Deterministic Finite Automata (DFAs)

Lecture 3

Tuesday, September 3, 2024

LATEXed: September 5, 2024 09:12

3.1
DFA Introduction

DFAs also called Finite State Machines (FSMs)

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

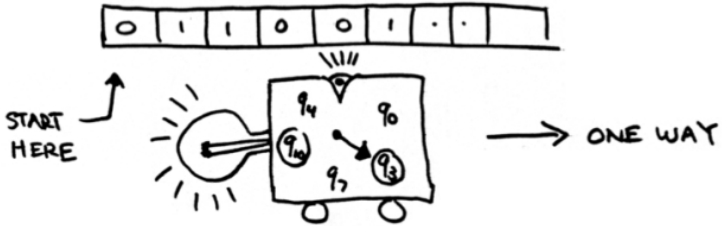
A simple program

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
```

```
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



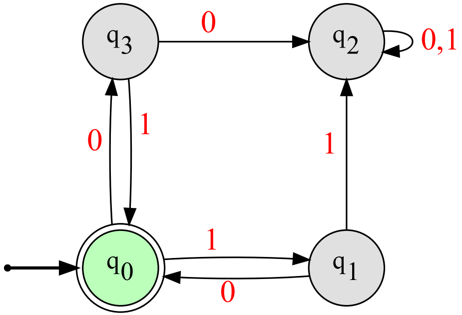
- Machine has input written on a <u>read-only</u> tape
- Start in specified start state
- Start at left, scan symbol, change state and move right
- Circled states are <u>accepting</u>
- Machine <u>accepts</u> input string if it is in an accepting state after scanning the last symbol.

Draw me a ~~sheep~~ DFA

DFA to check if a given input string has odd length

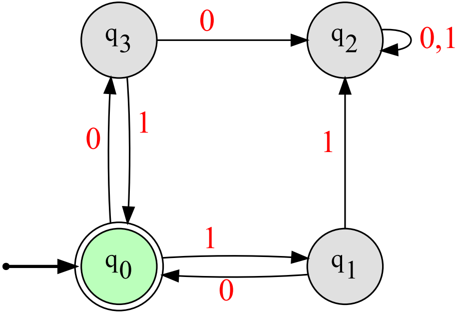
3.1.1
Graphical representation of DFA

Graphical Representation/State Machine



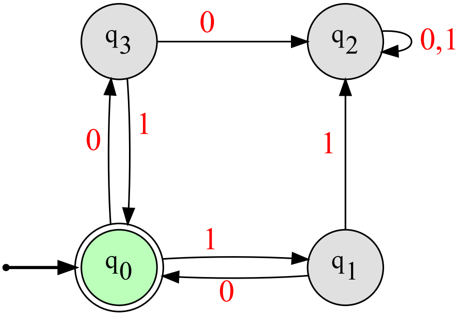
- Directed graph with nodes representing states and edge/arcs representing transitions labeled by symbols in Σ
- For each state (vertex) q and symbol $a \in \Sigma$ there is exactly one outgoing edge labeled by a
- Initial/start state has a pointer (or labeled as s, q_0 or "start")
- Some states with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

Graphical Representation



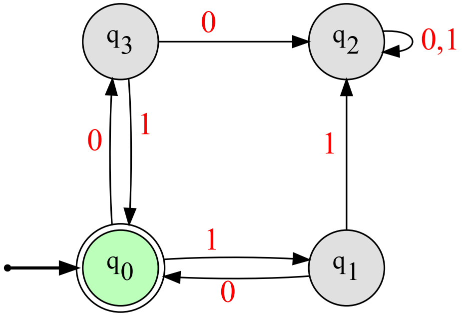
- Where does **001** lead?
- Where does **10010** lead?
- Which strings end up in accepting state?
- Can you prove it?
- Every string w has a unique walk that it follows from a given state q by reading one letter of w from left to right.

Graphical Representation



- Where does **001** lead?
- Where does **10010** lead?
- Which strings end up in accepting state?
- Can you prove it?
- 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 3.1.

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 3.2.

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 $\Sigma^* \setminus L$.

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

3.1.2
Formal definition of DFA

Formal Tuple Notation

Definition 3.3.

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

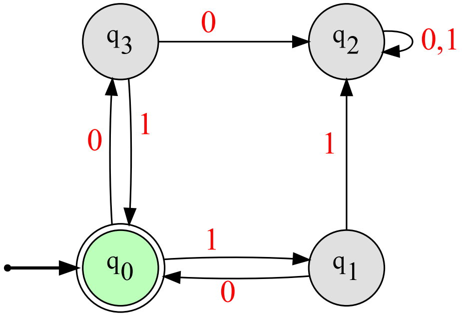
- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

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

DFA Notation

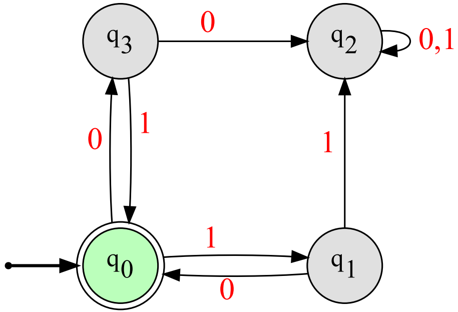
$$M = \Big(\overbrace{Q}^{\text{set of all states}}, \underbrace{\Sigma}_{\text{alphabet}}, \overbrace{\delta}^{\text{transition func}}, \underbrace{s}_{\text{start state}}, \overbrace{A}^{\text{set of all accept states}} \Big)$$

Example



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

Example: The transition function



δ :

state q	input c	result $\delta(q,c)$
$\in Q$	$\in \Sigma$	$\in Q$
q_0	0	q_3
q_0	1	q_1
q_1	0	q_0
q_1	1	q_2
q_2	0	q_2
q_2	1	q_2
q_3	0	q_2
q_3	1	q_0

3.1.3
Extending the transition function to strings

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 $\delta^* : Q \times \Sigma^* \rightarrow Q$ defined inductively as follows:

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

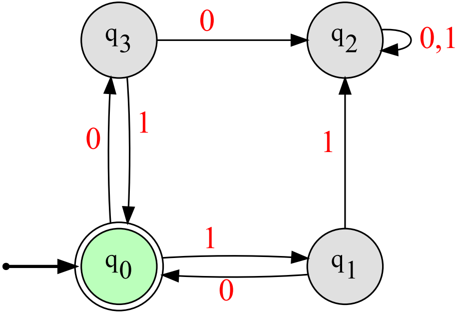
Formal definition of language accepted by $\mathbf{M}$

Definition 3.4.

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

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

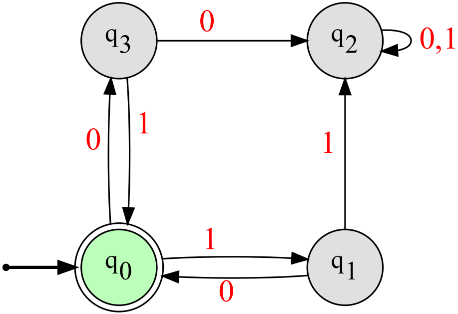
Example



What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$
- So what is $L(M)$??????

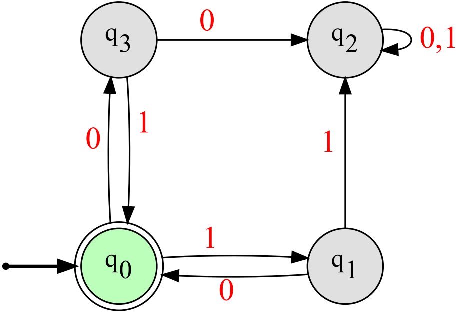
Example



What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$
- So what is $L(M)$??????

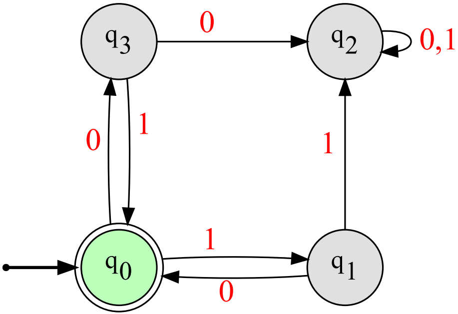
Example



What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$
- So what is $L(M)$??????

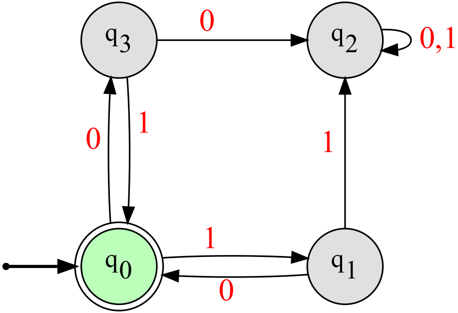
Example



What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$
- So what is $L(M)$??????

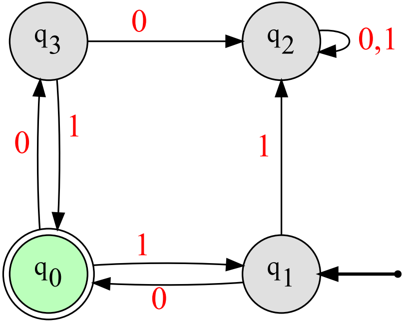
Example



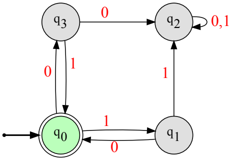
What is:

- $\delta^*(q_1, \epsilon)$
- $\delta^*(q_0, 1011)$
- $\delta^*(q_1, 010)$
- $\delta^*(q_4, 10)$
- So what is $L(M)$??????

Example continued

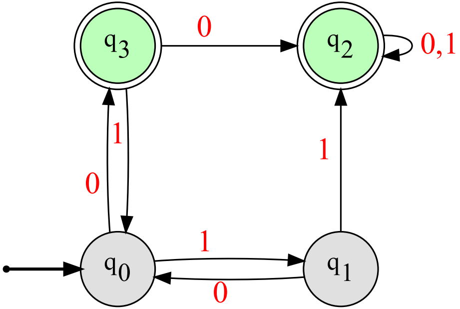


Old:

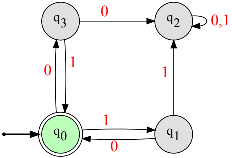


- What is $L(M)$ if start state is changed to q_1?

Example continued

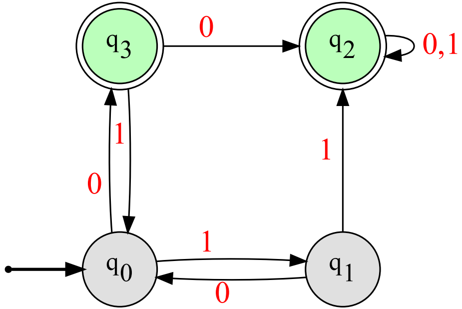


Old version:

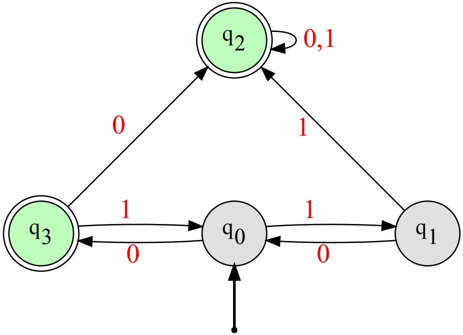


- What is $L(M)$ if final/accept states are set to $\{q_2, q_3\}$ instead of $\{q_0\}$?

Example continued

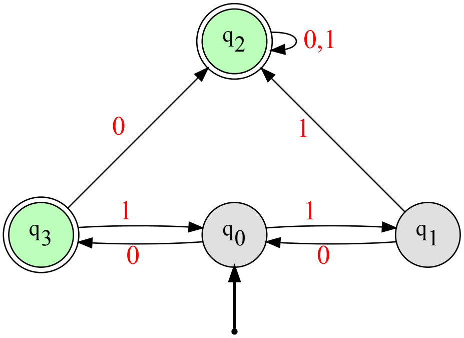


Redraw:



- What is $\boldsymbol{L(M)}$ if final/accept states are set to $\{q_2, q_3\}$ instead of $\{q_0\}$?

Example continued



- What is $\boldsymbol{L(M)}$ if final/accept 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, any state q,
$\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$.

3.2
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: Examples

Example I: Basic languages

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

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

DFA Construction: Examples

Example II: Length divisible by 5

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

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

DFA Construction: examples

Example III: Ends with 01

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

$L = \{w \in \{0, 1\}^* \mid w \text{ ends with } 01\}$

DFA Construction: examples

Example IV: Contains 001

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

$L = \{w \in \{\mathbf{0}, \mathbf{1}\}^* \mid w \text{ contains } 001 \text{ as substring}\}$

DFA Construction: examples

Example V: Contains 001 or 010

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

$L = \{w \in \{\mathbf{0}, \mathbf{1}\}^* \mid w \text{ contains 001 or 010 as substring}\}$

DFA construction examples

Example VI: Has a **1** exactly **k** positions from end

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

$L = \{w \mid w \text{ has a 1 } k \text{ positions from the end}\}$.

DFA Construction: Example

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

1. $\mathbf{1101011_2 = 107_{10} = 2 \mod 5}$,
2. $\mathbf{1010_2 = 10 = 0 \mod 5}$

Key observation:
$\mathrm{val}(w) \mod 5 = a$ implies

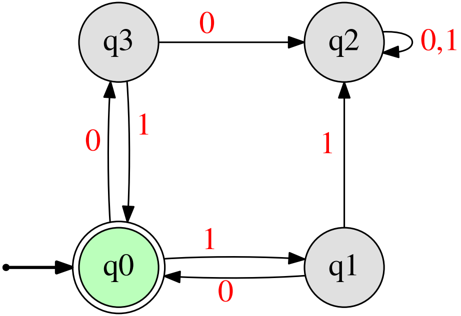
$$\mathrm{val}(w0) \mod 5 = (\mathrm{val}(w) * 2) \mod 5 = 2a \mod 5$$

$$\mathrm{val}(w1) \mod 5 = \big(\mathrm{val}(w) \cdot 2 + 1\big) \mod 5 = (2a + 1) \mod 5$$

3.3
Complement language

Complement

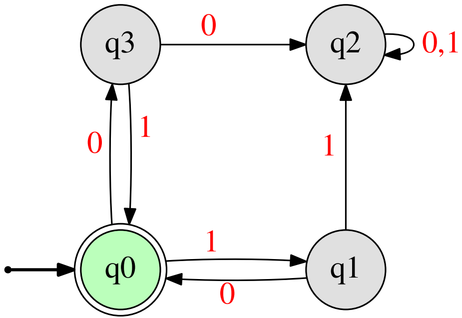
Question: If M is a DFA, is there a DFA M' such that $L(M') = \Sigma^* \setminus L(M)$? That is, are languages recognized by DFAs closed under complement?

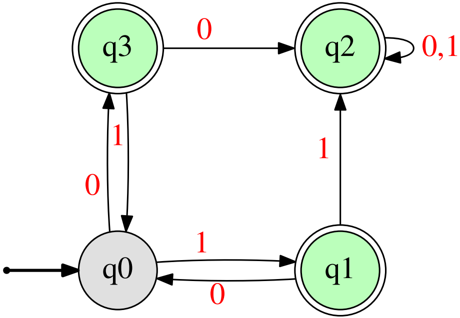


Complement

Example...

Just flip the state of the states!





Complement

Theorem 3.1.

Languages accepted by DFA*s 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, $\delta^*_M(s, w) = \delta^*_{M'}(s, w)$.
$\delta^*_M(s, w) \in A \Rightarrow \delta^*_{M'}(s, w) \notin Q \setminus A$. $\delta^*_M(s, w) \notin A \Rightarrow \delta^*_{M'}(s, w) \in Q \setminus A$. $\square$

3.4
Product Construction

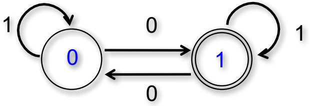
Union and Intersection

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 $L(M_1) \cap L(M_2)$?

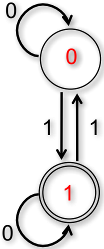
Idea from programming: on input string w

- 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

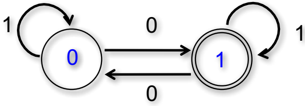


M_1 accepts #0 = odd

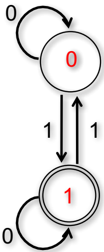


M_2 accepts #1 = odd

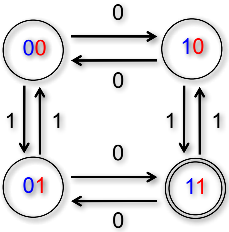
Example



M_1 accepts #0 = odd



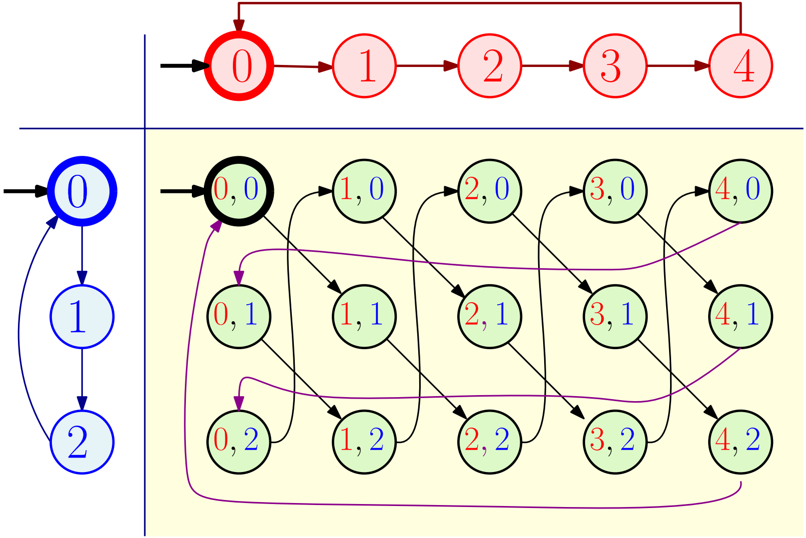
M_2 accepts #1 = odd



Cross-product machine

Example II

Accept all binary strings of length divisible by **3** and **5**



Assume all edges are labeled by **0**,**1**.

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$

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 3.1.

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

Correctness of construction

Lemma 3.2.

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

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

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

Theorem 3.3.

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

Set Difference

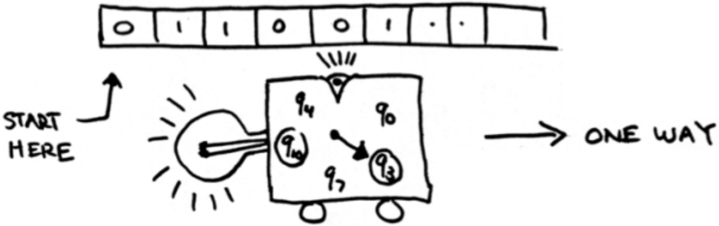
Theorem 3.4.

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.

- 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

3.5
Supplemental: DFA philosophy

A finite program can be simulated by a DFA...

1. Finite program = a program that uses a prespecified bounded amount of memory.
2. Given DFA and input, easy to decide if DFA accepts input.
3. A finite program is a DFA!
   # of states of memory of a finite program = finite.
   # states $\approx 2^{\text{\# of memory bits used by program}}$
4. Program using $1K$ memory = has...
5. Turing halting theorem: Not possible (in general) to decide if a program stops on an input.
6. DFA $\neq$ programs.

A finite program can be simulated by a DFA...

1. Finite program = a program that uses a prespecified bounded amount of memory.
2. Given DFA and input, easy to decide if DFA accepts input.
3. A finite program is a DFA!
   \# of states of memory of a finite program = finite.
   # states $\approx 2^{\text{\# of memory bits used by program}}$
4. Program using $\mathbf{1}K$ memory = has...
5. Turing halting theorem: Not possible (in general) to decide if a program stops on an input.
6. DFA $\neq$ programs.

A finite program can be simulated by a DFA...

1. Finite program = a program that uses a prespecified bounded amount of memory.
2. Given DFA and input, easy to decide if DFA accepts input.
3. A finite program is a DFA!
   # of states of memory of a finite program = finite.
   # states ≈ $2^{\#\text{ of memory bits used by program}}$
4. Program using $\mathbf{1}K$ memory = has...
5. Turing halting theorem: Not possible (in general) to decide if a program stops on an input.
6. DFA ≠ programs.

A finite program can be simulated by a DFA...

1. Finite program = a program that uses a prespecified bounded amount of memory.
2. Given DFA and input, easy to decide if DFA accepts input.
3. A finite program is a DFA!
   # of states of memory of a finite program = finite.
   # states ≈ $2^{\#\text{ of memory bits used by program}}$
4. Program using $\mathbf{1K}$ memory = has...
5. Turing halting theorem: Not possible (in general) to decide if a program stops on an input.
6. DFA ≠ programs.

A finite program can be simulated by a DFA...

1. Finite program = a program that uses a prespecified bounded amount of memory.
2. Given DFA and input, easy to decide if DFA accepts input.
3. A finite program is a DFA!
   $\#$ of states of memory of a finite program = finite.
   $\#$ states $\approx 2^{\#\text{ of memory bits used by program}}$
4. Program using **$1K$** memory = has...
5. Turing halting theorem: Not possible (in general) to decide if a program stops on an input.
6. DFA $\neq$ programs.

A finite program can be simulated by a DFA...

1. Finite program = a program that uses a prespecified bounded amount of memory.
2. Given DFA and input, easy to decide if DFA accepts input.
3. A finite program is a DFA!
   \# of states of memory of a finite program = finite.
   # states $\approx$ $\mathbf{2}^{\text{\# of memory bits used by program}}$
4. Program using $\mathbf{1}\boldsymbol{K}$ memory = has...
5. Turing halting theorem: Not possible (in general) to decide if a program stops on an input.
6. DFA $\neq$ programs.

But universe is finite...

1. Estimate # of atoms in the universe is $\mathbf{10^{82}}$.
2. Assuming each atom can store only finite number of bits.
3. So... number of states of the universe is finite!
4. So... All programs in this universe are DFAs.
5. Checkmate Mate!
6. What is all this nonsense?

But universe is finite...

1. Estimate # of atoms in the universe is $\mathbf{10^{82}}$.
2. Assuming each atom can store only finite number of bits.
3. So... number of states of the universe is finite!
4. So... All programs in this universe are DFAs.
5. Checkmate Mate!
6. What is all this nonsense?

But universe is finite...

1. Estimate # of atoms in the universe is $\mathbf{10^{82}}$.
2. Assuming each atom can store only finite number of bits.
3. So... number of states of the universe is finite!
4. So... All programs in this universe are DFAs.
5. Checkmate Mate!
6. What is all this nonsense?

But universe is finite...

1. Estimate # of atoms in the universe is $\mathbf{10^{82}}$.
2. Assuming each atom can store only finite number of bits.
3. So... number of states of the universe is finite!
4. So... All programs in this universe are DFAs.
5. Checkmate Mate!
6. What is all this nonsense?

But universe is finite...

1. Estimate # of atoms in the universe is $\mathbf{10^{82}}$.
2. Assuming each atom can store only finite number of bits.
3. So... number of states of the universe is finite!
4. So... All programs in this universe are DFAs.
5. Checkmate Mate!
6. What is all this nonsense?

But universe is finite...

1. Estimate # of atoms in the universe is $\mathbf{10^{82}}$.
2. Assuming each atom can store only finite number of bits.
3. So... number of states of the universe is finite!
4. So... All programs in this universe are DFAs.
5. Checkmate Mate!
6. What is all this nonsense?

So what is going on...

1. Theory models the world. (Oversimplifies it.)
2. Make it possible to think about it.
3. There are cases where theory does not model the world well.
4. Know when to apply the theory.
5. Reject statements that are correct but not useful.
6. Really Large finite numbers are