COMP2111 Week 7
Term 1, 2023
Finite automata
Summary

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

- Recap
- Deterministic Finite Automata
- Non-deterministic Finite Automata
- Regular languages
- Regular expressions
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:

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

If \(\rightarrow\) is a partial function (from \(S \times L\) to \(S\)), the transition system is deterministic. If \(\rightarrow\) is a function, the transition system is total.
Reachability and Runs

A state $s'$ is **reachable** from a state $s$ if $(s, s') \in \rightarrow^*$ (the reflexive and 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 (or no) runs from a state. In an unlabelled deterministic transition system there is exactly one maximal 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
A **deterministic finite automaton (DFA)** is a total, finite state acceptor.

DFAs represent “computation with finite memory”

DFAs are simple, easy to work with and show up all over the place.
Formally, a deterministic finite automaton (DFA) is a tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states
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
- **\(\delta : Q \times \Sigma \rightarrow Q\)** is the transition function
- **\(q_0 \in Q\)** is the start state
- **\(F \subseteq Q\)** is the set of final/accepting states
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
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

\[ \delta \]

\[
\begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
q_0 & q_0 & q_1 \\
q_1 & q_2 & q_1 \\
q_2 & q_1 & q_1 \\
\end{array}
\]
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 $\Sigma^*$
Language of a DFA

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

- Start in state $q_0$

$w$: 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 $w$
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 $\delta$

A DFA accepts the sequence $w: 1001$
Language of a DFA

\[ q_0 \rightarrow 0 \rightarrow q_0 \]
\[ q_0 \rightarrow 1 \rightarrow q_1 \]
\[ q_1 \rightarrow 0 \rightarrow q_2 \]
\[ q_1 \rightarrow 1 \rightarrow q_1 \]

**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 \( \delta \)
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 \( w \)
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 \( w \)
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 $w$
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 $w$

$w$: 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 $w$
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 $w$

$w$: 1001
Language of a DFA

\[ w: 1001 \]

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 \( w \)
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.
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

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

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 $A$. 
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 $\mathcal{A}$

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

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ we define $L_A : Q \to \Sigma^*$ inductively as follows:

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

\[
A_1
\]

\[
L(A_1) = ?
\]
Example

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

$A_2$

$q_0$  $q_1$

$a$  $b$

$b$  $a$

$L(A_2) = ?$
Example

$A_2$

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

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

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

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

$A_3$

$q_0$

$a, b$

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

Example

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

$A_3$

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

$A_4$
Example

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

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

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

**Example**

Find $A_6$ such that

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

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
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
- $\Sigma$ is the input alphabet
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states
**Non-deterministic Finite Automata**

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
- \(\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q\) is the transition relation
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final/accepting states
Non-deterministic Finite Automata

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
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

\[
\begin{array}{c|ccc}
\delta & \epsilon & 0 & 1 \\
\hline
q_0 & \emptyset & \{q_0\} & \{q_0, q_1\} \\
q_1 & \{q_2\} & \{q_2\} & \{q_1\} \\
q_2 & \emptyset & \{q_1\} & \emptyset \\
\end{array}
\]
Non-deterministic Finite Automata

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

For each symbol $c$ of $w$:
- Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$-transitions from the coloured states, and uncolour all other states.

Accept if there are no symbols left and a final state is coloured; otherwise, reject.

$w$: 1000
Language of an NFA

- Colour the state $q_0$

$w$: 1000
Language of an NFA

\[ w: 1000 \]

- Colour the state \( q_0 \)
- Colour states reachable by one or more \( \epsilon \) transitions from \( q_0 \).
- For each symbol \( c \) of \( w \):
  - Colour all states reachable by a \( c \)-transition followed by 0 or more \( \epsilon \) transitions from the coloured states, and uncolour all other states.
**Language of an NFA**

- Colour the state $q_0$
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.

$w: \text{1000}$
Language of an NFA

1 1

$w$: 1000

- Colour the state $q_0$
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.
Language of an NFA

$w$: 1000

- Colour the state $q_0$
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.
Language of an NFA

Colour the state $q_0$

Colour states reachable by one or more $\epsilon$ transitions from $q_0$.

For each symbol $c$ of $w$:

- Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.
Language of an NFA

![Diagram of an NFA]

- **Colour the state** $q_0$
- **Colour states reachable by one or more $\epsilon$ transitions from** $q_0$.
- **For each symbol** $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.

$w$: 1000
Language of an NFA

![Diagram of an NFA]

**$w$: 1000**

- Colour the state $q_0$.
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.

Accept if there are no symbols left and a final state is coloured; otherwise, reject.
Language of an NFA

1

1

1

0, σ

0

q₀

q₁

q₂

w: 1000

- Colour the state q₀
- Colour states reachable by one or more ϵ transitions from q₀.
- For each symbol c of w:
  - Colour all states reachable by a c-transition followed by 0 or more ϵ transitions from the coloured states, and uncolour all other states.
Language of an NFA

$w$: 1000

- Colour the state $q_0$
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.
Language of an NFA

- Colour the state \( q_0 \)
- Colour states reachable by one or more \( \epsilon \) transitions from \( q_0 \).
- For each symbol \( c \) of \( w \):
  - Colour all states reachable by a \( c \)-transition followed by 0 or more \( \epsilon \) transitions from the coloured states, and uncolour all other states.

\( w: 1000 \)
Language of an NFA

$w: 1000$

- Colour the state $q_0$
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.
Language of an NFA

Colour the state $q_0$

Colour states reachable by one or more $\epsilon$ transitions from $q_0$.

For each symbol $c$ of $w$:

- Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.

$w: 1000$
Language of an NFA

\[ w: 1000 \]

- Colour the state \( q_0 \)
- Colour states reachable by one or more \( \epsilon \) transitions from \( q_0 \).
- For each symbol \( c \) of \( w \):
  - Colour all states reachable by a \( c \)-transition followed by 0 or more \( \epsilon \) transitions from the coloured states, and uncolour all other states.
- Accept if there are no symbols left and a final state is coloured; otherwise, reject.
Language of an NFA

- Colour the state $q_0$
- Colour states reachable by one or more $\epsilon$ transitions from $q_0$.
- For each symbol $c$ of $w$:
  - Colour all states reachable by a $c$-transition followed by 0 or more $\epsilon$ transitions from the coloured states, and uncolour all other states.
- Accept if there are no symbols left and a final state is coloured; 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 \( \mathcal{A} = (Q, \Sigma, \delta, q_0, F) \), the language of \( \mathcal{A} \), \( L(\mathcal{A}) \), is the set of words from \( \Sigma^* \) which are accepted by \( \mathcal{A} \).
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 $q \xrightarrow{a} q'$ and $w \in L_A(q')$ then $aw \in L_A(q)$
- If $q \xrightarrow{\epsilon} q'$ and $w \in L_A(q')$ then $w \in L_A(q)$

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

Given an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ 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 $aw \in L_\mathcal{A}(q)$
- If $q \xrightarrow{\epsilon} q'$ and $w \in L_\mathcal{A}(q')$ then $w \in L_\mathcal{A}(q)$

We then define

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

$B_1$

$q_0 \xrightarrow[a,b]{\quad} q_0$

$q_0 \xrightarrow[b]{\quad} q_1$

$L(B_1) = ?$
Example

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

Example

$B_2$

$q_0 \xrightarrow{a,b} q_0 \xrightarrow{b} q_1$

$L(B_2) = ?$
Examples

Example

$B_2 \xrightarrow{a, b} q_0 \xrightarrow{b} q_1$

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

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

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

Example

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

$\mathcal{B}_3$

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

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

$B_3 

\rightarrow q_0$

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

$B_4

\rightarrow q_0$
Example

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

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

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 $L(A) = L(B)$.*
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:

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

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

NFA to DFA Example

Example

\[ B_5 \]

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

Example

\[ \delta' \]

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

Example

\[ \mathcal{B}_5 \]

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

Example

\[ B_5 \]

\[
\begin{array}{ccc}
q_0 & b & q_1 \\
q_1 & a, b & q_2 \\
\end{array}
\]

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

Example

\[ \mathcal{B}_5 \]

\[
\begin{array}{ccc}
q_0 & b & q_1 \\
\rightarrow & b & \rightarrow \\
q_0 & \rightarrow & q_1 \\
\end{array}
\]

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

Example

Example

\[ B_5 \]

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

Example

\[ \mathcal{B}_5 \]

\[ q_0 \quad b \quad q_1 \quad a, b \quad q_2 \]

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

Example

\[ \mathcal{B}_5 \]

\[ q_0 \] \quad b \quad \[ q_1 \] \quad a, b \quad \[ q_2 \]

<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_0 } ]</td>
<td>[ { q_0, q_1 } ]</td>
</tr>
<tr>
<td>[ { q_0, q_1, q_2 } ]</td>
<td>[ { q_0, q_1, q_2 } ]</td>
<td>[ { q_0, q_1, q_2 } ]</td>
</tr>
</tbody>
</table>
NFA to DFA Example

Example

\[ B_5 \]

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

Example

$B_5$

$\delta'$

<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

\[ \mathcal{B}_5 \]

\[
\begin{array}{ccc}
q_0 & \xrightarrow{b} & q_1 & \xrightarrow{a, b} & q_2 \\
\end{array}
\]

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

Example

\[ \delta' \]

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>( { q_0 } )</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>( { q_1 } )</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>( { q_2 } )</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>( { q_0, q_1 } )</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>( { q_0, q_2 } )</td>
<td>F</td>
<td>B</td>
</tr>
<tr>
<td>( { q_1, q_2 } )</td>
<td>G</td>
<td>D</td>
</tr>
<tr>
<td>( { q_0, q_1, q_2 } )</td>
<td>H</td>
<td>F</td>
</tr>
</tbody>
</table>
NFA to DFA Example

Example

\[ \delta' \]

\begin{tabular}{c|cc}
   & \( a \) & \( b \) \\
\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{tabular}

Diagram:

- \( B_5 \)
- States: \( q_0, q_1, q_2 \)
- Transitions:
  - \( a, b \) from \( q_0 \) to \( q_1 \)
  - \( b \) from \( q_1 \) to \( q_2 \)
NFA to DFA Example

Example

\[ \mathcal{B}_5 \]

- Transition from \( q_0 \) to \( q_1 \) on \( b \)
- Transition from \( q_1 \) to \( q_2 \) on \( a, b \)
- Transition from \( q_0 \) to \( q_2 \) on \( a, b \)

States:
- \( B \)
- \( F \)
- \( H \)
- \( E \)
- \( G \)
- \( C \)
- \( A \)
- \( D \)

Transitions:
- From \( B \) to \( E \) on \( a \)
- From \( B \) to \( F \) on \( b \)
- From \( F \) to \( B \) on \( a \)
- From \( F \) to \( H \) on \( b \)
- From \( H \) to \( B \) on \( a \)
- From \( H \) to \( F \) on \( b \)
- From \( E \) to \( B \) on \( a \)
- From \( E \) to \( H \) on \( b \)

Final States:
- \( B \)
- \( C \)
- \( A \)
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
A language \( L \subseteq \Sigma^* \) is regular if there is some DFA \( \mathcal{A} \) such that \( L = L(\mathcal{A}) \).
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?

"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.
Are there languages which are not regular? Yes

“Simple” counting argument: there are uncountably many languages, and only countably many DFAs
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 \( \mathcal{B}_1 \) and \( \mathcal{B}_2 \))
Union

**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* \( w \in L_1 \cup L_2 \).
Intersection

**Theorem**

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

Proof:
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 $y \in L_2$. 
Kleene star

Recall for a language $X$:
$X^* = \{w : w \text{ is the concatenation of 0 or more words in } X\}$

**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.
Summary

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

Regular expressions are a way of describing “finite automaton” patterns:

- Second-last letter is \textit{b}
- Every odd symbol is \textit{b}

Many applications in CS:

- Lexical analysis in compiler construction
- Search facilities provided by text editors and databases; utilities such as \texttt{grep} and \texttt{awk}
- Pattern matching on strings
Regular expressions

Given a finite set $\Sigma$, a regular expression (regexp) over $\Sigma$ 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 $E_1 E_2$ 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 regexps, though $\ast$ binds tighter than concatenation, which binds tighter than $\pm$. 
Examples

Example

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

- $\emptyset$
- $101 + 010$
- $(\epsilon + 10)^*01$
A regexp defines a language over $\Sigma$: the set of words which “match” the expression:

- Concatenation = sequences of expressions
- Union = choice of expressions
- Star = 0 or more occurrences of an expression

**Example**

The following words match $(000 + 10)^*01$:

- 01
- 101001
- 000101000001
Language of a Regular Expression

Formally, given a regexp, $E$, over $\Sigma$, we define $L(E) \subseteq \Sigma^*$ recursively as follows:

- If $E = \emptyset$ then $L(E) = \emptyset$
- If $E = \epsilon$ then $L(E) = \{\lambda\}$
- If $E = a$ where $a \in \Sigma$ then $L(E) = \{a\}$
- If $E = E_1E_2$, then $L(E) = L(E_1) \cdot L(E_2)$
- If $E = E_1 + E_2$, then $L(E) = L(E_1) \cup L(E_2)$
- If $E = E_1^*$ then $L(E) = (L(E_1))^*$

Example

$L(010 + 101) =$?

$L((\epsilon + 10)^*01) =$?
Language of a Regular Expression

Formally, given a regexp, $E$, over $\Sigma$, we define $L(E) \subseteq \Sigma^*$ recursively as follows:

- If $E = \emptyset$ then $L(E) = \emptyset$
- If $E = \epsilon$ then $L(E) = \{\lambda\}$
- If $E = a$ where $a \in \Sigma$ then $L(E) = \{a\}$
- If $E = E_1E_2$, then $L(E) = L(E_1) \cdot L(E_2)$
- If $E = E_1 + E_2$, then $L(E) = L(E_1) \cup L(E_2)$
- If $E = E_1^*$ then $L(E) = (L(E_1))^*$

Example

$L(010 + 101) = \{010, 101\}$

$L((\epsilon + 10)^*01) = ?$
Language of a Regular Expression

Formally, given a regexp, $E$, over $\Sigma$, we define $L(E) \subseteq \Sigma^*$ recursively as follows:

- If $E = \emptyset$ then $L(E) = \emptyset$
- If $E = \epsilon$ then $L(E) = \{\lambda\}$
- If $E = a$ where $a \in \Sigma$ then $L(E) = \{a\}$
- If $E = E_1E_2$, then $L(E) = L(E_1) \cdot L(E_2)$
- If $E = E_1 + E_2$, then $L(E) = L(E_1) \cup L(E_2)$
- If $E = E_1^*$ then $L(E) = (L(E_1))^*$

Example

$L(010 + 101) = \{010, 101\}$

$L((\epsilon + 10)^*01) = \{01, 1001, 101001, \ldots\}$
Regular expressions vs NfAs

**Theorem (Kleene’s theorem)**

- For any regular expression $E$, $L(E)$ is a regular language.
- For any regular language $L$, there is a regular expression $E$ such that $L = L(E)$
Proof of Kleene’s theorem

Given $E$, $L(E)$ is a regular language. Proof by induction on $E$. 
Proof of Kleene’s theorem

Given $E$, $L(E)$ is a regular language. Proof by induction on $E$.

Given $L$, find $E$ such that $L = L(E)$

- Let
  $$L^X_{q,q'} = \{ w \in \Sigma^* : q \xrightarrow{w}^* q' \text{ with all intermediate states in } X \}$$

- Define $E^X_{q,q'}$ such that $L(E^X_{q,q'}) = L^X_{q,q'}$:
  - When $q = q'$: $E^\emptyset_{q,q'} = \epsilon + a_1 + a_2 + \ldots + a_k$ where $q \xrightarrow{a_i} q$
  - When $q \neq q'$: $E^\emptyset_{q,q'} = \emptyset + a_1 + a_2 + \ldots + a_k$ where $q \xrightarrow{a_i} q'$
  - For $X \neq \emptyset$:
    $$E^X_{q,q'} = \underbrace{E^X_{q,q'} - \{ r \}}_{(1)} + \underbrace{E^X_{q,r} \cdot (E^X_{r,r} - \{ r \})^* \cdot E^X_{r,q'}}_{(2)}$$

- The required expression is then $E = \sum_{q \in F} E^Q_{q_0,q}$