Deterministic Finite Automata (DFAs)

Lecture 3

September 4, 2018

Part I

DFA Introduction

DFAs also called Finite State Machines (FSMs)

- The “simplest” model for computers?
- State machines that are very common in practice.
  - Vending machines
  - Elevators
  - Digital watches
  - Simple network protocols
- Programs with fixed memory

A simple program

Program to check if a given input string w has odd length

```
int n = 0
While input is not finished
    read next character c
    n ← n + 1
endWhile
If (n is odd) output YES
Else output NO
```

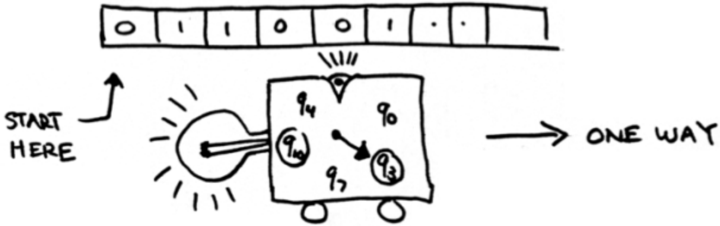
A simple program

Program to check if a given input string w has odd length

```
int n = 0
While input is not finished
    read next character c
    n ← n + 1
endWhile
If (n is odd) output YES
Else output NO
```

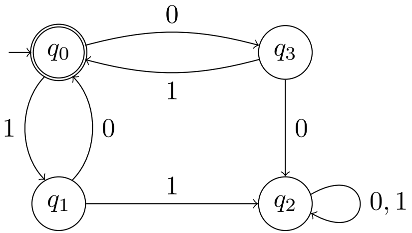
```
bit x = 0
While input is not finished
    read next character c
    x ← flip(x)
endWhile
If (x = 1) output YES
Else output NO
```

Another view



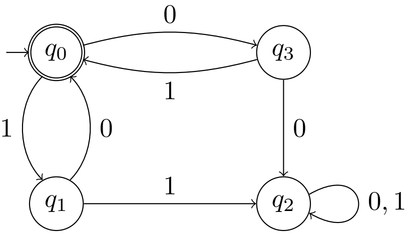
- Machine has input written on a <u>read-only</u> tape
- Start in specified start state
- Start at left, scan symbol, change state and move right
- Circled states are <u>accepting</u>
- Machine <u>accepts</u> input string if it is in an accepting state after scanning the last symbol.

Graphical Representation/State Machine



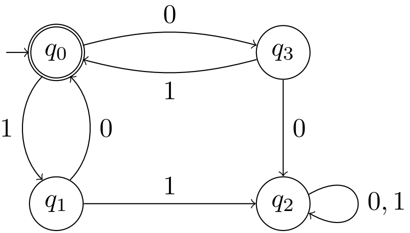
- Directed graph with nodes representing states and edge/arcs representing transitions labeled by symbols in Σ
- For each state (vertex) q and symbol $a \in \Sigma$ there is exactly one outgoing edge labeled by a
- Initial/start state has a pointer (or labeled as s, q_0 or "start")
- Some states with double circles labeled as accepting/final states

Graphical Representation



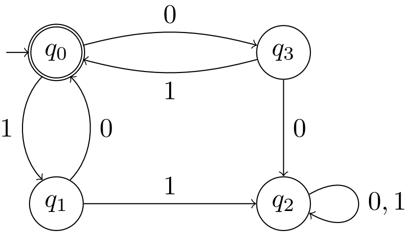
- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **10010**?

Graphical Representation



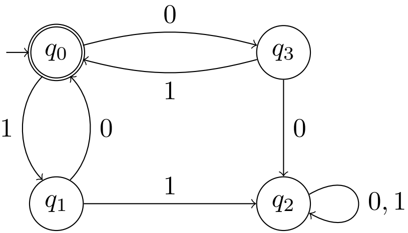
- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **10010**?
- Which strings end up in accepting state?

Graphical Representation



- **Convention:** Machine reads symbols from left to right
- Where does **001** lead? **10010**?
- Which strings end up in accepting state?
- Every string w has a unique walk that it follows from a given state q by reading one letter of w from left to right.

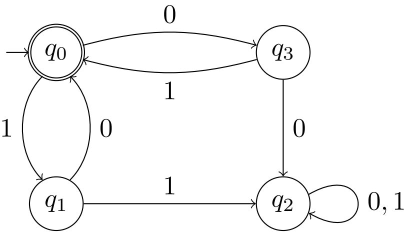
Graphical Representation



Definition

A DFA M accepts a string w iff the unique walk starting at the start state and spelling out w ends in an accepting state.

Graphical Representation



Definition

A DFA M accepts a string w iff the unique walk starting at the start state and spelling out w ends in an accepting state.

Definition

The language accepted (or recognized) by a DFA M is denote by $L(M)$ and defined as: $L(M) = \{w \mid M \text{ accepts } w\}$.

Warning

"M accepts language L" does not mean simply that that M accepts each string in L.

It means that M accepts each string in L and no others. Equivalently M accepts each string in L and does not accept/rejects strings in $\Sigma^* \setminus L$.

Warning

"M accepts language L" does not mean simply that that M accepts each string in L.

It means that M accepts each string in L and no others. Equivalently M accepts each string in L and does not accept/rejects strings in $\Sigma^* \setminus L$.

M "recognizes" L is a better term but "accepts" is widely accepted (and recognized) (joke attributed to Lenny Pitt)

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $s \in Q$ is the start state,

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

Formal Tuple Notation

Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

Formal Tuple Notation

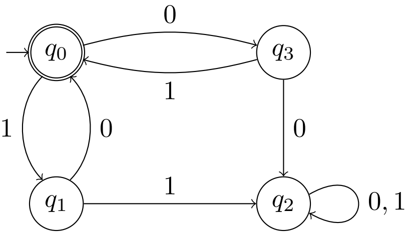
Definition

A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

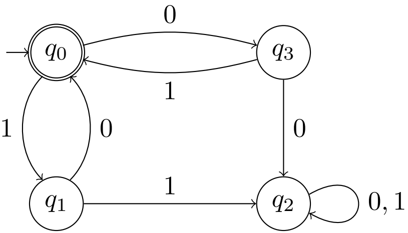
Common alternate notation: q_0 for start state, F for final states.

Example



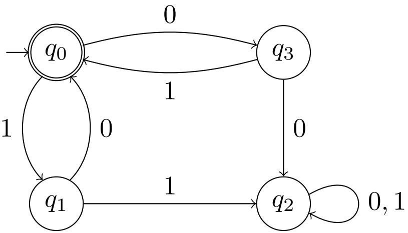
- $Q =$

Example



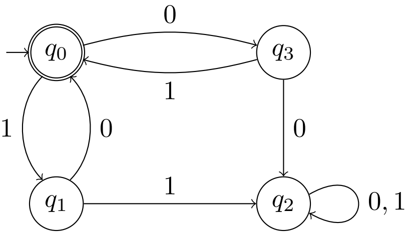
- $Q = \{q_0, q_1, q_1, q_3\}$

Example



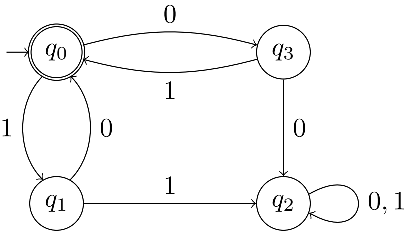
- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma =$

Example



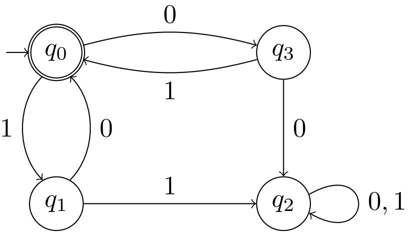
- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma = \{0, 1\}$

Example



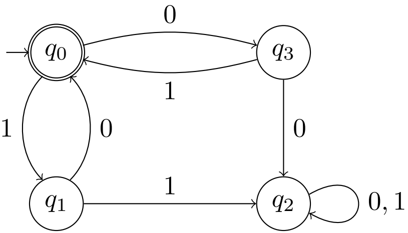
- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma = \{0, 1\}$
- δ

Example



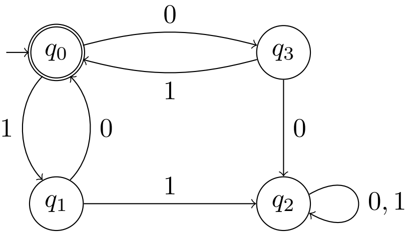
- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma = \{0, 1\}$
- δ
- $s =$

Example



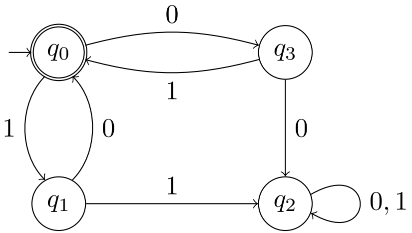
- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = q_0$

Example



- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = q_0$
- $A =$

Example



- $Q = \{q_0, q_1, q_1, q_3\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = q_0$
- $A = \{q_0\}$

Extending the transition function to strings

Given DFA $M = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is the state that M goes to from q on reading letter a

Useful to have notation to specify the unique state that M will reach from q on reading string w

Extending the transition function to strings

Given DFA $M = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is the state that M goes to from q on reading letter a

Useful to have notation to specify the unique state that M will reach from q on reading string w

Transition function $\delta^* : Q \times \Sigma^* \to Q$ defined inductively as follows:

- $\delta^*(q, w) = q$ if $w = \epsilon$
- $\delta^*(q, w) = \delta^*(\delta(q, a), x)$ if $w = ax$.

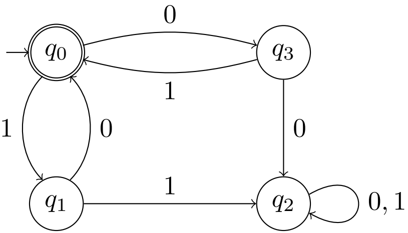
Formal definition of language accepted by M

Definition

The language $L(M)$ accepted by a DFA $M = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \in A\}.$$

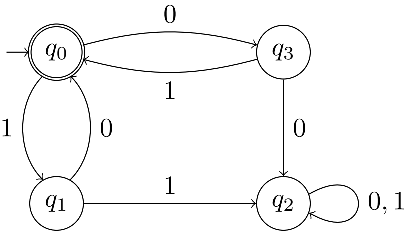
Example



What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$

Example continued



- What is $L(M)$ if start state is changed to q_1?
- What is $L(M)$ if final/accepting states are set to $\{q_2, q_3\}$ instead of $\{q_0\}$?

Advantages of formal specification

- Necessary for proofs
- Necessary to specify abstractly for class of languages

Exercise: Prove by induction that for any two strings u, v, and any state q,

$$\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$$

.

Part II

Constructing DFAs

DFAs: State = Memory

How do we design a DFA M for a given language L? That is $L(M) = L$.

- DFA is a like a program that has fixed amount of memory independent of input size.
- The memory of a DFA is encoded in its states
- The state/memory must capture enough information from the input seen so far that it is sufficient for the suffix that is yet to be seen (note that DFA cannot go back)

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is divisible by } 5\}$

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is divisible by } 5\}$
- $L = \{w \in \{0, 1\}^* \mid w \text{ ends with } 01\}$

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0, 1\}^* \mid |w| \text{ is divisible by } 5\}$
- $L = \{w \in \{0, 1\}^* \mid w \text{ ends with } 01\}$
- $L = \{w \in \{0, 1\}^* \mid w \text{ contains } 001 \text{ as substring}\}$

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0,1\}^* \mid |w| \text{ is divisible by } 5\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ ends with } 01\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ contains } 001 \text{ as substring}\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ contains } 001 \text{ or } 010 \text{ as substring}\}$

DFA Construction: Example

Assume $\Sigma = \{0, 1\}$

- $L = \emptyset$, $L = \Sigma^*$, $L = \{\epsilon\}$, $L = \{0\}$.
- $L = \{w \in \{0,1\}^* \mid |w| \text{ is divisible by } 5\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ ends with } 01\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ contains } 001 \text{ as substring}\}$
- $L = \{w \in \{0,1\}^* \mid w \text{ contains } 001 \text{ or } 010 \text{ as substring}\}$
- $L = \{w \mid w \text{ has a } 1 \ k \text{ positions from the end}\}$

DFA Construction: Example

$L = \{$Binary numbers congruent to $\mathbf{0 \mod 5}\}$
Example: $\mathbf{1101011 = 107 = 2 \mod 5}$, $\mathbf{1010 = 10 = 0 \mod 5}$

DFA Construction: Example

$L = \{\text{Binary numbers congruent to } \mathbf{0} \mod \mathbf{5}\}$

Example: $\mathbf{1101011 = 107 = 2} \mod \mathbf{5}$, $\mathbf{1010 = 10 = 0} \mod \mathbf{5}$

Key observation:

$w \mod \mathbf{5} = a$ implies

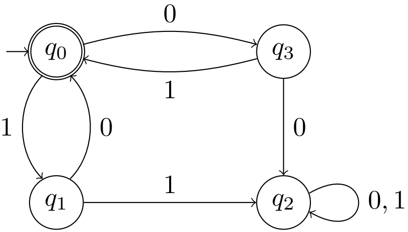
$w\mathbf{0} \mod \mathbf{5} = \mathbf{2}a \mod \mathbf{5}$ and $w\mathbf{1} \mod \mathbf{5} = (\mathbf{2}a + \mathbf{1}) \mod \mathbf{5}$

Part III

Complement

Complement

Question: If M is a DFA, is there a DFA M' such that $L(M') = \Sigma^* \setminus L(M)$? That is, are languages recognized by DFAs closed under complement?



Complement

Theorem

Languages accepted by DFAs are closed under complement.

Complement

Theorem

Languages accepted by DFAs are closed under complement.

Proof.

Let $M = (Q, \Sigma, \delta, s, A)$ such that $L = L(M)$.
Let $M' = (Q, \Sigma, \delta, s, Q \setminus A)$. Claim: $L(M') = \bar{L}$. Why?

Complement

Theorem

Languages accepted by DFAs are closed under complement.

Proof.

Let $M = (Q, \Sigma, \delta, s, A)$ such that $L = L(M)$.
Let $M' = (Q, \Sigma, \delta, s, Q \setminus A)$. Claim: $L(M') = \bar{L}$. Why?
$\delta^*_M = \delta^*_{M'}$. Thus, for every string w, $\delta^*_M(s, w) = \delta^*_{M'}(s, w)$.
$\delta^*_M(s, w) \in A \Rightarrow \delta^*_{M'}(s, w) \notin Q \setminus A$.
$\delta^*_M(s, w) \notin A \Rightarrow \delta^*_{M'}(s, w) \in Q \setminus A$. □

Part IV

Product Construction

Union and Intersection

Question: Are languages accepted by DFAs closed under union?
That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?
How about intersection $L(M_1) \cap L(M_2)$?

Union and Intersection

Question: Are languages accepted by DFAs closed under union? That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?
How about intersection $L(M_1) \cap L(M_2)$?

Idea from programming: on input string w

- Simulate M_1 on w
- Simulate M_2 on w
- If both accept than $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$.

Union and Intersection

Question: Are languages accepted by DFAs closed under union? That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?
How about intersection $L(M_1) \cap L(M_2)$?

Idea from programming: on input string w

- Simulate M_1 on w
- Simulate M_2 on w
- If both accept than $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$.
- Catch: We want a single DFA M that can only read w once.

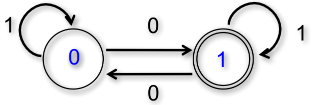
Union and Intersection

Question: Are languages accepted by DFAs closed under union?
That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?
How about intersection $L(M_1) \cap L(M_2)$?

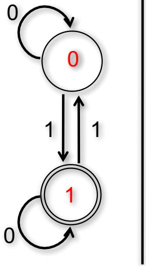
Idea from programming: on input string w

- Simulate M_1 on w
- Simulate M_2 on w
- If both accept than $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$.
- Catch: We want a single DFA M that can only read w once.
- Solution: Simulate M_1 and M_2 in parallel by keeping track of states of both machines

Example

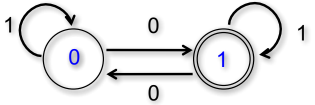


M_1 accepts #0 = odd

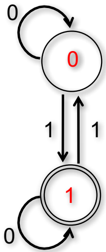


M_2 accepts #1 = odd

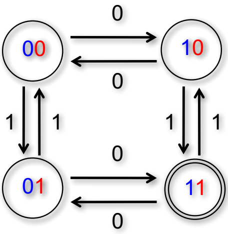
Example



M_1 accepts #0 = odd



M_2 accepts #1 = odd



Cross-product machine

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q =$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s =$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) =$$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

- $A =$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

- $A = A_1 \times A_2 = \{(q_1, q_2) \mid q_1 \in A_1, q_2 \in A_2\}$

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

- $A = A_1 \times A_2 = \{(q_1, q_2) \mid q_1 \in A_1, q_2 \in A_2\}$

Theorem

$L(M) = L(M_1) \cap L(M_2)$.

Correctness of construction

Lemma

For each string w, $\delta^*(s, w) = (\delta_1^*(s_1, w), \delta_2^*(s_2, w))$.

Correctness of construction

Lemma

For each string w*,* $\delta^*(s, w) = (\delta_1^*(s_1, w), \delta_2^*(s_2, w))$*.*

Exercise: Assuming lemma prove the theorem in previous slide.

Correctness of construction

Lemma

For each string w*,* $\delta^*(s, w) = (\delta_1^*(s_1, w), \delta_2^*(s_2, w))$*.*

Exercise: Assuming lemma prove the theorem in previous slide.

Proof of lemma by induction on $|w|$

Product construction for union

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

- $A =$

Product construction for union

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2)$

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

- $A = \{(q_1, q_2) \mid q_1 \in A_1$ or $q_2 \in A_2\}$

Theorem

$L(M) = L(M_1) \cup L(M_2)$.

Set Difference

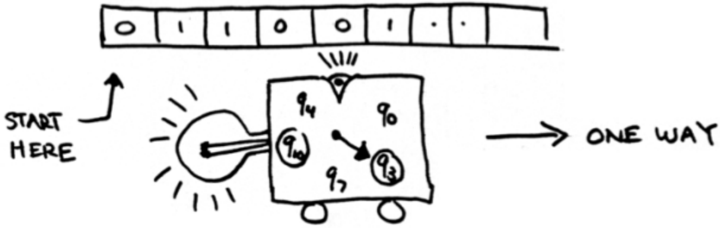
Theorem

M_1, M_2 DFAs. There is a DFA M such that
$L(M) = L(M_1) \setminus L(M_2)$.

Exercise: Prove the above using two methods.

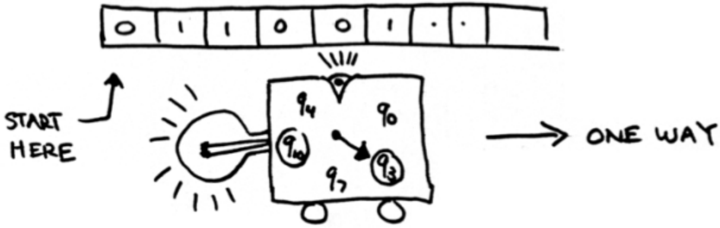
- Using a direct product construction
- Using closure under complement and intersection and union

Things to know: 2-way DFA



Question: Why are DFAs required to only move right?
Can we allow DFA to scan back and forth? Caveat: Tape is read-only so only memory is in machine's state.

Things to know: 2-way DFA



Question: Why are DFAs required to only move right?
Can we allow DFA to scan back and forth? Caveat: Tape is read-only so only memory is in machine's state.

- Can define a formal notion of a "2-way" DFA
- Can show that any language recognized by a 2-way DFA can be recognized by a regular (1-way) DFA
- Proof is tricky simulation via NFAs