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:

- Λ-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

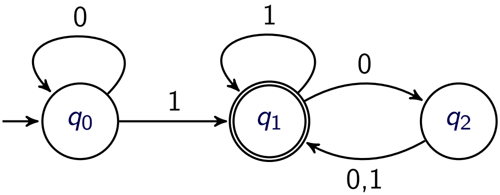
NB

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

Deterministic Finite Automata

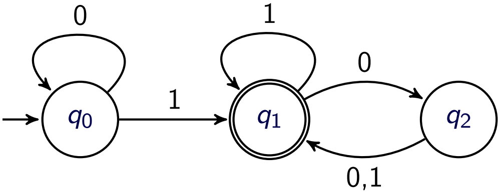


A **deterministic finite automaton (DFA)** is a deterministic, finite state acceptor.

DFAs represent "computation with finite memory"

DFAs form the backbone of most computational models

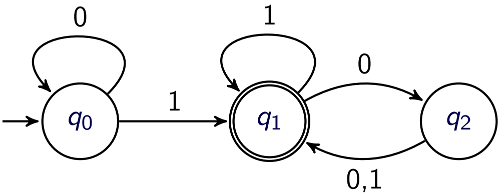
Deterministic Finite Automata



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

- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ 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: $F = \{q_1\}$

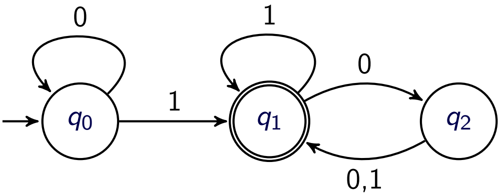
Deterministic Finite Automata



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

- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ 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: $F = \{q_1\}$

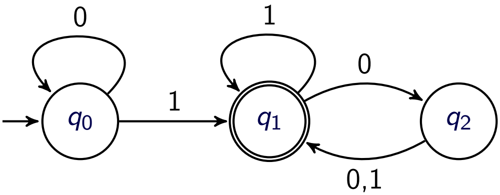
Deterministic Finite Automata



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

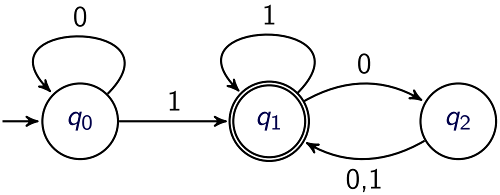
- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$

Deterministic Finite Automata



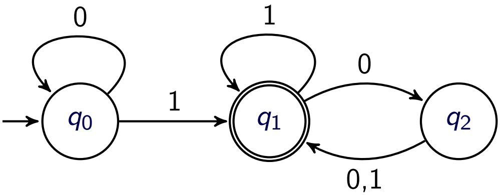
$$\delta(q_0, 0) = q_0$$
$$\delta(q_0, 1) = q_1$$
$$\delta(q_1, 0) = q_2$$
$$\delta(q_1, 1) = q_1$$
$$\delta(q_2, 0) = q_1$$
$$\delta(q_2, 1) = q_1$$

Deterministic Finite Automata



δ	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

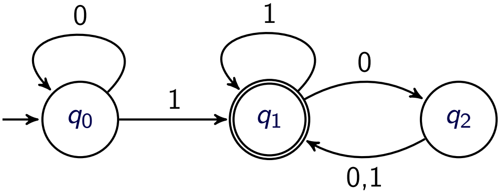
Deterministic Finite Automata



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

- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ 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: $F = \{q_1\}$

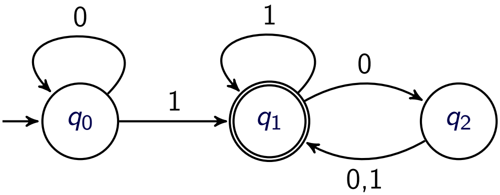
Language of a DFA



A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

Informally: A word defines a run in the DFA and the word is accepted if the run ends in a final state.

Language of a DFA

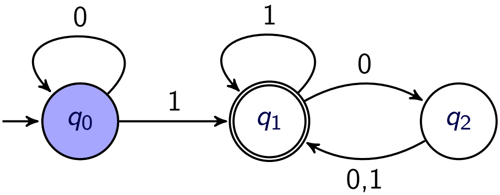


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

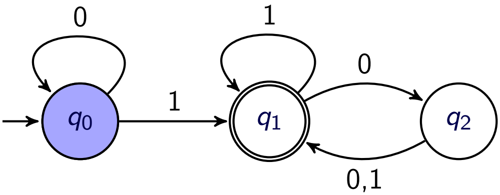


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

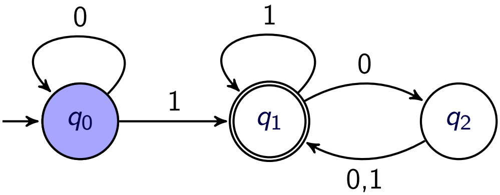


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

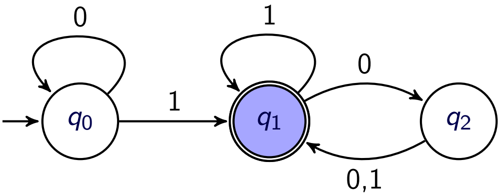


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

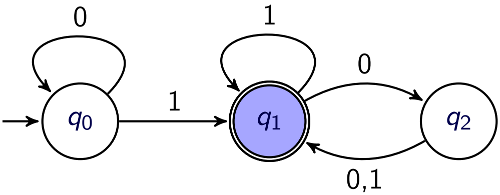


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

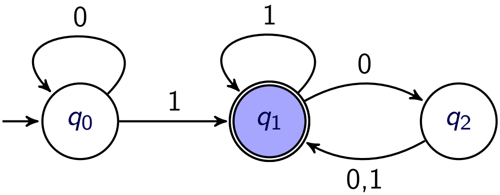


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

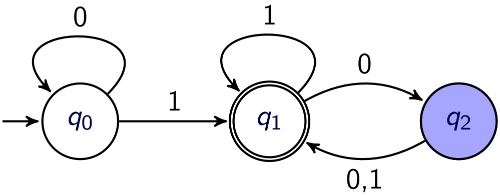


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

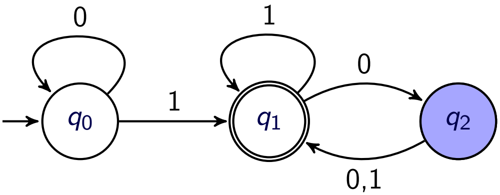


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

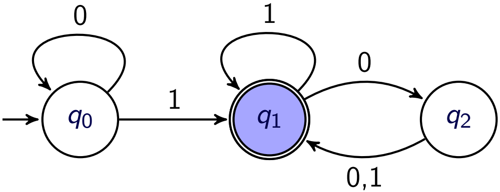


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

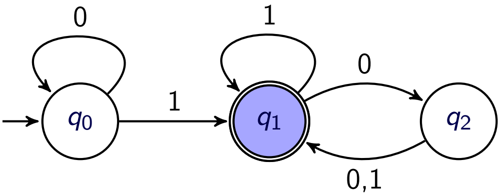


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

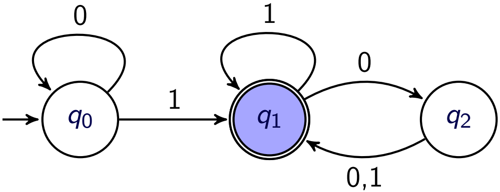


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
    - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

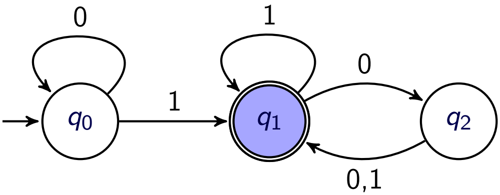


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

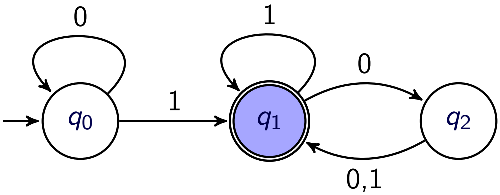


w: 1001

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

Language of a DFA

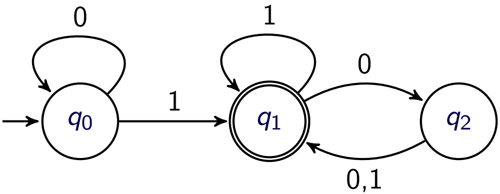


w: 1001 ✓

A DFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

- 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 δ
  - Move to the next symbol in w
- Accept if the process ends in a final state, otherwise reject.

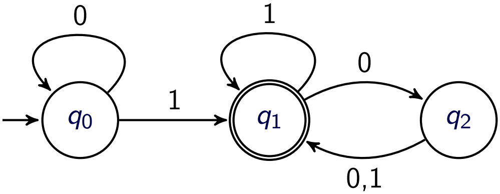
Language of a DFA



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 Σ^* 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

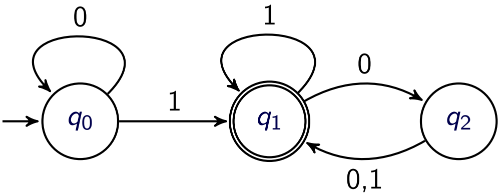


$$L(\mathcal{A}) = \{1, 01, 11, 101, \ldots\}$$

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 Σ^* 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



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

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

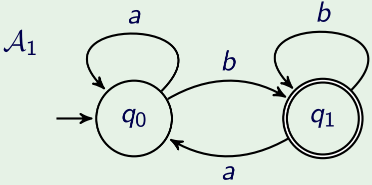
- If $q \in F$ then $\lambda \in L_\mathcal{A}(q)$
- If $q \xrightarrow{a} q'$ and $w \in L_\mathcal{A}(q')$ then $aw \in L_\mathcal{A}(q)$

We then define

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

Examples

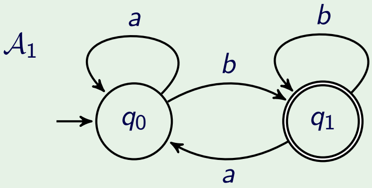
Example



$$L(\mathcal{A}_1) = ?$$

Examples

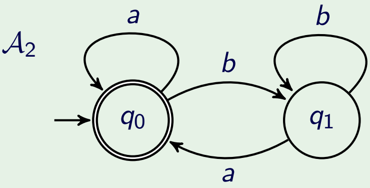
Example



$$L(\mathcal{A}_1) = \{w \in \{a, b\}^* \ : \ w \text{ ends with } b\}$$

Examples

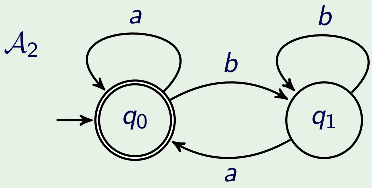
Example



$$L(\mathcal{A}_2) = ?$$

Examples

Example



$$L(\mathcal{A}_2) = \{w \in \{a, b\}^* \ : \ w \text{ ends with } a\} \cup \{\lambda\}$$

Examples

Example

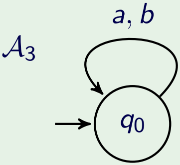
Find $\mathcal{A}_3$ such that $L(\mathcal{A}_3) = \emptyset$

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

Examples

Example

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

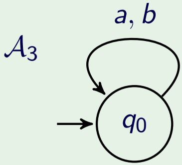


Find $\mathcal{A}_4$ such that $L(\mathcal{A}_4) = \{\lambda\}$

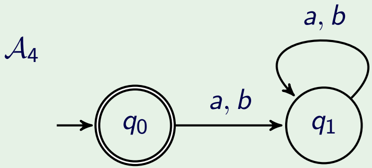
Examples

Example

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



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



Examples

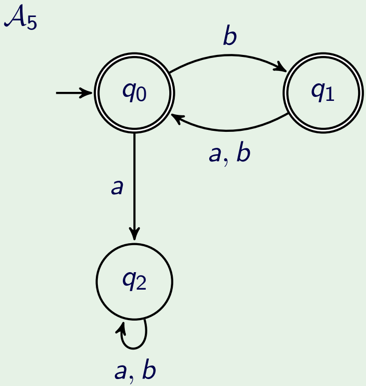
Example

Find $\mathcal{A}_5$ such that $L(\mathcal{A}_5) = \{w \in \{a, b\}^* \ : \ \text{every odd symbol is } b\}$

Examples

Example

Find $\mathcal{A}_5$ such that $L(\mathcal{A}_5) = \{w \in \{a, b\}^* \ : \ \text{every odd symbol is } b\}$



Examples

Example

Find $\mathcal{A}_6$ such that

$L(\mathcal{A}_6) = \{w \in \{a, b\}^* : \text{second-last symbol is } b\}$

Examples

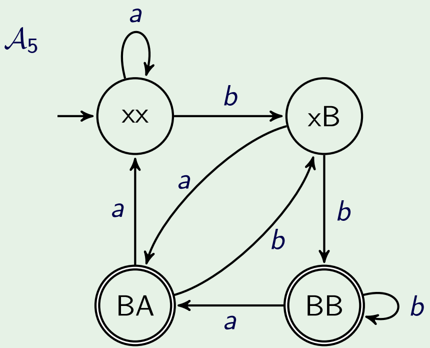
Example

Find $\mathcal{A}_6$ such that
$L(\mathcal{A}_6) = \{w \in \{a, b\}^* : \text{second-last symbol is } b\}$

Examples

Example

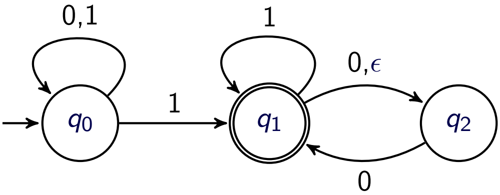
Find $\mathcal{A}_6$ such that
$L(\mathcal{A}_6) = \{w \in \{a, b\}^* : \text{second-last symbol is } b\}$



Summary

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

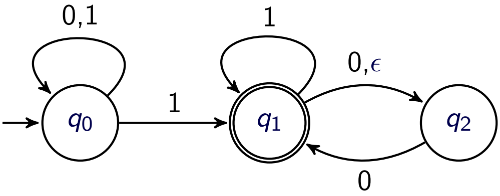
Non-deterministic Finite Automata



A **non-deterministic finite automaton (NFA)** is a non-deterministic, finite state acceptor.

More general than DFAs: A DFA is an NFA

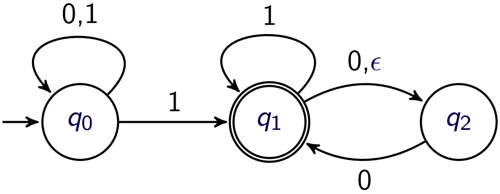
Non-deterministic Finite Automata



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

- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ 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: $F = \{q_1\}$

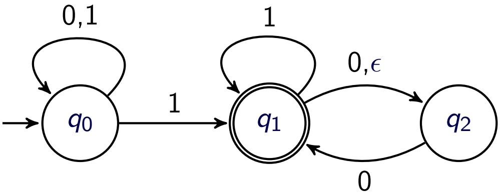
Non-deterministic Finite Automata



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

- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ 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: $F = \{q_1\}$

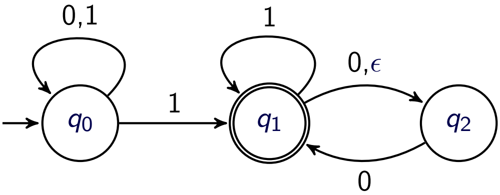
Non-deterministic Finite Automata



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

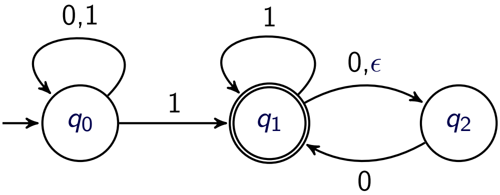
- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ is the input alphabet: $\Sigma = \{0, 1\}$
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of final/accepting states: $F = \{q_1\}$

Non-deterministic Finite Automata



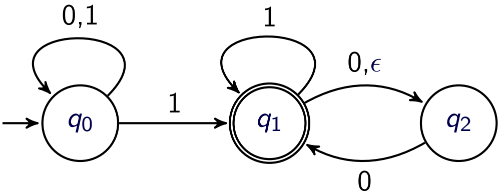
$$\delta = \left\{ \begin{array}{lll} (q_0, 0, q_0), & (q_0, 1, q_0), & (q_0, 1, q_1), \\ (q_1, \epsilon, q_2), & (q_1, 0, q_2), & (q_1, 1, q_1), \\ & (q_2, 0, q_1) & \end{array} \right\}$$

Non-deterministic Finite Automata



δ	ϵ	0	1
q_0	$\emptyset$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$	$\{q_1\}$
q_2	$\emptyset$	$\{q_1\}$	$\emptyset$

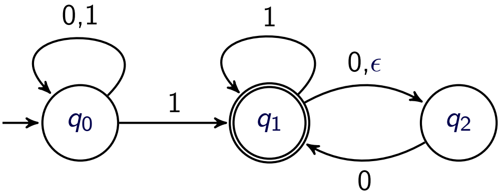
Non-deterministic Finite Automata



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

- Q is a finite set of states: $Q = \{q_0, q_1, q_2\}$
- Σ 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: $F = \{q_1\}$

Language of an NFA



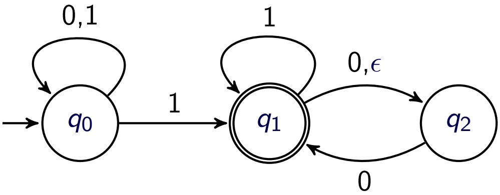
An NFA accepts a sequence of symbols from Σ – i.e. elements of Σ^*

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"

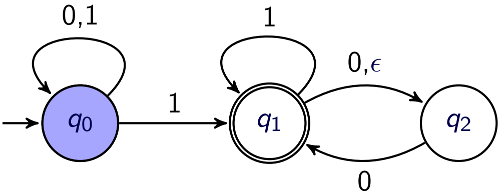
Language of an NFA



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 ϵ, transition to any state determined by δ
    - If not an ϵ-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.

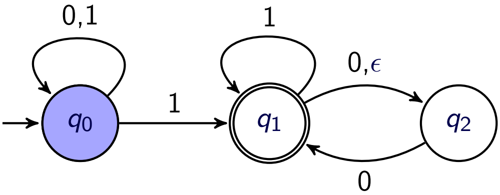
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

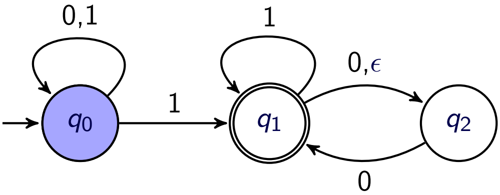
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

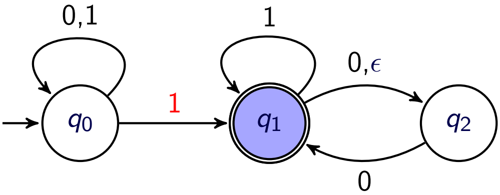
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

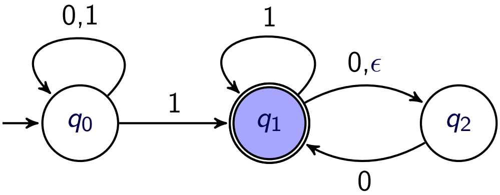
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

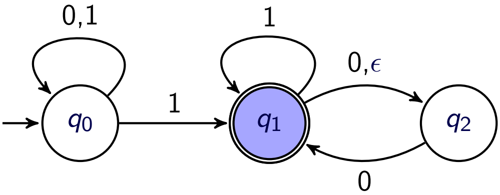
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

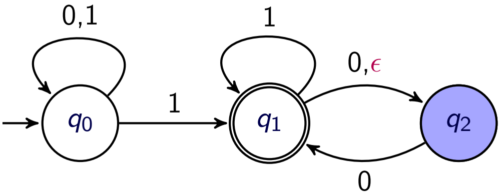
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

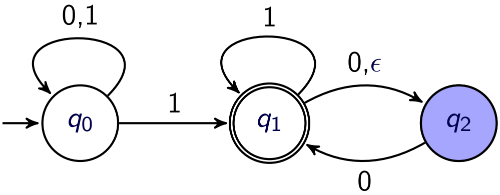
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

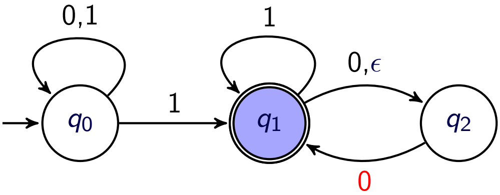
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

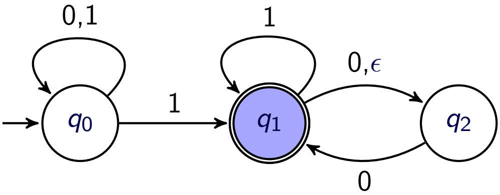
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

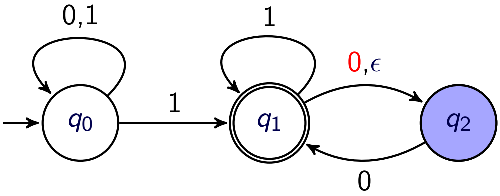
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

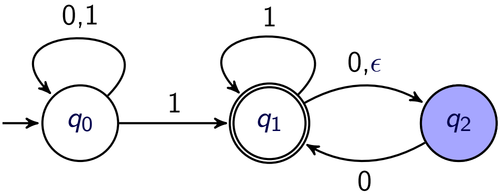
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

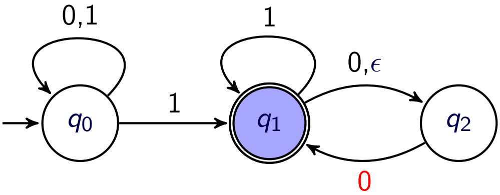
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

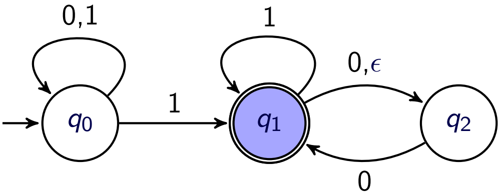
Language of an NFA



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 ϵ, transition to any state determined by δ
  - If not an ϵ-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.

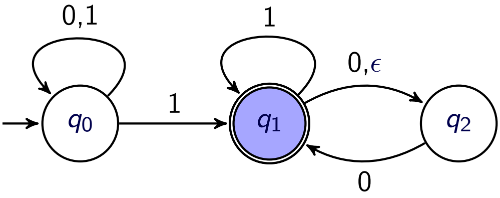
Language of an NFA



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 ϵ, transition to any state determined by δ
    - If not an ϵ-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.

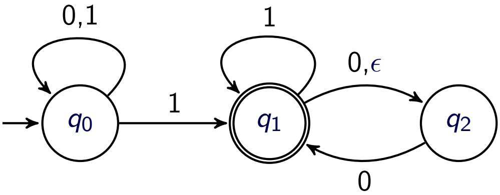
Language of an NFA



w: 1000 ✓

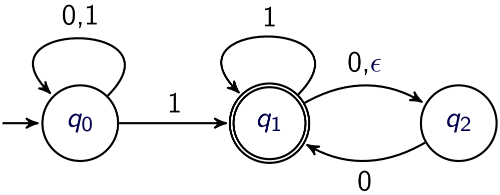
- Start in state q_0
- Take the first symbol of w
- Repeat until there are no symbols left or no transitions available:
  - Based on the current state and current input symbol or ϵ, transition to any state determined by δ
  - If not an ϵ-transition, move to the next symbol in w
- Accept if there are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

Language of an NFA



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 Σ^* which are accepted by $\mathcal{A}$

Language of an NFA



$$L(\mathcal{A}) = \{1, 01, 11, 10, \ldots\}$$

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 Σ^* which are accepted by $\mathcal{A}$

Language of an NFA: formally

Given an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ we define $L_\mathcal{A} : Q \to \Sigma^*$ 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

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

Language of an NFA: formally

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

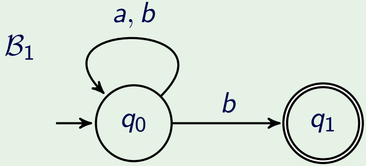
- If $q \in F$ then $\lambda \in L_{\mathcal{A}}(q)$
- If $q \xrightarrow{a} q'$ and $w \in L_{\mathcal{A}}(q')$ then $aw \in L_{\mathcal{A}}(q)$
- If $q \xrightarrow{\epsilon} q'$ and $w \in L_{\mathcal{A}}(q')$ then $w \in L_{\mathcal{A}}(q)$

We then define

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

Examples

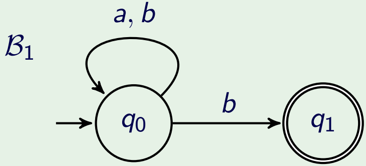
Example



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

Examples

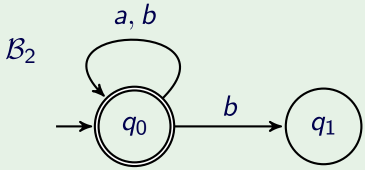
Example



$$L(\mathcal{B}_1) = \{w \in \{a, b\}^* \; : \; w \text{ ends with } b\}$$

Examples

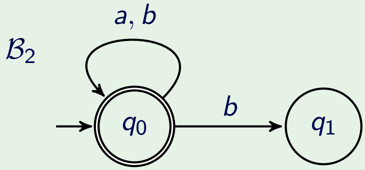
Example



$$L(\mathcal{B}_2) = ?$$

Examples

Example



$$L(\mathcal{B}_2) = \{a, b\}^*$$

Examples

Example

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

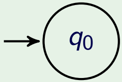
Find $\mathcal{B}_4$ such that $L(\mathcal{B}_4) = \{\lambda\}$

Examples

Example

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

$\mathcal{B}_3$



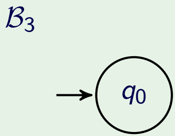
Find $\mathcal{B}_4$ such that $L(\mathcal{B}_4) = \{\lambda\}$

$\mathcal{B}_4$

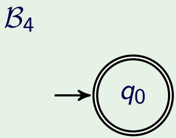
Examples

Example

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



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



Examples

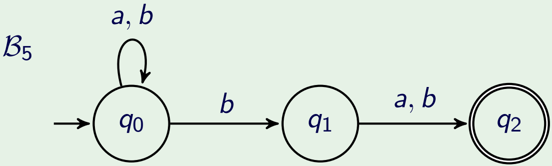
Example

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

Examples

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 $\mathcal{A}$ there is an NFA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.

Theorem

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

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

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

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

NFAs vs DFAs

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

Theorem

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

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

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

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

NFAs vs DFAs

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

Theorem

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

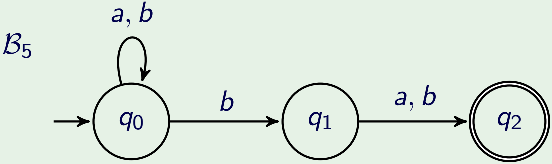
Proof sketch: (Subset construction)
Given $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$, construct $\mathcal{A} = (Q', \Sigma, \delta', q_0', F')$ as follows:

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

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

NFA to DFA Example

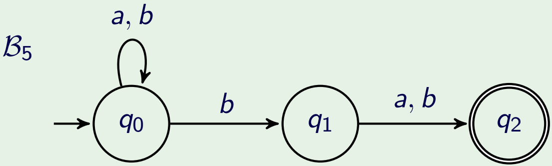
Example



δ'	a	b
$\emptyset$		
$\{q_0\}$		
$\{q_1\}$		
$\{q_2\}$		
$\{q_0, q_1\}$		
$\{q_0, q_2\}$		
$\{q_1, q_2\}$		
$\{q_0, q_1, q_2\}$		

NFA to DFA Example

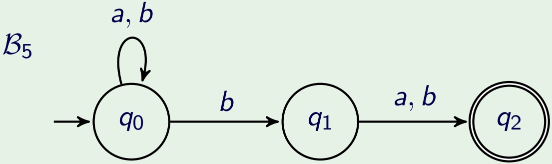
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

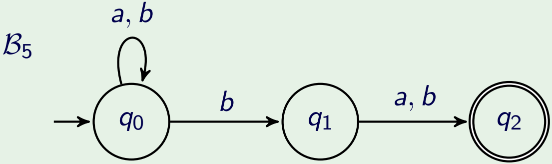
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

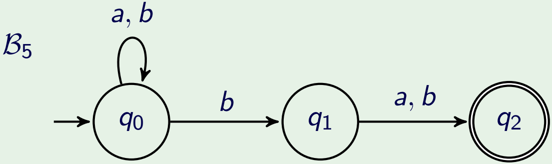
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

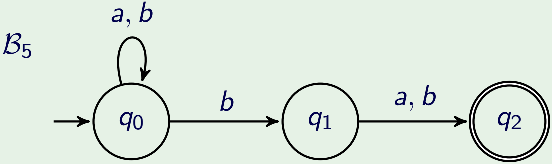
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

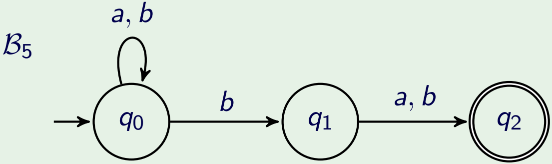
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

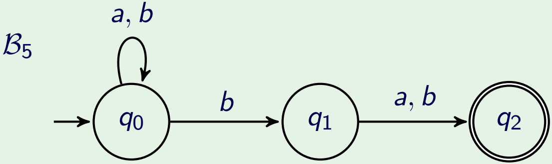
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

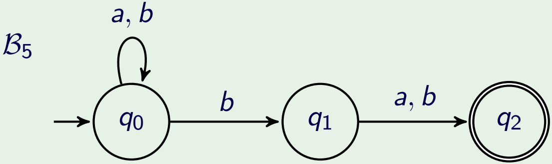
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

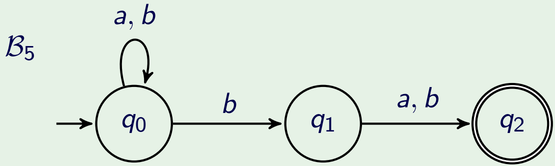
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

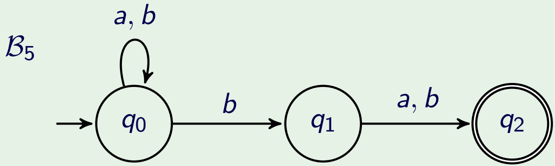
Example



δ'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

NFA to DFA Example

Example

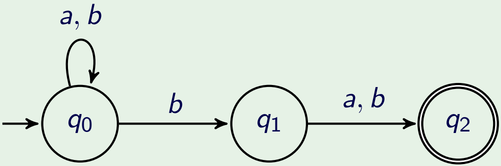


δ'		a	b
$\emptyset$	A	A	A
$\{q_0\}$	B	B	E
$\{q_1\}$	C	D	D
$\{q_2\}$	D	A	A
$\{q_0, q_1\}$	E	F	H
$\{q_0, q_2\}$	F	B	E
$\{q_1, q_2\}$	G	D	D
$\{q_0, q_1, q_2\}$	H	F	H

NFA to DFA Example

Example

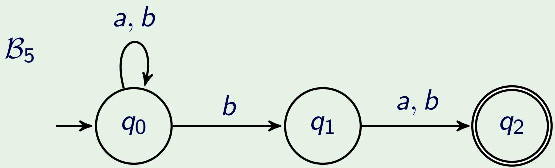
$\mathcal{B}_5$



δ'		a	b
$\emptyset$	A	A	A
$\{q_0\}$	B	B	E
$\{q_1\}$	C	D	D
$\{q_2\}$	D	A	A
$\{q_0, q_1\}$	E	F	H
$\{q_0, q_2\}$	F	B	E
$\{q_1, q_2\}$	G	D	D
$\{q_0, q_1, q_2\}$	H	F	H

NFA to DFA Example

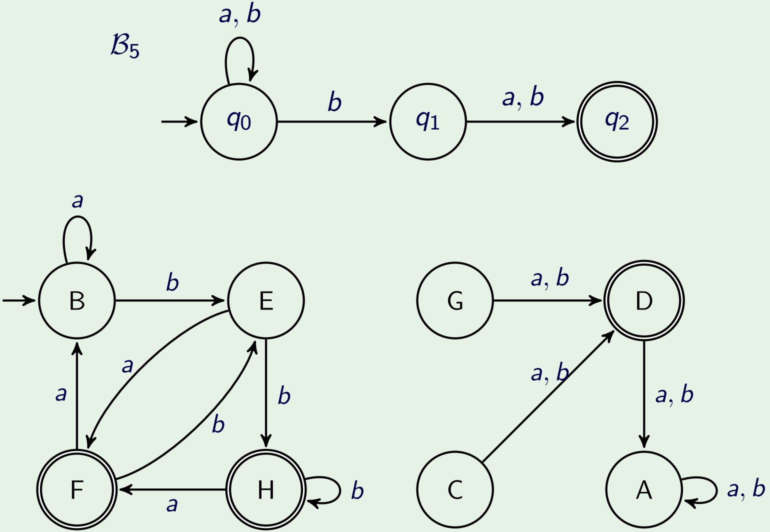
Example



δ'		a	b
$\emptyset$	A	A	A
$\{q_0\}$	B	B	E
$\{q_1\}$	C	D	D
$\{q_2\}$	D	A	A
$\{q_0, q_1\}$	E	F	H
$\{q_0, q_2\}$	F	B	E
$\{q_1, q_2\}$	G	D	D
$\{q_0, q_1, q_2\}$	H	F	H

NFA to DFA Example

Example



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

Equivalently, there is some NFA $\mathcal{B}$ such that $L = L(\mathcal{B})$

Regular languages

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

Equivalently, there is some NFA $\mathcal{B}$ such that $L = L(\mathcal{B})$

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: $\{0^n 1^n \ : \ n \in \mathbb{N}\}$
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: $\{0^n 1^n \ : \ n \in \mathbb{N}\}$
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 $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ 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 $\mathcal{B}_1$ and $\mathcal{B}_2$)

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 $\mathcal{B}_1$ and $\mathcal{B}_2$)

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 $\mathcal{B}_1$ and $\mathcal{B}_2$)

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 $\mathcal{B}_1$ and $\mathcal{B}_2$)

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 $\mathcal{B}_1$ and $\mathcal{B}_2$)

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 $\mathcal{B}_1$ 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 $\mathcal{B}_1$ 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 ϵ-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 $w \in L_1 \cup L_2$

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 ϵ-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 $w \in L_1 \cup L_2$

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 ϵ-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 $w \in L_1 \cup L_2$

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 ϵ-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 $w \in L_1 \cup L_2$

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 ϵ-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 $\mathcal{B}_1$ 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 adding ϵ-transitions from the final states of $\mathcal{B}_1$ to the start state of $\mathcal{B}_2$. Let the start state of $\mathcal{B}$ be the start state of $\mathcal{B}_1$; and let the final states of $\mathcal{B}$ be the final states of $\mathcal{B}_2$.
- 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 $\mathcal{B}_1$ and v has an accepting run in $\mathcal{B}_2$, so wv has an accepting run in $\mathcal{B}$.
- Conversely, any word w with an accepting run in $\mathcal{B}$ can be broken up into an accepting run in $\mathcal{B}_1$ followed by an accepting run in $\mathcal{B}_2$. 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 $\mathcal{B}_1$ 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 adding ϵ-transitions from the final states of $\mathcal{B}_1$ to the start state of $\mathcal{B}_2$. Let the start state of $\mathcal{B}$ be the start state of $\mathcal{B}_1$; and let the final states of $\mathcal{B}$ be the final states of $\mathcal{B}_2$.
- 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 $\mathcal{B}_1$ and v has an accepting run in $\mathcal{B}_2$, so wv has an accepting run in $\mathcal{B}$.
- Conversely, any word w with an accepting run in $\mathcal{B}$ can be broken up into an accepting run in $\mathcal{B}_1$ followed by an accepting run in $\mathcal{B}_2$. 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 $\mathcal{B}_1$ 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 adding ϵ-transitions from the final states of $\mathcal{B}_1$ to the start state of $\mathcal{B}_2$. Let the start state of $\mathcal{B}$ be the start state of $\mathcal{B}_1$; and let the final states of $\mathcal{B}$ be the final states of $\mathcal{B}_2$.
- 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 $\mathcal{B}_1$ and v has an accepting run in $\mathcal{B}_2$, so wv has an accepting run in $\mathcal{B}$.
- Conversely, any word w with an accepting run in $\mathcal{B}$ can be broken up into an accepting run in $\mathcal{B}_1$ followed by an accepting run in $\mathcal{B}_2$. 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 $\mathcal{B}_1$ 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 adding ϵ-transitions from the final states of $\mathcal{B}_1$ to the start state of $\mathcal{B}_2$. Let the start state of $\mathcal{B}$ be the start state of $\mathcal{B}_1$; and let the final states of $\mathcal{B}$ be the final states of $\mathcal{B}_2$.
- 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 $\mathcal{B}_1$ and v has an accepting run in $\mathcal{B}_2$, so wv has an accepting run in $\mathcal{B}$.
- Conversely, any word w with an accepting run in $\mathcal{B}$ can be broken up into an accepting run in $\mathcal{B}_1$ followed by an accepting run in $\mathcal{B}_2$. 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 $\mathcal{B}_1$ 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 adding ϵ-transitions from the final states of $\mathcal{B}_1$ to the start state of $\mathcal{B}_2$. Let the start state of $\mathcal{B}$ be the start state of $\mathcal{B}_1$; and let the final states of $\mathcal{B}$ be the final states of $\mathcal{B}_2$.
- 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 $\mathcal{B}_1$ and v has an accepting run in $\mathcal{B}_2$, so wv has an accepting run in $\mathcal{B}$.
- Conversely, any word w with an accepting run in $\mathcal{B}$ can be broken up into an accepting run in $\mathcal{B}_1$ followed by an accepting run in $\mathcal{B}_2$. 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 $\mathcal{B}$ be an NFA such that $L(\mathcal{B}) = L$
- Construct an NFA $\mathcal{B}'$ by:
  - creating a new start state which is accepting;
  - adding an ϵ-transition from the new start state to the start state of $\mathcal{B}$
  - adding ϵ-transitions from the final states of $\mathcal{B}$ to the new start state.
- Similar arguments as before show that $L(\mathcal{B}') = L(\mathcal{B})^*$

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

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

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

Regular operations

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

Recall:

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

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

Regular operations

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

Recall:

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

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

Summary

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

Regular expressions

Given a finite set Σ, a **regular expression over Σ (RE)** is defined recursively as follows:

- $\emptyset$ is a regular expression
- ϵ 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 REs, though $*$ binds tighter than concatenation, which binds tighter than $+$.

Examples

Example

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

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