COMP2111 Week 7
Term 1, 2019
Finite automata
Summary

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
- Mealy machines
Summary

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
- Mealy machines
Transition systems

A transition system (or state machine) is a pair \((S, \rightarrow)\) where \(S\) is a set and \(\rightarrow \subseteq S \times S\) is a binary relation.

**NB**

\(S\) is not necessarily finite.

Transition systems may have:

- \(\Lambda\)-labelled transitions: \(\rightarrow \subseteq S \times \Lambda \times S\)
- A start/initial state \(s_0 \in S\)
- A set of final states \(F \subseteq S\) (where runs terminate)

If \(\rightarrow\) is a function (from \(S \times \Lambda\) to \(S\)) then the transition system is **deterministic**. In general a transition system is **non-deterministic**.
Abstraction

Transition systems model computational processes *abstractly*.

We are not concerned with:

- the internal structure of states; or
- the nature of the transition relation (i.e. *why* two states are related)
Reachability and Runs

A state $s'$ is **reachable** from a state $s$ if $(s, s') \in \rightarrow^*$ (the transitive closure of $\rightarrow$).

A **run** from a state $s$ is a sequence $s_1, s_2, \ldots$ such that $s_1 = s$ and $s_i \rightarrow s_{i+1}$ for all $i$.

**NB**

*In a non-deterministic transition system there may be many (including none) runs from a state. In an unlabelled deterministic transition system there is exactly one run from every state.*
Acceptors and Transducers

An acceptor is a transition system with:
- (input-)labelled transitions
- a start/initial state
- a set of final states

A transducer is a transition system with:
- (input & output-)labelled transitions
- a start/initial state

NB

Acceptors accept/reject sequences of inputs. Transducers map sequences of inputs to sequences of outputs.
Summary

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
- Mealy machines
A deterministic finite automaton (DFA) is a deterministic, finite state acceptor.

DFAs represent “computation with finite memory”

DFAs form the backbone of most computational models
Formally, a **deterministic finite automaton (DFA)** is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states $Q = \{q_0, q_1, q_2\}$
- $\Sigma$ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states $F = \{q_1\}$
Formally, a **deterministic finite automaton (DFA)** is a tuple \((Q, \Sigma, \delta, q_0, F)\) where

- \(Q\) is a finite set of states: \(Q = \{q_0, q_1, q_2\}\)
- \(\Sigma\) is the input alphabet: \(\Sigma = \{0, 1\}\)
- \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final/accepting states: \(F = \{q_1\}\)
Formally, a **deterministic finite automaton (DFA)** is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- $\Sigma$ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$

![Deterministic Finite Automata](image)

- $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$
Deterministic Finite Automata

\[\delta(q_0, 0) = q_0\]
\[\delta(q_0, 1) = q_1\]
\[\delta(q_1, 0) = q_2\]
\[\delta(q_1, 1) = q_1\]
\[\delta(q_2, 0) = q_1\]
\[\delta(q_2, 1) = q_1\]
Deterministic Finite Automata

\[\begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
q_0 & q_0 & q_1 \\
q_1 & q_2 & q_1 \\
q_2 & q_1 & q_1 \\
\end{array}\]
Deterministic Finite Automata

Formally, a **deterministic finite automaton (DFA)** is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- $\Sigma$ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

Informally: A word defines a run in the DFA and the word is accepted if the run ends in a final state.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.

$w$: 1001
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
  - Accept if the process ends in a final state, otherwise reject.

$w$: 1001
Language of a DFA

A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.

$w$: 1001
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from \( \Sigma \) – i.e. elements of \( \Sigma^* \)

- Start in state \( q_0 \)
- Take the first symbol of \( w \)
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by \( \delta \)
  - Move to the next symbol in \( w \)
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$

Accept if the process ends in a final state, otherwise reject.

$w$: 1001
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.

$w$: 1001
Language of a DFA

A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.

$w$: 1001
A DFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat the following until there are no symbols left:
  - Based on the current state and current input symbol, transition to the appropriate state determined by $\delta$
  - Move to the next symbol in $w$
- Accept if the process ends in a final state, otherwise reject.
For a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, the language of $\mathcal{A}$, $L(\mathcal{A})$, is the set of words from $\Sigma^*$ which are accepted by $\mathcal{A}$.
Language of a DFA

For a DFA $A = (Q, \Sigma, \delta, q_0, F)$, the **language of** $A$, $L(A)$, is the set of words from $\Sigma^*$ which are accepted by $A$.

A language $L \subseteq \Sigma^*$ is **regular** if there is some DFA $A$ such that $L = L(A)$. 

$L(A) = \{1, 01, 11, 101, \ldots\}$
Language of a DFA

$L(A) = \{1, 01, 11, 101, \ldots\}$

For a DFA $A = (Q, \Sigma, \delta, q_0, F)$, the **language of** $A$, $L(A)$, is the set of words from $\Sigma^*$ which are accepted by $A$.

A language $L \subseteq \Sigma^*$ is **regular** if there is some DFA $A$ such that $L = L(A)$.
Language of a DFA: formally

Given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ we define $L_\mathcal{A} : Q \rightarrow \Sigma^*$ inductively as follows:

- If $q \in F$ then $\lambda \in L_\mathcal{A}(q)$
- If $q \xrightarrow{a} q'$ and $w \in L_\mathcal{A}(q')$ then $aw \in L_\mathcal{A}(q)$

We then define

$$L(\mathcal{A}) = L_\mathcal{A}(q_0)$$
Language of a DFA: formally

Given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ we define $L_\mathcal{A} : Q \rightarrow \Sigma^*$ inductively as follows:

- If $q \in F$ then $\lambda \in L_\mathcal{A}(q)$
- If $q \xrightarrow{a} q'$ and $w \in L_\mathcal{A}(q')$ then $aw \in L_\mathcal{A}(q)$

We then define

$$L(\mathcal{A}) = L_\mathcal{A}(q_0)$$
Examples

Example

\[ A_1 \]

\[ \begin{array}{c}
q_0 \\
\rightarrow \\
\begin{array}{c}
a \\
\rightarrow \\
q_1
\end{array}
\end{array} \]

\[ \begin{array}{c}
b \\
\rightarrow \\
\begin{array}{c}
q_0 \\
\rightarrow \\
\begin{array}{c}
a \\
\rightarrow \\
q_1
\end{array}
\end{array}
\end{array} \]

\[ L(A_1) = ? \]
Examples

Example

\[ L(A_1) = \{ w \in \{a, b\}^* : w \text{ ends with } b \} \]
Examples

Example

\[ A_2 \]

\[ L(A_2) = ? \]
Examples

Example

$$L(A_2) = \{ w \in \{a, b\}^* : w \text{ ends with } a \} \cup \{\lambda\}$$
Examples

Example

Find $A_3$ such that $L(A_3) = \emptyset$

![Diagram of $A_3$]

Find $A_4$ such that $L(A_4) = \{\lambda\}$

![Diagram of $A_4$]
Examples

Example

Find $A_3$ such that $L(A_3) = \emptyset$

Find $A_4$ such that $L(A_4) = \{\lambda\}$
Examples

Example

Find $A_3$ such that $L(A_3) = \emptyset$

Find $A_4$ such that $L(A_4) = \{\lambda\}$

\[ A_3 \]

\[ A_4 \]
Example

Find $A_5$ such that $L(A_5) = \{ w \in \{a, b\}^* : \text{every odd symbol is } b \}$.
Example

Find $A_5$ such that $L(A_5) = \{ w \in \{a, b\}^* : \text{every odd symbol is } b \}$
Example

Find $A_6$ such that
$L(A_6) = \{ w \in \{a, b\}^* : \text{second-last symbol is } b \}$
Example

Find $A_6$ such that

$L(A_6) = \{ w \in \{a, b\}^* : \text{second-last symbol is } b \}$
Example

Find $A_6$ such that $L(A_6) = \{w \in \{a, b\}^* : \text{second-last symbol is } b\}$
Summary

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
- Mealy machines
A **non-deterministic finite automaton (NFA)** is a non-deterministic, finite state acceptor.

More general than DFAs: A DFA is an NFA
Formally, a **non-deterministic finite automaton (NFA)** is a tuple $(Q, \Sigma, \delta, q_0, F)$ where
- $Q$ is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- $\Sigma$ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$
Formally, a **non-deterministic finite automaton (NFA)** is a tuple \((Q, \Sigma, \delta, q_0, F)\) where

- \(Q\) is a finite set of states: \(Q = \{q_0, q_1, q_2\}\)
- \(\Sigma\) is the input alphabet: \(\Sigma = \{0, 1\}\)
- \(\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q\) is the transition relation
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final/accepting states: \(F = \{q_1\}\)
Formally, a **non-deterministic finite automaton (NFA)** is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- $\Sigma$ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$
Non-deterministic Finite Automata

\[
\delta = \begin{cases} 
(q_0, 0, q_0), & (q_0, 1, q_0), & (q_0, 1, q_1), \\
(q_1, \epsilon, q_2), & (q_1, 0, q_2), & (q_1, 1, q_1), \\
(q_2, 0, q_1)
\end{cases}
\]
Non-deterministic Finite Automata

\[
\begin{array}{c|ccc}
\delta & \epsilon & 0 & 1 \\
\hline
q_0 & \emptyset & \{q_0\} & \{q_0, q_1\} \\
q_1 & \{q_2\} & \{q_2\} & \{q_1\} \\
q_2 & \emptyset & \{q_1\} & \emptyset \\
\end{array}
\]
Formally, a non-deterministic finite automaton (NFA) is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- $\Sigma$ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$
An NFA accepts a sequence of symbols from $\Sigma$ – i.e. elements of $\Sigma^*$

Informally: A word defines several runs in the NFA and the word is accepted if at least one run ends in a final state.

Note 1: Runs can end prematurely (these don’t count)

Note 2: An NFA will always “choose wisely”
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat until there are no symbols left or no transitions available:
  - Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
  - If not an $\epsilon$-transition, move to the next symbol in $w$
- Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000
Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:
  - Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
  - If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat until there are no symbols left or no transitions available:
  - Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
  - If not an $\epsilon$-transition, move to the next symbol in $w$
- Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

- Start in state $q_0$
- Take the first symbol of $w$
- Repeat until there are no symbols left or no transitions available:
  - Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
  - If not an $\epsilon$-transition, move to the next symbol in $w$
- Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.
Language of an NFA

Start in state $q_0$

Take the first symbol of $w$

Repeat until there are no symbols left or no transitions available:

- Based on the current state and current input symbol or $\epsilon$, transition to any state determined by $\delta$
- If not an $\epsilon$-transition, move to the next symbol in $w$

Accept if there are no symbols left and the process ends in a final state, otherwise reject.

$w$: 1000 ✓
For an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, the language of $\mathcal{A}$, $L(\mathcal{A})$, is the set of words from $\Sigma^*$ which are accepted by $\mathcal{A}$.
For an NFA $A = (Q, \Sigma, \delta, q_0, F)$, the **language of $A$, $L(A)$**, is the set of words from $\Sigma^*$ which are accepted by $A$. 

$L(A) = \{1, 01, 11, 10, \ldots\}$
Language of an NFA: formally

Given an NFA $A = (Q, \Sigma, \delta, q_0, F)$ we define $L_A : Q \rightarrow \Sigma^*$ inductively as follows:

- If $q \in F$ then $\lambda \in L_A(q)$
- If $q \overset{a}{\rightarrow} q'$ and $w \in L_A(q')$ then $aw \in L_A(q)$
- If $q \overset{\epsilon}{\rightarrow} q'$ and $w \in L_A(q')$ then $w \in L_A(q)$

We then define

$$L(A) = L_A(q_0)$$
Language of an NFA: formally

Given an NFA $A = (Q, \Sigma, \delta, q_0, F)$ we define $L_A : Q \rightarrow \Sigma^*$ inductively as follows:

1. If $q \in F$ then $\lambda \in L_A(q)$
2. If $q \xrightarrow{a} q'$ and $w \in L_A(q')$ then $aw \in L_A(q)$
3. If $q \xrightarrow{\epsilon} q'$ and $w \in L_A(q')$ then $w \in L_A(q)$

We then define

$$L(A) = L_A(q_0)$$
Example

$\mathcal{B}_1$

$L(\mathcal{B}_1) = ?$
Examples

**Example**

\[ L(B_1) = \{ w \in \{a, b\}^* : w \text{ ends with } b \} \]
Examples

Example

$$L(B_2) = ?$$
Examples

Example

$L(B_2) = \{a, b\}^*$
Examples

Example

Find $B_3$ such that $L(B_3) = \emptyset$

Find $B_4$ such that $L(B_4) = \{\lambda\}$
Examples

Example

Find $B_3$ such that $L(B_3) = \emptyset$

Find $B_4$ such that $L(B_4) = \{\lambda\}$
Examples

Example

Find $\mathcal{B}_3$ such that $L(\mathcal{B}_3) = \emptyset$

$\mathcal{B}_3$

Find $\mathcal{B}_4$ such that $L(\mathcal{B}_4) = \{\lambda\}$

$\mathcal{B}_4$
Example

Find $B_5$ such that $L(B_5) = \{w \in \{a, b\}^* : \text{second-last symbol is } b\}$
Example

Find $\mathcal{B}_5$ such that $L(\mathcal{B}_5) = \{ w \in \{a, b\}^* : \text{second-last symbol is } b \}$
NFAs vs DFAs

Clearly for any DFA $A$ there is an NFA $B$ such that $L(A) = L(B)$.

**Theorem**

For any NFA $B$ there is a DFA $A$ such that $L(A) = L(B)$.

Proof sketch: (Subset construction)

Given $B = (Q, \Sigma, \delta, q_0, F)$, construct $A = (Q', \Sigma, \delta', q'_0, F')$ as follows:

- $Q' = \text{Pow}(Q)$
- $\delta'(X, a) = \{ q' \in Q : \exists q \in X, q'' \in Q. q \xrightarrow{a} q'' \xrightarrow{*} q' \}$
- $q'_0 = \{ q_0 \}$
- $F' = \{ X \in Q' : X \cap F \neq \emptyset \}$

Intuitively: $A$ keeps track of all the possible states $B$ could be in after seeing a given sequence of symbols.
NFAs vs DFAs

Clearly for any DFA $A$ there is an NFA $B$ such that $L(A) = L(B)$.

**Theorem**

*For any NFA $B$ there is a DFA $A$ such that $L(A) = L(B)$.*

Proof sketch: (Subset construction)
Given $B = (Q, \Sigma, \delta, q_0, F)$, construct $A = (Q', \Sigma, \delta', q'_0, F')$ as follows:

- $Q' = \text{Pow}(Q)$
- $\delta'(X, a) = \{ q' \in Q : \exists q \in X, q'' \in Q. q \xrightarrow{a} q'' \xrightarrow{*} q' \}$
- $q'_0 = \{ q_0 \}$
- $F' = \{ X \in Q' : X \cap F \neq \emptyset \}$

Intuitively: $A$ keeps track of all the possible states $B$ could be in after seeing a given sequence of symbols.
NFAs vs DFAs

Clearly for any DFA $A$ there is an NFA $B$ such that $L(A) = L(B)$.

Theorem

For any NFA $B$ there is a DFA $A$ such that $L(A) = L(B)$.

Proof sketch: (Subset construction)

Given $B = (Q, \Sigma, \delta, q_0, F)$, construct $A = (Q', \Sigma, \delta', q'_0, F')$ as follows:

- $Q' = \text{Pow}(Q)$
- $\delta'(X, a) = \{ q' \in Q : \exists q \in X, q'' \in Q. q \xrightarrow{a} q'' \xrightarrow{\epsilon}^* q' \}$
- $q'_0 = \{ q_0 \}$
- $F' = \{ X \in Q' : X \cap F \neq \emptyset \}$

Intuitively: $A$ keeps track of all the possible states $B$ could be in after seeing a given sequence of symbols.
NFA to DFA Example

Example

\[ \mathcal{N}_5 \]

\[ \begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & \emptyset & \emptyset \\
\{q_0\} & \{q_0\} & \{q_0\} \\
\{q_1\} & \{q_1\} & \{q_1\} \\
\{q_2\} & \{q_2\} & \{q_2\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_1\} \\
\{q_0, q_2\} & \{q_0, q_2\} & \{q_0, q_2\} \\
\{q_1, q_2\} & \{q_1, q_2\} & \{q_1, q_2\} \\
\{q_0, q_1, q_2\} & \{q_0, q_1, q_2\} & \{q_0, q_1, q_2\} \\
\end{array} \]
NFA to DFA Example

Example

\[ B_5 \]

\[
\begin{array}{c}
\delta' \\
\emptyset \\
\{ q_0 \} \\
\{ q_1 \} \\
\{ q_2 \} \\
\{ q_0, q_1 \} \\
\{ q_0, q_2 \} \\
\{ q_1, q_2 \} \\
\{ q_0, q_1, q_2 \}
\end{array}
\begin{array}{ccc}
a \\
\{ q_0 \} \\
\{ q_2 \} \\
\{ q_0, q_1 \} \\
\{ q_0 \} \\
\{ q_2 \} \\
\{ q_0, q_1 \} \\
\{ q_0, q_2 \} \\
\{ q_0, q_1, q_2 \}
\end{array}
\begin{array}{ccc}
b \\
\{ q_0, q_1 \} \\
\{ q_2 \} \\
\{ q_0, q_1, q_2 \} \\
\{ q_0, q_2 \} \\
\{ q_2 \} \\
\{ q_0, q_1 \} \\
\{ q_0, q_2 \} \\
\{ q_0, q_1, q_2 \}
\end{array}\]
NFA to DFA Example

Example

\[\mathcal{B}_5\]  

\begin{array}{cccc}
\delta' & a, b & b & a, b \\
\emptyset & \emptyset & \emptyset & \emptyset \\
\{q_0\} & \{q_0\} & \{q_0, q_1\} & \\
\{q_1\} & \{q_2\} & \{q_2\} & \\
\{q_2\} & \emptyset & \emptyset & \\
\{q_0, q_1\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} & \\
\{q_0, q_2\} & \{q_0\} & \{q_0, q_1\} & \\
\{q_1, q_2\} & \{q_2\} & \{q_2\} & \\
\{q_0, q_1, q_2\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} & \\
\end{array}
**Example**

\[ \mathcal{B}_5 \]

\[ q_0 \quad \xrightarrow{a, b} \quad q_1 \quad \xrightarrow{a, b} \quad q_2 \]

\[
\begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & \emptyset & \emptyset \\
\{q_0\} & \{q_0\} & \{q_0, q_1\} \\
\{q_1\} & \{q_2\} & \{q_2\} \\
\{q_2\} & \emptyset & \emptyset \\
\{q_0, q_1\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} \\
\{q_0, q_2\} & \{q_0\} & \{q_0, q_1\} \\
\{q_1, q_2\} & \{q_2\} & \{q_2\} \\
\{q_0, q_1, q_2\} & \{q_0, q_2\} & \{q_0, q_1, q_2\}
\end{array}
\]
NFA to DFA Example

Example

\[ B_5 \]

\[ \begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & \emptyset & \emptyset \\
\{q_0\} & \{q_0\} & \{q_0, q_1\} \\
\{q_1\} & \{q_2\} & \{q_2\} \\
\{q_2\} & \emptyset & \emptyset \\
\{q_0, q_1\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} \\
\{q_0, q_2\} & \{q_0\} & \{q_0, q_1\} \\
\{q_1, q_2\} & \{q_2\} & \{q_2\} \\
\{q_0, q_1, q_2\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} \\
\end{array} \]
NFA to DFA Example

Example

\[B_5\]

\[\begin{array}{ccc}
q_0 & b & q_1 \\
\delta' & & a, b \\
\emptyset & \emptyset & \emptyset \\
\{q_0\} & \{q_0\} & \{q_0, q_1\} \\
\{q_1\} & \{q_2\} & \{q_2\} \\
\{q_2\} & \emptyset & \emptyset \\
\{q_0, q_1\} & \{q_0, q_2\} & \{q_0, q_1, q_2\} \\
\{q_0, q_2\} & \{q_0\} & \{q_0, q_1\} \\
\{q_1, q_2\} & \{q_2\} & \{q_2\} \\
\{q_0, q_1, q_2\} & \{q_0, q_2\} & \{q_0, q_1, q_2\}
\end{array}\]
### NFA to DFA Example

**Example**

- **States:** $B_5 = \{q_0, q_1, q_2\}$
- **Transitions:**
  - $q_0 \xrightarrow{a,b} q_1$
  - $q_1 \xrightarrow{a,b} q_2$
  - $q_0 \xrightarrow{b} q_0$

**Transition Table:**

<table>
<thead>
<tr>
<th>$\delta'$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_0}$</td>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
</tr>
<tr>
<td>${q_1}$</td>
<td>${q_2}$</td>
<td>${q_2}$</td>
</tr>
<tr>
<td>${q_2}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
</tr>
<tr>
<td>${q_1, q_2}$</td>
<td>${q_2}$</td>
<td>${q_2}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_2}$</td>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_2}$</td>
</tr>
</tbody>
</table>
NFA to DFA Example

Example

\( B_5 \)

\[ q_0 \rightarrow_{a, b} q_1 \rightarrow_{a, b} q_2 \]

\[
\begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & \emptyset & \emptyset \\
\{ q_0 \} & \{ q_0 \} & \{ q_0, q_1 \} \\
\{ q_1 \} & \{ q_2 \} & \{ q_2 \} \\
\{ q_2 \} & \emptyset & \emptyset \\
\{ q_0, q_1 \} & \{ q_0, q_2 \} & \{ q_0, q_1, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0 \} & \{ q_0, q_1 \} \\
\{ q_1, q_2 \} & \{ q_2 \} & \{ q_2 \} \\
\{ q_0, q_1, q_2 \} & \{ q_0, q_2 \} & \{ q_0, q_1, q_2 \}
\end{array}
\]
Example

NFA to DFA Example

\[ \delta' \]

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( { q_0 } )</td>
<td>( { q_0 } )</td>
<td>( { q_0, q_1 } )</td>
</tr>
<tr>
<td>( { q_1 } )</td>
<td>( { q_2 } )</td>
<td>( { q_2 } )</td>
</tr>
<tr>
<td>( { q_2 } )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_2 } )</td>
<td>( { q_0, q_1, q_2 } )</td>
</tr>
<tr>
<td>( { q_0, q_2 } )</td>
<td>( { q_0 } )</td>
<td>( { q_0, q_1 } )</td>
</tr>
<tr>
<td>( { q_1, q_2 } )</td>
<td>( { q_2 } )</td>
<td>( { q_2 } )</td>
</tr>
<tr>
<td>( { q_0, q_1, q_2 } )</td>
<td>( { q_0, q_2 } )</td>
<td>( { q_0, q_1, q_2 } )</td>
</tr>
</tbody>
</table>
NFA to DFA Example

Example

\[ \delta' \]

<table>
<thead>
<tr>
<th>State</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( { q_0 } )</td>
<td>( { q_0 } )</td>
<td>( { q_0, q_1 } )</td>
</tr>
<tr>
<td>( { q_1 } )</td>
<td>( { q_2 } )</td>
<td>( { q_2 } )</td>
</tr>
<tr>
<td>( { q_2 } )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_2 } )</td>
<td>( { q_0, q_1, q_2 } )</td>
</tr>
<tr>
<td>( { q_0, q_2 } )</td>
<td>( { q_0 } )</td>
<td>( { q_0, q_1 } )</td>
</tr>
<tr>
<td>( { q_1, q_2 } )</td>
<td>( { q_2 } )</td>
<td>( { q_2 } )</td>
</tr>
<tr>
<td>( { q_0, q_1, q_2 } )</td>
<td>( { q_0, q_2 } )</td>
<td>( { q_0, q_1, q_2 } )</td>
</tr>
</tbody>
</table>
NFA to DFA Example

Example

\[ B_5 \]

\[
\begin{array}{c}
q_0 \quad b \quad q_1 \quad a, b \quad q_2 \\
\end{array}
\]

\[
\begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & A & A \\
\{q_0\} & B & E \\
\{q_1\} & C & D \\
\{q_2\} & D & A \\
\{q_0, q_1\} & E & H \\
\{q_0, q_2\} & F & E \\
\{q_1, q_2\} & G & D \\
\{q_0, q_1, q_2\} & H & H \\
\end{array}
\]
NFA to DFA Example

Example

\[ B_5 \]

\[ \begin{array}{c}
q_0 \rightarrow b \rightarrow q_1 \rightarrow a, b \rightarrow q_2 \\
\end{array} \]

\[ \begin{array}{c|cc}
\delta' & a & b \\
\hline
\emptyset & A & A \\
\{ q_0 \} & B & E \\
\{ q_1 \} & C & D \\
\{ q_2 \} & D & A \\
\{ q_0, q_1 \} & E & F \\
\{ q_0, q_2 \} & F & E \\
\{ q_1, q_2 \} & G & D \\
\{ q_0, q_1, q_2 \} & H & F \\
\end{array} \]
NFA to DFA Example

Example

\[ B_5 \]

\[ \delta' \]

<table>
<thead>
<tr>
<th>{q_0}</th>
<th>{q_1}</th>
<th>{q_2}</th>
<th>{q_0, q_1}</th>
<th>{q_0, q_2}</th>
<th>{q_1, q_2}</th>
<th>{q_0, q_1, q_2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

\[ a, b \]

\[ q_0 \rightarrow b \rightarrow q_1 \rightarrow a, b \rightarrow q_2 \]
NFA to DFA Example

Example

Transition Table:

- $B_5$: $a, b$

Diagram:

- $q_0$ transitions to $q_1$ on $b$ and to $q_2$ on $a, b$
- $B$ transitions to $E$ on $a, b$
- $F$ transitions to $H$ on $a, b$
- $G$ transitions to $D$ on $a, b$
- $A$ transitions to itself on $a, b$

States:

- $q_0, q_1, q_2$
- $B, E, F, H, G, D, A$
NFAs vs DFAs

**Theorem**

- For any NFA with $n$ states there exists a DFA with at most $2^n$ states that accepts the same language.
- There exist NFAs with $n$ states such that the smallest DFA that accepts the same language has at least $2^n$ states.
Summary

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
- Mealy machines
Regular languages

A language $L \subseteq \Sigma^*$ is regular if there is some DFA $A$ such that $L = L(A)$.

Equivalently, there is some NFA $B$ such that $L = L(B)$. 
A language $L \subseteq \Sigma^*$ is **regular** if there is some DFA $A$ such that $L = L(A)$

Equivalently, there is some NFA $B$ such that $L = L(B)$
Non-regular languages

Are there languages which are not regular? Yes

“Simple” counting argument: there are uncountably many languages, and only countably many DFAs

An example of a non-regular language: \( \{0^n 1^n : n \in \mathbb{N}\} \)

Intuitively: need arbitrary large memory to “remember” the number of 0’s
Non-regular languages

Are there languages which are not regular? Yes

“Simple” counting argument: there are uncountably many languages, and only countably many DFAs

An example of a non-regular language: \( \{0^n1^n : n \in \mathbb{N}\} \)

Intuitively: need arbitrary large memory to “remember” the number of 0’s
Non-regular languages

Are there languages which are not regular? Yes

“Simple” counting argument: there are uncountably many languages, and only countably many DFAs

An example of a non-regular language: \( \{0^n1^n : n \in \mathbb{N}\} \)
Intuitively: need arbitrary large memory to “remember” the number of 0’s
Complementation

**Theorem**

If $L$ is a regular language then $L^c = \Sigma^* \setminus L$ is a regular language.

**Proof:**

- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that $L(A) = L$.
- Consider $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$.
- For any word $w \in \Sigma^*$, the corresponding run in $A$ is unique, so:
  - If $w \in L(A)$ then $w \notin L(A')$, and
  - If $w \notin L(A)$ then $w \in L(A')$.
- Therefore $L(A') = \Sigma^* \setminus L(A) = L^c$.

**NB**

This argument does not apply for NFAs (see $B_1$ and $B_2$).
**Complementation**

**Theorem**

If $L$ is a regular language then $L^c = \Sigma^* \setminus L$ is a regular language.

**Proof:**

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that $L(\mathcal{A}) = L$
- Consider $\mathcal{A}' = (Q, \Sigma, \delta, q_0, Q \setminus F)$
- For any word $w \in \Sigma^*$, the corresponding run in $\mathcal{A}$ is unique, so:
  - If $w \in L(\mathcal{A})$ then $w \notin L(\mathcal{A}')$, and
  - If $w \notin L(\mathcal{A})$ then $w \in L(\mathcal{A}')$,
- Therefore $L(\mathcal{A}') = \Sigma^* \setminus L(\mathcal{A}) = L^c$

**NB**

This argument does not apply for NFAs (see $B_1$ and $B_2$)
Complementation

**Theorem**

If $L$ is a regular language then $L^c = \Sigma^* \setminus L$ is a regular language.

**Proof:**

- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that $L(A) = L$
- Consider $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$
- For any word $w \in \Sigma^*$, the corresponding run in $A$ is unique, so:
  - If $w \in L(A)$ then $w \notin L(A')$, and
  - If $w \notin L(A)$ then $w \in L(A')$,
- Therefore $L(A') = \Sigma^* \setminus L(A) = L^c$

**NB**

This argument does not apply for NFAs (see $B_1$ and $B_2$)
Complementation

**Theorem**

If $L$ is a regular language then $L^c = \Sigma^* \setminus L$ is a regular language.

**Proof:**

- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that $L(A) = L$
- Consider $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$
- For any word $w \in \Sigma^*$, the corresponding run in $A$ is unique, so:
  - If $w \in L(A)$ then $w \notin L(A')$, and
  - If $w \notin L(A)$ then $w \in L(A')$,
- Therefore $L(A') = \Sigma^* \setminus L(A) = L^c$

**NB**

This argument does not apply for NFAs (see $B_1$ and $B_2$)
Complementation

**Theorem**

If \( L \) is a regular language then \( L^c = \Sigma^* \setminus L \) is a regular language.

**Proof:**

- Let \( A = (Q, \Sigma, \delta, q_0, F) \) be a DFA such that \( L(A) = L \)
- Consider \( A' = (Q, \Sigma, \delta, q_0, Q \setminus F) \)
- For any word \( w \in \Sigma^* \), the corresponding run in \( A \) is unique, so:
  - If \( w \in L(A) \) then \( w \notin L(A') \), and
  - If \( w \notin L(A) \) then \( w \in L(A') \),
- Therefore \( L(A') = \Sigma^* \setminus L(A) = L^c \)

**NB**

This argument does not apply for NFAs (see \( B_1 \) and \( B_2 \))
Theorem
If $L$ is a regular language then $L^c = \Sigma^* \setminus L$ is a regular language.

Proof:
- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that $L(A) = L$
- Consider $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$
- For any word $w \in \Sigma^*$, the corresponding run in $A$ is unique, so:
  - If $w \in L(A)$ then $w \notin L(A')$, and
  - If $w \notin L(A)$ then $w \in L(A')$,
- Therefore $L(A') = \Sigma^* \setminus L(A) = L^c$

NB
This argument does not apply for NFAs (see $B_1$ and $B_2$)
**Theorem**

*If* $L_1$ *and* $L_2$ *are regular languages, then* $L_1 \cup L_2$ *is regular.*

**Proof:**

- Let $B_1$ and $B_2$ be NFAs such that $L(B_1) = L_1$ and $L(B_2) = L_2$.
- Construct an NFA $B$ by having a new start state with $\epsilon$-transitions to the start states of $B_1$ and $B_2$.
- Consider $w \in L_1 \cup L_2$:
  - If $w \in L_1$ then there is a run in $B_1$, and hence in $B$, which ends in a final state.
  - If $w \in L_2$ then there is a run in $B_2$, and hence in $B$, which ends in a final state.
- In either case $w \in L(B)$.
- Conversely, any accepting run in $B$ will be either an accepting run in $B_1$ or in $B_2$; so if $w \in L(B)$ then $w \in L_1 \cup L_2$. 
**Theorem**

If $L_1$ and $L_2$ are regular languages, then $L_1 \cup L_2$ is regular.

**Proof:**

- Let $B_1$ and $B_2$ be NFAs such that $L(B_1) = L_1$ and $L(B_2) = L_2$
- Construct an NFA $B$ by having a new start state with $\epsilon$-transitions to the start states of $B_1$ and $B_2$
- Consider $w \in L_1 \cup L_2$:
  - If $w \in L_1$ then there is a run in $B_1$, and hence in $B$, which ends in a final state
  - If $w \in L_2$ then there is a run in $B_2$, and hence in $B$, which ends in a final state
  - In either case $w \in L(B)$
- Conversely, any accepting run in $B$ will be either an accepting run in $B_1$ or in $B_2$; so if $w \in L(B)$ then $w \in L_1 \cup L_2$
If \( L_1 \) and \( L_2 \) are regular languages, then \( L_1 \cup L_2 \) is regular.

Proof:

- Let \( B_1 \) and \( B_2 \) be NFAs such that \( L(B_1) = L_1 \) and \( L(B_2) = L_2 \).
- Construct an NFA \( B \) by having a new start state with \( \epsilon \)-transitions to the start states of \( B_1 \) and \( B_2 \).
- Consider \( w \in L_1 \cup L_2 \):
  - If \( w \in L_1 \) then there is a run in \( B_1 \), and hence in \( B \), which ends in a final state.
  - If \( w \in L_2 \) then there is a run in \( B_2 \), and hence in \( B \), which ends in a final state.
  - In either case, \( w \in L(B) \).
- Conversely, any accepting run in \( B \) will be either an accepting run in \( B_1 \) or in \( B_2 \); so if \( w \in L(B) \) then \( w \in L_1 \cup L_2 \).
Theorem

If \( L_1 \) and \( L_2 \) are regular languages, then \( L_1 \cup L_2 \) is regular.

Proof:

- Let \( B_1 \) and \( B_2 \) be NFAs such that \( L(B_1) = L_1 \) and \( L(B_2) = L_2 \).
- Construct an NFA \( B \) by having a new start state with \( \epsilon \)-transitions to the start states of \( B_1 \) and \( B_2 \).
- Consider \( w \in L_1 \cup L_2 \):
  - If \( w \in L_1 \) then there is a run in \( B_1 \), and hence in \( B \), which ends in a final state.
  - If \( w \in L_2 \) then there is a run in \( B_2 \), and hence in \( B \), which ends in a final state.
  - In either case \( w \in L(B) \).
- Conversely, any accepting run in \( B \) will be either an accepting run in \( B_1 \) or in \( B_2 \); so if \( w \in L(B) \) then \( w \in L_1 \cup L_2 \).
Union

**Theorem**

*If* $L_1$ *and* $L_2$ *are regular languages, then* $L_1 \cup L_2$ *is regular.*

**Proof:**

- Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be NFAs such that $L(\mathcal{B}_1) = L_1$ and $L(\mathcal{B}_2) = L_2$.
- Construct an NFA $\mathcal{B}$ by having a new start state with $\epsilon$-transitions to the start states of $\mathcal{B}_1$ and $\mathcal{B}_2$.
- Consider $w \in L_1 \cup L_2$:
  - If $w \in L_1$ then there is a run in $\mathcal{B}_1$, and hence in $\mathcal{B}$, which ends in a final state.
  - If $w \in L_2$ then there is a run in $\mathcal{B}_2$, and hence in $\mathcal{B}$, which ends in a final state.
  - In either case $w \in L(\mathcal{B})$.
- Conversely, any accepting run in $\mathcal{B}$ will be either an accepting run in $\mathcal{B}_1$ or in $\mathcal{B}_2$; so if $w \in L(\mathcal{B})$ then $w \in L_1 \cup L_2$. 
Intersection

Theorem

If $L_1$ and $L_2$ are regular languages, then $L_1 \cap L_2$ is regular.

Proof:

\[ L_1 \cap L_2 = (L_1^c \cup L_2^c)^c \]
Intersection

**Theorem**

*If $L_1$ and $L_2$ are regular languages, then $L_1 \cap L_2$ is regular.*

Proof:

$$L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$$
Concatenation

Recall for languages $X$ and $Y$: $X \cdot Y = \{xy : x \in X, y \in Y\}$

**Theorem**

*If $L_1$ and $L_2$ are regular languages, then $L_1 \cdot L_2$ is regular.*

**Proof:**

- Let $B_1$ and $B_2$ be NFAs such that $L(B_1) = L_1$ and $L(B_2) = L_2$.
- Construct an NFA $B$ by adding $\epsilon$-transitions from the final states of $B_1$ to the start state of $B_2$. Let the start state of $B$ be the start state of $B_1$; and let the final states of $B$ be the final states of $B_2$.
- Any word in $L_1 \cdot L_2$ can be written as $wv$ with $w \in L_1$ and $v \in L_2$. $w$ has an accepting run in $B_1$ and $v$ has an accepting run in $B_2$, so $wv$ has an accepting run in $B$.
- Conversely, any word $w$ with an accepting run in $B$ can be broken up into an accepting run in $B_1$ followed by an accepting run in $B_2$. Thus $w$ can be broken up into two words $w = xy$ where $x \in L_1$ and $y \in L_2$. 
**Concatenation**

Recall for languages \( X \) and \( Y \): 
\[
X \cdot Y = \{xy : x \in X, y \in Y\}
\]

**Theorem**

If \( L_1 \) and \( L_2 \) are regular languages, then \( L_1 \cdot L_2 \) is regular.

**Proof:**

- Let \( B_1 \) and \( B_2 \) be NFAs such that \( L(B_1) = L_1 \) and \( L(B_2) = L_2 \).
- Construct an NFA \( B \) by adding \( \epsilon \)-transitions from the final states of \( B_1 \) to the start state of \( B_2 \). Let the start state of \( B \) be the start state of \( B_1 \); and let the final states of \( B \) be the final states of \( B_2 \).
- Any word in \( L_1 \cdot L_2 \) can be written as \( wv \) with \( w \in L_1 \) and \( v \in L_2 \). \( w \) has an accepting run in \( B_1 \) and \( v \) has an accepting run in \( B_2 \), so \( wv \) has an accepting run in \( B \).
- Conversely, any word \( w \) with an accepting run in \( B \) can be broken up into an accepting run in \( B_1 \) followed by an accepting run in \( B_2 \). Thus \( w \) can be broken up into two words \( w = xy \) where \( x \in L_1 \) and \( y \in L_2 \).
**Concatenation**

Recall for languages $X$ and $Y$: $X \cdot Y = \{ xy : x \in X, y \in Y \}$

---

**Theorem**

*If $L_1$ and $L_2$ are regular languages, then $L_1 \cdot L_2$ is regular.*

---

**Proof:**

- Let $B_1$ and $B_2$ be NFAs such that $L(B_1) = L_1$ and $L(B_2) = L_2$.
- Construct an NFA $B$ by adding $\epsilon$-transitions from the final states of $B_1$ to the start state of $B_2$. Let the start state of $B$ be the start state of $B_1$; and let the final states of $B$ be the final states of $B_2$.
- Any word in $L_1 \cdot L_2$ can be written as $wv$ with $w \in L_1$ and $v \in L_2$. $w$ has an accepting run in $B_1$ and $v$ has an accepting run in $B_2$, so $wv$ has an accepting run in $B$.
- Conversely, any word $w$ with an accepting run in $B$ can be broken up into an accepting run in $B_1$ followed by an accepting run in $B_2$. Thus $w$ can be broken up into two words $w = xy$ where $x \in L_1$ and $y \in L_2$. 
Concatenation

Recall for languages $X$ and $Y$: $X \cdot Y = \{xy : x \in X, y \in Y\}$

**Theorem**

If $L_1$ and $L_2$ are regular languages, then $L_1 \cdot L_2$ is regular.

**Proof:**

- Let $B_1$ and $B_2$ be NFAs such that $L(B_1) = L_1$ and $L(B_2) = L_2$.
- Construct an NFA $B$ by adding $\epsilon$-transitions from the final states of $B_1$ to the start state of $B_2$. Let the start state of $B$ be the start state of $B_1$; and let the final states of $B$ be the final states of $B_2$.
- Any word in $L_1 \cdot L_2$ can be written as $wv$ with $w \in L_1$ and $v \in L_2$. $w$ has an accepting run in $B_1$ and $v$ has an accepting run in $B_2$, so $wv$ has an accepting run in $B$.
- Conversely, any word $w$ with an accepting run in $B$ can be broken up into an accepting run in $B_1$ followed by an accepting run in $B_2$. Thus $w$ can be broken up into two words $w = xy$ where $x \in L_1$ and $y \in L_2$. 


**Concatenation**

Recall for languages $X$ and $Y$: $X \cdot Y = \{xy : x \in X, y \in Y\}$

**Theorem**

If $L_1$ and $L_2$ are regular languages, then $L_1 \cdot L_2$ is regular.

**Proof:**

- Let $B_1$ and $B_2$ be NFAs such that $L(B_1) = L_1$ and $L(B_2) = L_2$.
- Construct an NFA $B$ by adding $\epsilon$-transitions from the final states of $B_1$ to the start state of $B_2$. Let the start state of $B$ be the start state of $B_1$; and let the final states of $B$ be the final states of $B_2$.
- Any word in $L_1 \cdot L_2$ can be written as $wv$ with $w \in L_1$ and $v \in L_2$. $w$ has an accepting run in $B_1$ and $v$ has an accepting run in $B_2$, so $wv$ has an accepting run in $B$.
- Conversely, any word $w$ with an accepting run in $B$ can be broken up into an accepting run in $B_1$ followed by an accepting run in $B_2$. Thus $w$ can be broken up into two words $w = xy$ where $x \in L_1$ and $y \in L_2$. 
Kleene star

Recall for a language \( X \):
\[ X^* = \{ w : w \text{ can be made up from 0 or more words in } X \} \]

**Theorem**

If \( L \) is regular languages, then \( L^* \) is regular.

**Proof:**

- Let \( B \) be an NFA such that \( L(B) = L \).
- Construct an NFA \( B' \) by:
  - creating a new start state which is accepting;
  - adding an \( \epsilon \)-transition from the new start state to the start state of \( B \);
  - adding \( \epsilon \)-transitions from the final states of \( B \) to the new start state.
- Similar arguments as before show that \( L(B') = L(B)^* \).
Kleene star

Recall for a language $X$:

$$X^* = \{ w : w \text{ can be made up from 0 or more words in } X \}$$

**Theorem**

*If $L$ is regular languages, then $L^*$ is regular.*

**Proof:**

- Let $B$ be an NFA such that $L(B) = L$
- Construct an NFA $B'$ by:
  - creating a new start state which is accepting;
  - adding an $\epsilon$-transition from the new start state to the start state of $B$;
  - adding $\epsilon$-transitions from the final states of $B$ to the new start state.
- Similar arguments as before show that $L(B') = L(B)^*$
Kleene star

Recall for a language $X$:

\[ X^* = \{ w : w \text{ can be made up from 0 or more words in } X \} \]

**Theorem**

If $L$ is regular languages, then $L^*$ is regular.

**Proof:**

- Let $B$ be an NFA such that $L(B) = L$
- Construct an NFA $B'$ by:
  - creating a new start state which is accepting;
  - adding an $\varepsilon$-transition from the new start state to the start state of $B$
  - adding $\varepsilon$-transitions from the final states of $B$ to the new start state.

  Similar arguments as before show that $L(B') = L(B)^*$
Kleene star

Recall for a language $X$:

$X^* = \{ w : w \text{ can be made up from 0 or more words in } X \}$

**Theorem**

*If $L$ is regular languages, then $L^*$ is regular.*

**Proof:**

- Let $B$ be an NFA such that $L(B) = L$
- Construct an NFA $B'$ by:
  - creating a new start state which is accepting;
  - adding an $\epsilon$-transition from the new start state to the start state of $B$
  - adding $\epsilon$-transitions from the final states of $B$ to the new start state.

- Similar arguments as before show that $L(B') = L(B)^*$
Regular operations

Concatenation, union, and Kleene star are collectively known as the regular operations.

Recall:
The definition of a program in $\mathcal{L}^+$:

$$P ::= (x := e) \mid \varnothing \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$
Regular operations

Concatenation, union, and Kleene star are collectively known as the regular operations.

Recall:
The definition of a program in $L^+$:

$$P ::= (x := e) \mid \varnothing \mid P_1; P_2 \mid P_1 + P_2 \mid P_1^*$$
Summary

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
- Mealy machines
Regular expressions

Given a finite set $\Sigma$, a regular expression over $\Sigma$ (RE) is defined recursively as follows:

- $\emptyset$ is a regular expression
- $\epsilon$ is a regular expression
- $a$ is a regular expression for all $a \in \Sigma$
- If $E_1$ and $E_2$ are regular expressions, then $E_1E_2$ is a regular expression
- If $E_1$ and $E_2$ are regular expressions, then $E_1 + E_2$ is a regular expression
- If $E$ is a regular expression, then $E^*$ is a regular expression

We use parentheses to disambiguate REs, though $*$ binds tighter than concatenation, which binds tighter than $+$. 
Examples

Example

The following are regular expressions over $\Sigma = \{0, 1\}$:

- $\emptyset$
- $101 + 010$
- $(\epsilon + 10)^*01$