xy^k yz$; by the assumption, substring $y^k$ invokes
transitions from $y_j$ to $y_l$, the following $y$ will invoke transitions
from $y_l$ (which is $y_j$) to $y_l$, and continue to the accept state $q_n$. 
1.4 Non-Regular Languages

To prove (1) $M$ accepts $xy^iz$ for every $i \geq 0$, we restate it as

Claim: For every $i \geq 0$, there is an accept path for $xy^iz$ in which substring $y^i$ invokes transitions from $q_j$ to the same state $q_l$.

Proof (by induction on $i$)

Basis $i = 0$, $xy^0z = xz = s_1 \ldots s_j s_{l+1} \ldots s_n$ that is accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Basis $i = 1$, $xy^1z = xyz = s \in A$, accepted with path $q_0 q_1 \ldots q_{j-1} q_j q_{l+1} \ldots q_p \ldots q_n$. The claim is true.

Assumption: assume that for $i = k \geq 1$, the claim is true.

Induction: $xy^{k+1}z = xy^k yz$; by the assumption, substring $y^k$ invokes transitions from $y_j$ to $y_l$, the following $y$ will invoke transitions from $y_l$ (which is $y_j$) to $y_l$, and continue to the accept state $q_n$.

So the claim is proved.
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages. So we first look at its applications in regular languages. Let $\Sigma = \{a, b\}$. Define $L = \{s : s = a^k$ or $s = a^k b^3, k \geq 3\}$. So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc. Now we apply the Pumping lemma to $L$. 

- assume pumping length $p = 7$. How do we know this is okay?
- choose $s = aaaaabbb$, such that $|s| = 8 > p = 7$.
- according to the lemma, $s = xyz$, for some substrings $x, y, z$.
- (1) $y$ can be all $a$'s, or (2) $a$'s followed by $b$'s, or (3) all $b$'s;
- The lemma is correct because case (1) would allow pumped strings to belong to $L$ as well.

If we were asked to prove $L$ is not regular, we would have to show that none of the above 3 cases allow pumped strings to belong to $L$. 
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let \( \Sigma = \{a, b\} \). Define

\[
L = \{ s : s = a^k \text{ or } s = a^k b^3, \, k \geq 3 \}
\]

So \( L \) contains strings like \(aaa, aaaa, aaabbb, aaaaabbb\), etc.
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, \ k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$. 
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, \ k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaaabbb, \text{ etc.}$

Now we apply the Pumping lemma to $L$.

• assume pumping length $p = 7$. 
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{ s : s = a^k \text{ or } s = a^k b^3, \ k \geq 3 \}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$.

- assume pumping length $p = 7$. How do we know this is okay?
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, \ k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$.

- assume pumping length $p = 7$. How do we know this is okay?
- choose $s = aaaaaabbb$, such that $|s| = 8 > p = 7$. 
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$.

• assume pumping length $p = 7$. How do we know this is okay?
• choose $s = aaaaabbb$, such that $|s| = 8 > p = 7$.
• according to the lemma, $s = xyz$, for some substrings $x, y, z$
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, \ k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$.

- assume pumping length $p = 7$. How do we know this is okay?
- choose $s = aaaaaabbb$, such that $|s| = 8 > p = 7$.
- according to the lemma, $s = xyz$, for some substrings $x, y, z$
- (1) $y$ can be all $a$’s, or (2) $a$’s followed by $b$’s, or (3) all $b$’s;
1.4 Non-Regular Languages

The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$.

• assume pumping length $p = 7$. How do we know this is okay?
• choose $s = aaaaaabbb$, such that $|s| = 8 > p = 7$.
• according to the lemma, $s = xyz$, for some substrings $x, y, z$
• (1) $y$ can be all $a$’s, or (2) $a$’s followed by $b$’s, or (3) all $b$’s;
• The lemma is correct because case (1) would allow pumped strings to belong to $L$ as well.
The Pumping lemma exists without referring at all to non-regular languages.

So we first look at its applications in regular languages.

Let $\Sigma = \{a, b\}$. Define

$$L = \{s : s = a^k \text{ or } s = a^k b^3, \ k \geq 3\}$$

So $L$ contains strings like $aaa, aaaa, aaabbb, aaaaabbb$, etc.

Now we apply the Pumping lemma to $L$.

- assume pumping length $p = 7$. How do we know this is okay?
- choose $s = aaaaaabbb$, such that $|s| = 8 > p = 7$.
- according to the lemma, $s = xyz$, for some substrings $x, y, z$
- (1) $y$ can be all $a$’s, or (2) $a$’s followed by $b$’s, or (3) all $b$’s;
- The lemma is correct because case (1) would allow pumped strings to belong to $L$ as well.

If we were asked to prove $L$ is not regular, we would have to show that none of the above 3 cases allow pumped strings to belong to $L$. 
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)

Claim: \( L_2 = \{0^k 1^k : k \geq 0\} \) is not a regular language.

Proof: (use proof by contradiction) Assume \( L_2 \) is a regular language. According to the Pumping Lemma, there is a number \( p \), pumping length. Choose string \( s = 0^p 1^p \in L_2 \). Thus \( |s| = 2p \geq p \).

According to the Pumping Lemma, \( s = xyz \), such that

1. \( xy^i z \in L_2 \), for every \( i \geq 0 \)
2. \( |y| > 0 \),
3. \( |xy| \leq p \),

From (2) and (3) we conclude that \( y = 0^k \), for some \( k \geq 1 \); \( x = 0^l \), for some \( l \geq 0 \); \( z = 0^{p-l-k} \).

From (1), we conclude string \( xy^0 z \in L_2 \), that is \( 0^l 0^0 0^{p-l-k} = 0^{p-k} \in L_2 \).

But such string is abnormal, NOT in the format \( 0^k 1^k \). Contradicts!

So \( L_2 \) is not a regular language.
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)

Claim: $L_2 = \{0^k 1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language. According to the Pumping Lemma, there is a number $p$, pumping length. Choose string $s = 0^p 1^p \in L_2$. Thus $|s| = 2p \geq p$. According to the Pumping lemma, $s = xyz$, such that

1. $xy^iz \in L_2$, for every $i \geq 0$
2. $|y| > 0$
3. $|xy| \leq p$

From (2) and (3) we conclude that

$y = 0^k$, for some $k \geq 1$;

$x = 0^l$, for some $l \geq 0$;

$z = 0^{p-l-k}$.

From (1), we conclude string $xy^0z \in L_2$ That is $0^l 0^0 0^{p-l-k} = 0^{p-k} \in L_2$. But such string is abnormal, NOT in the format $0^k 1^k$. Contradicts!

So $L_2$ is not a regular language.
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: \( L_2 = \{0^k1^k : k \geq 0\} \) is not a regular language.
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: \( L_2 = \{0^k1^k : k \geq 0\} \) is not a regular language.

Proof: (use proof by contradiction) Assume \( L_2 \) is a regular language.
Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.
   According to the Pumping Lemma, there is a number $p$, pumping length.
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: \( L_2 = \{0^k1^k : k \geq 0\} \) is not a regular language.

Proof: (use proof by contradiction) Assume \( L_2 \) is a regular language.
   According to the Pumping Lemma, there is a number \( p \), pumping length.

\[\text{Choose string } s = 0^p1^p \in L_2. \text{ Thus } |s| = 2p \geq p.\]

According to the Pumping lemma, \( s = xyz \), such that
\(1) \ xy^i z \in L_2 \), for every \( i \geq 0 \)
\(2) \ |y| > 0 \),
\(3) \ |xy| \leq p \).

From (2) and (3) we conclude that \( y = 0^k \), for some \( k \geq 1 \);
\(x = 0^l \), for some \( l \geq 0 \);
\(z = 0^{p-l-k}1^p \).

From (1), we conclude string \( xy^0z \in L_2 \).
That is \( 0^l0^00^{p-l-k}1^p \in L_2 \).

But such string is abnormal, NOT in the format \( 0^k1^k \). Contradicts!

So \( L_2 \) is not a regular language.
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: \( L_2 = \{0^k1^k : k \geq 0\} \) is not a regular language.

Proof: (use proof by contradiction) Assume \( L_2 \) is a regular language.
   According to the Pumping Lemma, there is a number \( p \), pumping length.
   Choose string \( s = 0^p1^p \in L_2 \). Thus \( |s| = 2p \geq p \)
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.
According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$
According to the Pumping lemma, $s = xyz$, such that

1. $xy^iz \in L_2$, for every $i \geq 0$
2. $|y| > 0$,
3. $|xy| \leq p$,
1.4 Non-regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.

According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$

According to the Pumping lemma, $s = xyz$, such that

1. $xy^iz \in L_2$, for every $i \geq 0$
2. $|y| > 0$,
3. $|xy| \leq p$,

From (2) and (3) we conclude that $y = 0^k$, for some $k \geq 1$;
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.

According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$

According to the Pumping lemma, $s = xyz$, such that

(1) $xy^iz \in L_2$, for every $i \geq 0$
(2) $|y| > 0$,
(3) $|xy| \leq p$,

From (2) and (3) we conclude that

$y = 0^k$, for some $k \geq 1$;
$x = 0^l$, for some $l \geq 0$;
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.
According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$
According to the Pumping lemma, $s = xyz$, such that

1. $xy^iz \in L_2$, for every $i \geq 0$
2. $|y| > 0$,
3. $|xy| \leq p$,

From (2) and (3) we conclude that
$y = 0^k$, for some $k \geq 1$;
$x = 0^l$, for some $l \geq 0$;
$z = 0^{p-l-k}1^p$;
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.
According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$
According to the Pumping lemma, $s = xyz$, such that

(1) $xyz \in L_2$, for every $i \geq 0$
(2) $|y| > 0$,
(3) $|xy| \leq p$,

From (2) and (3) we conclude that
$y = 0^k$, for some $k \geq 1$;
$x = 0^l$, for some $l \geq 0$;
$z = 0^{p-l-k}1^p$;

From (1), we conclude string $xy^0z \in L_2$
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.
According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$
According to the Pumping lemma, $s = xyz$, such that

1. $xyz \in L_2$, for every $i \geq 0$
2. $|y| > 0$,
3. $|xy| \leq p$,

From (2) and (3) we conclude that
\[ y = 0^k, \text{ for some } k \geq 1; \]
\[ x = 0^l, \text{ for some } l \geq 0; \]
\[ z = 0^{p-l-k}1^p; \]

From (1), we conclude string $xy^0z \in L_2$. That is $0^l0^0 \times 0^p-l-k1^p$
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: \( L_2 = \{0^k1^k : k \geq 0\} \) is not a regular language.

Proof: (use proof by contradiction) Assume \( L_2 \) is a regular language.
According to the Pumping Lemma, there is a number \( p \), pumping length.

Choose string \( s = 0^p1^p \in L_2 \). Thus \( |s| = 2p \geq p \)
According to the Pumping lemma, \( s = xyz \), such that

(1) \( xy^iz \in L_2 \), for every \( i \geq 0 \)
(2) \( |y| > 0 \),
(3) \( |xy| \leq p \),

From (2) and (3) we conclude that
\[ y = 0^k, \text{ for some } k \geq 1; \]
\[ x = 0^l, \text{ for some } l \geq 0; \]
\[ z = 0^{p-l-k}1^p; \]

From (1), we conclude string \( xy^0z \in L_2 \). That is \( 0^l0^0 \times 0^k0^{p-l-k}1^p = 0^{p-k}1^p \)
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)

Claim: \( L_2 = \{0^k1^k : k \geq 0\} \) is not a regular language.

Proof: (use proof by contradiction) Assume \( L_2 \) is a regular language. According to the Pumping Lemma, there is a number \( p \), pumping length.

Choose string \( s = 0^p1^p \in L_2 \). Thus \( |s| = 2p \geq p \)

According to the Pumping lemma, \( s = xyz \), such that

1. \( xy^iz \in L_2 \), for every \( i \geq 0 \)
2. \( |y| > 0 \),
3. \( |xy| \leq p \),

From (2) and (3) we conclude that

\( y = 0^k \), for some \( k \geq 1 \);
\( x = 0^l \), for some \( l \geq 0 \);
\( z = 0^{p-l-k}1^p \);

From (1), we conclude string \( xy^0z \in L_2 \). That is \( 0^l0^0 \times 0^p-l-k1^p = 0^{p-k}1^p \in L_2 \).
1.4 Non-Regular Languages

Using the Pumping lemma to prove certain languages are NOT regular.

Example 1.73 (page 80)
Claim: $L_2 = \{0^k1^k : k \geq 0\}$ is not a regular language.

Proof: (use proof by contradiction) Assume $L_2$ is a regular language.
According to the Pumping Lemma, there is a number $p$, pumping length.

Choose string $s = 0^p1^p \in L_2$. Thus $|s| = 2p \geq p$
According to the Pumping lemma, $s = xyz$, such that

(1) $xy^iz \in L_2$, for every $i \geq 0$
(2) $|y| > 0$,
(3) $|xy| \leq p$,

From (2) and (3) we conclude that
$y = 0^k$, for some $k \geq 1$;
$x = 0^l$, for some $l \geq 0$;
$z = 0^{p-l-k}1^p$;

From (1), we conclude string $xy^0z \in L_2$. That is $0^l0^0 \times k0^{p-l-k}1^p = 0^{p-k}1^p \in L_2$.

But such string is abnormal, NOT in the format $0^k1^k$. Contradicts!
So $L_2$ is not a regular language.
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0 \) \( p^2 \) (assuming \( p \) is even).

- Because \( |s| = p^2 + p^2 \geq p \), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \( |y| > 0 \) and \( |xy| \leq p \).

There are three possible cases for what \( y \) may be:

1. \( y = 0^k \) for some \( k \geq 1 \), \( x = 0^l \) for some \( l \geq 0 \), \( z = 0^{p^2 - k - l} \)
2. \( y = 0^k 1^l \), \( x = 0^{p^2 - k} \), \( z = 1^{p^2 - l} \), for some \( k \geq 1 \), \( l \geq 1 \)
3. \( y = 1^k \), \( z = 1^l \), \( x = 0^{p^2 - 1^k - 1^l} \), for some \( k \geq 1 \), \( l \geq 0 \)

Now we need to show that for every one of the 3 cases, pumping \( y \) will lead to creating strings NOT in the format of \( 0^k 1^k \).

But the lemma says that such abnormal strings belong to \( L_2 \). Contradicts.
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{p/2} 1^{p/2} \) (assuming \( p \) is even)
Alternatively, we may choose $s = 0^\frac{p}{2} 1^\frac{p}{2}$ (assuming $p$ is even).

- Because $|s| = \frac{p}{2} + \frac{p}{2} \geq p$, we can apply the lemma to $s$. 

Now we need to show that for every one of the 3 cases, pumping $y$ will lead to creating strings NOT in the format of $0^k 1^k$.

But the lemma says that such abnormal strings belong to $L_2$. Contradicts.
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{\frac{p}{2}} 1^{\frac{p}{2}} \) (assuming \( p \) is even)

- Because \(|s| = \frac{p}{2} + \frac{p}{2} \geq p\), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \(|y| > 0\) and \(|xy| \leq p\).
1.4 Non-Regular Languages

Alternatively, we may choose $s = 0^\frac{p}{2}1^\frac{p}{2}$ (assuming $p$ is even)

- Because $|s| = \frac{p}{2} + \frac{p}{2} \geq p$, we can apply the lemma to $s$;
- thus $s = xyz$ for some substrings $x, y, z$ such that $|y| > 0$ and $|xy| \leq p$.
- There are three possible cases for what $y$ may be:
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{\frac{p}{2}} 1^{\frac{p}{2}} \) (assuming \( p \) is even)

- Because \( |s| = \frac{p}{2} + \frac{p}{2} \geq p \), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \( |y| > 0 \) and \( |xy| \leq p \).
- There are three possible cases for what \( y \) may be:

  1. \( y = 0^k \) for some \( k \geq 1 \), \( x = 0^l \) for some \( l \geq 0 \), \( z = 0^{\frac{p}{2}-k-l} \)
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{\frac{p}{2}} 1^{\frac{p}{2}} \) (assuming \( p \) is even)

- Because \( |s| = \frac{p}{2} + \frac{p}{2} \geq p \), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \( |y| > 0 \) and \( |xy| \leq p \).

There are three possible cases for what \( y \) may be:

1. \( y = 0^k \) for some \( k \geq 1 \), \( x = 0^l \) for some \( l \geq 0 \), \( z = 0^{\frac{p}{2} - k - l} \)
2. \( y = 0^k 1^l \), \( x = 0^{\frac{p}{2} - k} \), \( z = 1^{\frac{p}{2} - l} \), for some \( k \geq 1, l \geq 1 \)
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{\frac{p}{2}} 1^{\frac{p}{2}} \) (assuming \( p \) is even)

- Because \( |s| = \frac{p}{2} + \frac{p}{2} \geq p \), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \( |y| > 0 \) and \( |xy| \leq p \).
- There are three possible cases for what \( y \) may be:
  1. \( y = 0^k \) for some \( k \geq 1 \), \( x = 0^l \) for some \( l \geq 0 \), \( z = 0^{\frac{p}{2} - k - l} \)
  2. \( y = 0^k 1^l \), \( x = 0^{\frac{p}{2} - k} \), \( z = 1^{\frac{p}{2} - l} \), for some \( k \geq 1 \), \( l \geq 1 \)
  3. \( y = 1^k \), \( z = 1^l \), \( x = 0^{\frac{p}{2}} 1^{\frac{p}{2} - k - l} \), for some \( k \geq 1 \), \( l \geq 0 \)
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{p/2}1^{p/2} \) (assuming \( p \) is even)

- Because \( |s| = \frac{p}{2} + \frac{p}{2} \geq p \), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \( |y| > 0 \) and \( |xy| \leq p \).

- There are three possible cases for what \( y \) may be:
  1. \( y = 0^k \) for some \( k \geq 1 \), \( x = 0^l \) for some \( l \geq 0 \), \( z = 0^{p/2-k-l} \)
  2. \( y = 0^k1^l \), \( x = 0^{p/2-k} \), \( z = 1^{p/2-l} \), for some \( k \geq 1 \), \( l \geq 1 \)
  3. \( y = 1^k \), \( z = 1^l \), \( x = 0^{p/2}1^{p/2-k-l} \), for some \( k \geq 1 \), \( l \geq 0 \)

- Now we need to show that for every one of the 3 cases, pumping \( y \) will lead to creating strings NOT in the format of \( 0^k1^k \).
1.4 Non-Regular Languages

Alternatively, we may choose \( s = 0^{p/2} 1^{p/2} \) (assuming \( p \) is even)

- Because \( |s| = \frac{p}{2} + \frac{p}{2} \geq p \), we can apply the lemma to \( s \);
- thus \( s = xyz \) for some substrings \( x, y, z \) such that \( |y| > 0 \) and \( |xy| \leq p \).
- There are three possible cases for what \( y \) may be:
  
  (1) \( y = 0^k \) for some \( k \geq 1 \), \( x = 0^l \) for some \( l \geq 0 \), \( z = 0^{p/2-k-l} \)
  
  (2) \( y = 0^k 1^l \), \( x = 0^{p/2-k} \), \( z = 1^{p/2-l} \), for some \( k \geq 1 \), \( l \geq 1 \)
  
  (3) \( y = 1^k \), \( z = 1^l \), \( x = 0^{p/2} 1^{p/2-k-l} \), for some \( k \geq 1 \), \( l \geq 0 \)

- Now we need to show that for every one of the 3 cases, pumping \( y \) will lead to creating strings NOT in the format of \( 0^k 1^k \).
- But the lemma says that such abnormal strings belong to \( L_2 \). Contradicts.
Example: Prove that language of two repeat strings $L = \{ww : w \in \{0, 1\}^*\}$ is not regular.

The most critical step is to identify such a string $s \in L$ such that pumping its prefix of at most $p$ symbols would result in abnormal strings of form $ww$.

Hint: choose a string $s$ of length $|s| \geq p$, then we may locate $y$ in the first $s$; pumping the $y$ in the first $s$ would make the new string NOT in the format of $ww$.

One good choice of string is $s = 0^p10^p1$. Please complete the proof!
Example: Prove that language of two repeat strings $L = \{ww : w \in \{0, 1\}^*\}$ is not regular.

The most critical step is to identify such a string $s \in L$ such that pumping its prefix of at most $p$ symbols would result in abnormal strings of form $ww$.

Hint: choose a string $s$ of length $|s| \geq p$, then we may locate $y$ in the first $s$; pumping the $y$ in the first $s$ would make the new string NOT in the format of $ww$.

One good choice of string is $s = 0^p10^p1$. Please complete the proof!
Example: Prove that language of two repeat strings

\[ L = \{ww : w \in \{0, 1\}^*\} \]

is not regular.
Example: Prove that language of two repeat strings

\[ L = \{ww : w \in \{0, 1\}^*\} \]

is not regular.

The most critical step is to identify such a string \( s \in L \) such that pumping its prefix of at most \( p \) symbols would result in abnormal strings of form \( ww \).
Example: Prove that language of two repeat strings

\[ L = \{ww : w \in \{0, 1\}^*\} \]

is not regular.

The most critical step is to identify such a string \( s \in L \) such that pumping its prefix of at most \( p \) symbols would result in abnormal strings of form \( ww \).

Hint: choose a string \( s \) of length \( |s| \geq p \), then we may locate \( y \) in the first \( s \); pumping the \( y \) in the first \( s \) would make the new string NOT in the format of \( ww \).
1.4 Non-Regular Languages

Example: Prove that language of two repeat strings

\[ L = \{ww : w \in \{0, 1\}^*\} \]

is not regular.

The most critical step is to identify such a string \( s \in L \) such that
pumping its prefix of at most \( p \) symbols
would result in abnormal strings of form \( ww \).

Hint: choose a string \( s \) of length \( |s| \geq p \), then we may locate \( y \)
in the first \( s \); pumping the \( y \) in the first \( s \) would make the new
string NOT in the format of \( ww \).

One good choice of string is \( s = 0^p10^p1 \). Please complete the proof!
1.4 Non-Regular Languages

Often we create abnormal strings by making multiple copies of $y$. But it may not work on the following example:

$L = \{0^i 1^j : \text{positive integers } i > j\}$

To prove $L$ is not regular, we assume it is regular. Then by the lemma, there is a number $p$, the pumping length;

Selecting string $s = 0^{p+1} 1^p$, by the lemma $s = xyz$, for some substrings $x, y, z$; $|xy| \leq p$, and $|y| > 0$;

Because $|xy| \leq p$, $y = 0^k$ for some $k \geq 1$ and $x = 0^l$ for some $l \geq 0$.

$s = 0^l 0^k 0^{p+1-l-k} 1^p = 0^p+1+k 1^p$. Pumping up $0^k$ would only add more 0's, NOT creating abnormal strings.

Pumping down will work!
1.4 Non-Regular Languages

“Pumping down”
1.4 Non-Regular Languages

“Pumping down”

Often we create \textit{abnormal} strings by making multiple copy of $y$. But it may not work on the following example:
“Pumping down”

Often we create abnormal strings by making multiple copy of $y$. But it may not work on the following example:

$$L = \{0^i1^j : \text{ positive integers } i > j \}$$
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of \( y \). But it may not work on the following example:

\[
L = \{0^i1^j : \text{positive integers } i > j\}
\]

- To prove \( L \) is not regular, we assume it is regular

Pumping down will work!
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of $y$. But it may not work on the following example:

$$L = \{0^i1^j : \text{positive integers } i > j\}$$

• To prove $L$ is not regular, we assume it is regular
  Then by the lemma, there is a number $p$, the pumping length;

Pumping up $0^k$ would only add more 0's, NOT creating abnormal strings.

Pumping down will work!
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of $y$. But it may not work on the following example:

$$L = \{0^i1^j : \text{positive integers } i > j\}$$

- To prove $L$ is not regular, we assume it is regular
  Then by the lemma, there is a number $p$, the pumping length;
- Selecting string $s = 0^{p+1}1^p$, by the lemma
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of $y$. But it may not work on the following example:

$L = \{0^i1^j : \text{positive integers } i > j \}$

- To prove $L$ is not regular, we assume it is regular. Then by the lemma, there is a number $p$, the pumping length;
- Selecting string $s = 0^{p+1}1^p$, by the lemma $s = xyz$, for some substrings $x, y, z$; $|xy| \leq p$, and $|y| > 0$;
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of $y$. But it may not work on the following example:

$$L = \{0^i1^j : \text{positive integers } i > j \}$$

- To prove $L$ is not regular, we assume it is regular. Then by the lemma, there is a number $p$, the pumping length;

- Selecting string $s = 0^{p+1}1^p$, by the lemma $s = xyz$, for some substrings $x, y, z$; $|xy| \leq p$, and $|y| > 0$;

- Because $|xy| \leq p$, $y = 0^k$ for some $k \geq 1$ and $x = 0^l$ for some $l \geq 0$. 

Pumping up $0^k$ would only add more 0’s, NOT creating abnormal strings.

Pumping down will work!
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of \( y \). But it may not work on the following example:

\[
L = \{0^i1^j : \text{positive integers } i > j \}
\]

- To prove \( L \) is not regular, we assume it is regular.
  Then by the lemma, there is a number \( p \), the pumping length;

- Selecting string \( s = 0^{p+1}1^p \), by the lemma
  \( s = xyz \), for some substrings \( x, y, z \); \( |xy| \leq p \), and \( |y| > 0 \);

- Because \( |xy| \leq p \), \( y = 0^k \) for some \( k \geq 1 \) and \( x = 0^l \) for some \( l \geq 0 \).

- \( s = 0^l0^k0^{p+1-l-k}1^p = 0^{p+1+k}1^p \).
“Pumping down”

Often we create abnormal strings by making multiple copy of \( y \). But it may not work on the following example:

\[
L = \{0^i1^j : \text{ positive integers } i > j \}
\]

- To prove \( L \) is not regular, we assume it is regular
  Then by the lemma, there is a number \( p \), the pumping length;

- Selecting string \( s = 0^{p+1}1^p \), by the lemma
  \( s = xyz \), for some substrings \( x, y, z \); \( |xy| \leq p \), and \( |y| > 0 \);

- Because \( |xy| \leq p \), \( y = 0^k \) for some \( k \geq 1 \) and \( x = 0^l \) for some \( l \geq 0 \).

- \( s = 0^l0^k0^{p+1-l-k}1^p = 0^{p+1+k}1^p \).
  
  **Pumping up** \( 0^k \) would only add more 0’s, NOT creating abnormal strings.
1.4 Non-Regular Languages

“Pumping down”

Often we create abnormal strings by making multiple copy of \( y \). But it may not work on the following example:

\[ L = \{0^i1^j : \text{positive integers } i > j \} \]

- To prove \( L \) is not regular, we assume it is regular. Then by the lemma, there is a number \( p \), the pumping length;
- Selecting string \( s = 0^{p+1}1^p \), by the lemma \( s = xyz \), for some substrings \( x, y, z \); \( |xy| \leq p \), and \( |y| > 0 \);
- Because \( |xy| \leq p \), \( y = 0^k \) for some \( k \geq 1 \) and \( x = 0^l \) for some \( l \geq 0 \).
- \( s = 0^l0^k0^{p+1-l-k}1^p = 0^{p+1+k}1^p \).
  - **Pumping up** \( 0^k \) would only add more 0’s, NOT creating abnormal strings.
- **Pumping down** will work!
1.4 Non-Regular Languages

Pumping down strings $ = l_0 k_0 p + 1 - l - k_1 p$.

New string $l(0 k) i_0 p + 1 - l - k_1 p$.

Let $i = 0$, $(0 k) i = \epsilon$.

By the lemma, new string $l(0 k) i_0 p + 1 - l - k_1 p = 0 p + 1 - k_1 p \in L$.

However, the number of 0's is NOT $\geq$ that of 1's in $0 p + 1 - k_1 p$.

Because $k \geq 1$.

Contradicts!
1.4 Non-Regular Languages

Pumping down string $s = 0^{l0^k0^{p+1-l-k}1^p}$. 
Pumping down string $s = 0^l0^k0^p+1-l-k1^p$.

New string $0^l(0^k)^i0^p+1-l-k1^p$, 

However, the number of 0's is NOT ≥ that of 1's in $0^l(0^k)^i0^p+1-l-k1^p$, because $k ≥ 1$.

Contradicts!
1.4 Non-Regular Languages

Pumping down string $s = 0^l0^k0^{p+1-l-k}1^p$.

New string $0^l(0^k)^i0^{p+1-l-k}1^p$, let $i = 0$, $(0^k)^i = \epsilon$. 
Pumping down string \( s = 0^l0^k0^{p+1-l-k}1^p \).

New string \( 0^l(0^k)^i0^{p+1-l-k}1^p \), let \( i = 0 \), \( (0^k)^i = \epsilon \).

By the lemma new string \( 0^l(0^k)^i0^{p+1-l-k}1^p = 0^{p+1-k}1^p \in L \);
1.4 Non-Regular Languages

Pumping down string \( s = 0^l0^k0^{p+1-l-k}1^p \).

New string \( 0^l(0^k)^i0^{p+1-l-k}1^p \), let \( i = 0 \), \( (0^k)^i = \epsilon \).

By the lemma new string \( 0^l(0^k)^i0^{p+1-l-k}1^p = 0^{p+1-k}1^p \in L \);
However, the number of 0’s is NOT \( \geq \) that of 1’s in \( 0^{p+1-k}1^p \),
1.4 Non-Regular Languages

**Pumping down** string \( s = 0^l0^k0^{p+1-l-k}1^p \).

New string \( 0^l(0^k)^i0^{p+1-l-k}1^p \), let \( i = 0 \), \((0^k)^i = \epsilon\).

By the lemma new string \( 0^l(0^k)^i0^{p+1-l-k}1^p = 0^{p+1-k}1^p \in L; \)

However, the number of 0’s is NOT \( \geq \) that of 1’s in \( 0^{p+1-k}1^p \),

because \( k \geq 1 \).

Contradicts!
1.4 Non-Regular Languages

Further questions concerning the Pumping Lemma and NFAs:

• In proving the Pumping Lemma, we assume a DFA \( M \) for language \( A \). Would an NFA work well for the proof?

• The pumping lemma underscores the fundamental limitation of FAs, the capability of such a machine equal that of a constant-memory algorithm;

• Loops (self-loops or not) in an NFA allows to accepts infinite number of strings;
Further questions concerning the Pumping Lemma and NFAs:
Further questions concerning the Pumping Lemma and NFAs:

- In proving the Pumping Lemma, we assume a DFA $M$ for language $A$. Would an NFA work well for the proof?
1.4 Non-Regular Languages

Further questions concerning the Pumping Lemma and NFAs:

• In proving the Pumping Lemma, we assume a DFA $M$ for language $A$. Would an NFA work well for the proof?

• The pumping lemma underscores the fundamental limitation of FAs, the capability of such a machine equal that of a constant-memory algorithm;
Further questions concerning the Pumping Lemma and NFAs:

- In proving the Pumping Lemma, we assume a DFA $M$ for language $A$. Would an NFA work well for the proof?
- The pumping lemma underscores the fundamental limitation of FAs, the capability of such a machine equal that of a constant-memory algorithm;
- Loops (self-loops or not) in an NFA allows to accepts infinite number of strings;
1.4 Non-Regular Languages

Additional notes for regular expressions

(1) In conversion from regular languages to regular expressions, we used DFA. Can you use NFA instead?

(2) Regular expressions: operations and simplification

• Every regular expression describes a set of strings;
• Operations star $^*$, concatenation $\circ$, and union $\cup$ are used to make larger regular expressions;
• Let $A, B, C$ be regular expressions; where $A = \{a_1, a_2, \ldots\}$, $B = \{b_1, b_2, \ldots\}$, and $C = \{c_1, c_2, \ldots\}$, then $(A \cup B) \circ C = ?$

$(A \cup B) \circ C = ?$

Is $(A \circ B)^*$ the same as $(A \cup B)^*$ or $A^* \circ B^*$?
1.4 Non-Regular Languages

Additional notes for regular expressions
1.4 Non-Regular Languages

Additional notes for regular expressions

(1) In conversion from regular languages to regular expressions, we used DFA. Can you use NFA instead?
1.4 Non-Regular Languages

**Additional notes for regular expressions**

(1) In conversion from regular languages to regular expressions, we used DFA. Can you use NFA instead?

(2) Regular expressions: operations and simplification
1.4 Non-Regular Languages

**Additional notes for regular expressions**

(1) In conversion from regular languages to regular expressions, we used DFA. Can you use NFA instead?

(2) Regular expressions: operations and simplification

- every regular expression describes a set of strings;
1.4 Non-Regular Languages

Additional notes for regular expressions

(1) In conversion from regular languages to regular expressions, we used DFA. Can you use NFA instead?

(2) Regular expressions: operations and simplification

• every regular expression describes a set of strings;
• operations star *, concatenation ⊙, and union ∪ are used to make larger regular expressions;
1.4 Non-Regular Languages

Additional notes for regular expressions

(1) In conversion from regular languages to regular expressions, we used DFA. Can you use NFA instead?

(2) Regular expressions: operations and simplification

- every regular expression describes a set of strings;
- operations star $^*$, concatenation $\circ$, and union $\cup$ are used to make larger regular expressions;

Let $A, B, C$ be regular expressions; where $A = \{a_1, a_2, \ldots\}$, $B = \{b_1, b_2, \ldots\}$, and $C = \{c_1, c_2, \ldots\}$

Then $(A \cup B) \circ C =?\quad (A \cup B)^* =?\quad (A \circ B)^*$ the same as $(A \cup B)^*$ or $A^* \circ B^*$ ?
1.4 Non-Regular Languages

• Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;

• $\emptyset$, $\{\epsilon\}$, $\Sigma^*$, and finite languages are all regular;

• Let $A$ and $B$ be two languages. If $A \subseteq B$ and $A$ is regular, should $B$ be regular?

• If $A \subseteq B$ and $B$ is regular, should $A$ be regular?

• If $A$ is regular, should $A$ be regular?

• If $A$ and $B$ are both regular, should $A \cap B$ be regular?
1.4 Non-Regular Languages

Additional notes about regular languages:

• Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;
• $\emptyset$, $\{\epsilon\}$, $\Sigma^*$, and finite languages are all regular;
• Let $A$ and $B$ be two languages. If $A \subseteq B$ and $A$ is regular, should $B$ be regular? If $A \subseteq B$ and $B$ is regular, should $A$ be regular?
• If $A$ is regular, should $A$ be regular?
• If $A$ and $B$ are both regular, should $A \cap B$ be regular?
1.4 Non-Regular Languages

Additional notes about regular languages:

• Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;
1.4 Non-Regular Languages

Additional notes about regular languages:

- Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;
- $\emptyset$, $\{\epsilon\}$, $\Sigma^*$, and finite languages are all regular;
1.4 Non-Regular Languages

Additional notes about regular languages:

• Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;

• $\emptyset$, $\{\epsilon\}$, $\Sigma^*$, and finite languages are all regular;

• Let $A$ and $B$ be two languages.
  
  If $A \subseteq B$ and $A$ is regular, should $B$ be regular?
  If $A \subseteq B$ and $B$ is regular, should $A$ be regular?
Additional notes about regular languages:

• Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;
• $\emptyset$, $\{\epsilon\}$, $\Sigma^*$, and finite languages are all regular;
• Let $A$ and $B$ be two languages.
  
  If $A \subseteq B$ and $A$ is regular, should $B$ be regular?
  If $A \subseteq B$ and $B$ is regular, should $A$ be regular?
• If $A$ is regular, should $\overline{A}$ be regular?
1.4 Non-Regular Languages

Additional notes about regular languages:

- Every language is a subset of $\Sigma^*$ for some finite alphabet $\Sigma$;
- $\emptyset$, $\{\epsilon\}$, $\Sigma^*$, and finite languages are all regular;
- Let $A$ and $B$ be two languages.
  If $A \subseteq B$ and $A$ is regular, should $B$ be regular?
  If $A \subseteq B$ and $B$ is regular, should $A$ be regular?
- If $A$ is regular, should $\overline{A}$ be regular?
- If $A$ and $B$ are both regular, should $A \cap B$ be regular?
Summary of Chapter 1

- Finite Automata
  - FA, alphabet, transition, states
  - Computation of strings with FA
  - Regular languages
    - Design of FA diagrams for regular languages

- Nondeterminism
  - Difference between DFA and NFA
  - Design of FA diagrams
  - Equivalence between NFA to DFA
  - Regular languages are closed under $\cup$, $\circ$, $\ast$
Summary of Chapter 1

- Finite Automata

- Design of FA diagrams for regular languages

- Difference between DFA and NFA

- Design of FA diagrams

- Equivalence between NFA to DFA

- Regular languages are closed under $\cup$, $\circ$, $\ast$
Summary of Chapter 1

- Finite Automata
  - FA, alphabet, transition, states
  - computation of strings with FA
  - regular languages
    - design of FA diagrams for regular languages

- nondeterminism
  - difference between DFA and NFA
  - design of FA diagrams
  - equivalence between NFA to DFA
  - regular languages are closed under $\cup$, $\circ$, $^*$
Summary of Chapter 1

- Finite Automata
  - FA, alphabet, transition, states
  - computation of strings with FA
  - regular languages
    - design of FA diagrams for regular languages

- nondeterminism
Summary of Chapter 1

▶ Finite Automata
  • FA, alphabet, transition, states
  • computation of strings with FA
  • regular languages
    design of FA diagrams for regular languages

▶ nondeterminism
  • difference between DFA and NFA
  • design of FA diagrams
  • equivalence between NFA to DFA
  • regular languages are closed under $\cup$, $\circ$, $\ast$
Summary of Chapter 1

- Finite Automata
  - FA, alphabet, transition, states
  - computation of strings with FA
  - regular languages
    design of FA diagrams for regular languages

- nondeterminism
  - difference between DFA and NFA
  - design of FA diagrams
  - equivalence between NFA to DFA
  - regular languages are closed under $\cup$, $\circ$, $\ast$
Summary of Chapter 1

- Regular expression
  - Basic notations and composition rules for regular expression
  - Regular expression design
  - Converting regular expressions to NFA
  - Converting DFA to regular expressions

- Non-regular languages
  - Pumping lemma (principle)
  - Pumping lemma (theorem)
  - Proofs for non-regular languages
Summary of Chapter 1

- regular expression
Summary of Chapter 1

- regular expression
  - basic notations
    and composition rules for regular expression
  - regular expression design
  - converting regular expressions to NFA
  - converting DFA to regular expressions

- non-regular languages
  - pumping lemma (principle)
  - pumping lemma (theorem)
  - proofs for non-regular languages
Summary of Chapter 1

- regular expression
  - basic notations and composition rules for regular expression
  - regular expression design
  - converting regular expressions to NFA
  - converting DFA to regular expressions

- non-regular languages
Summary of Chapter 1

- regular expression
  - basic notations and composition rules for regular expression
  - regular expression design
  - converting regular expressions to NFA
  - converting DFA to regular expressions

- non-regular languages
  - pumping lemma (principle)
  - pumping lemma (theorem)
  - proofs for non-regular languages