Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings with an odd number of 0’s

Formulate a **language** that describes the above problem.
CS/ECE-374: Lecture 3 - DFAs

Lecturer: Nickvash Kani  
Chat moderator: Samir Khan  
February 02, 2021  

University of Illinois at Urbana-Champaign
Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings with an odd number of 0’s

\[(01^*001^*)^*\] can we get \(010010\)

\[\exists = \{0, 1, 3\}\]

\[\exists = \{0, 1, 3\} \quad 01^* (001^*)^* 1^* \quad \equiv \quad 01^* (01^*01^*)^*\]
Deterministic-finite-automata (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 an input string w has odd number of 0’s

```plaintext
int n = 0
While input is not finished
    read next character c
    If (c \equiv '0')
        n ← n + 1
endWhile
If (n is odd) output YES
Else output NO
```
A simple program

Program to check if an input string $w$ has odd number of 0's

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

```
bit $x = 0$
While input is not finished
    read next character $c$
    If ($c \equiv '0'$)
        $x \leftarrow \text{flip}(x)$
    endWhile
If ($x = 1$) output YES
Else output NO
```
Another view

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

- Directed graph with nodes representing states and edge/arc pairs representing transitions labeled by symbols in \( \Sigma \)
- 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? $q_0$
Graphical Representation

- Where does 001 lead?
- Where does 10010 lead?
Graphical Representation

- Where does 001 lead?
- Where does 10010 lead?
- Which strings end up in accepting state?

All strings with odd number of 0's
Graphical Representation

- Where does 001 lead?
- Where does 10010 lead?
- 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.
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**
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: $L(M) = \{ w \mid M \text{ accepts } w \}$. 
Formal definition of DFA
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**, 
- $\Sigma$ 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

\[ M = (Q, \Sigma, \delta, S, A) \]

- \( Q \): all states
- \( \Sigma \): input alphabet
- \( \delta \): transition functions
- \( S \): start state
- \( A \): accepting states
Example

\[
\begin{align*}
Q &= \{ q_0, q_1 \} \\
\Sigma &= \{ 0, 1 \} \\
\delta &= \begin{pmatrix}
q_0 & q_1 & q_0 \\
q_1 & q_0 & q_1 \\
\end{pmatrix}
\end{align*}
\]

\[
\begin{array}{c|cc}
\text{state} & \text{char} & \text{new state} \\
\hline
q_0 & 0 & q_1 \\
q_1 & 0 & q_0 \\
q_1 & 1 & q_1 \\
\end{array}
\]

\[
s = q_0
\]

\[
A = \{ q_1 \}
\]
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 $w$:

$$\delta(q, w)$$
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 $\delta^* : Q \times \Sigma^* \rightarrow Q$ defined inductively as follows:

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

$w = y \ast$

$\delta^*(\delta^*(q, y), x) \ast \ast \ast \ast$
Formal definition of language accepted by $M$

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

$$\{ w \in \Sigma^* | \delta^*(s, w) \in A \}.$$
What is:

- $\delta^*(q_1, \epsilon) = q_1$
Example

What is:

• $\delta^*(q_1, \epsilon) = \_

• $\delta^*(q_0, 1011) = q_1$
What is:

- $\delta^*(q_1, \varepsilon) =$
- $\delta^*(q_0, 1011) =$
- $\delta^*(q_1, 010) = q_2$. 
Constructing DFAs: Examples
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 I: Basic languages

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

- $L = \emptyset = \varepsilon^*$

- $L = \Sigma^*$

- $L = \{\varepsilon\}$

- $L = \{0\}$
DFA Construction: Example II: Length divisible by 5

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

$L = \{ w \in \{0, 1\}^* \mid |w| \text{ is divisible by 5} \}$
DFA Construction: Example III: Ends with 01

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

$L = \{w \in \{0, 1\}^* \mid w \text{ ends with } 01\}$
Constructing regular expressions
DFAs to regular expressions

Personal Lemma:
Mastering a concept means being able to do a problem in both direction.

Time to reverse problem direction and find regular expressions using DFAs.

Multiple methods but the ones I’m focusing on:

- State removal method
- Algebraic method
State Removal method

If \( q_1 = \delta(q_0, x) \) and \( q_2 = \delta(q_1, y) \)

then \( q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y) = \delta(q_0, xy) \)
State Removal method - Example
State Removal method - Example

![State Transition Diagram](image-url)
State Removal method - Example

- Start state: $q_0$
- Transitions:
  - $01 \rightarrow q_0$
  - $100 \rightarrow q_2$
  - $001 \rightarrow q_2$
  - $011 \rightarrow q_2$

- Expressions:
  - $(1+00)(10)^*(0+11)$
State Removal method - Example

\[
01 + (1 + 00)(10)^*(0 + 11)
\]
State Removal method - Example

\[ 01 + (1 + 00)(10)^*(0 + 11) \]

\[ \text{(Start: } q_0 \text{)} \]

\[ (01 + (1 + 00)(10)^*(0 + 11))^* \]
Algebraic method

Transition functions are themselves algebraic expressions!

Demarcate states as variables.

Can rewrite $q_1 = \delta(q_0, x)$ as $q_1 = q_0x$

Solve for accepting state.

$q_0 = q_0x$
Algebraic method - Example

Diagram:

- \( q_0 \) is the start state.
- Transitions:
  - From \( q_0 \) to \( q_0 \) on 0.
  - From \( q_0 \) to \( q_1 \) on 1.
  - From \( q_1 \) to \( q_0 \) on 0.
  - From \( q_1 \) to \( q_3 \) on 0.
  - From \( q_2 \) to \( q_3 \) on 1.
  - From \( q_2 \) to \( q_2 \) on 0.
  - From \( q_1 \) to \( q_1 \) on 1.
  - From \( q_3 \) to \( q_3 \) on 0,1.
Algebraic method - Example

- $q_0 = \epsilon + q_11 + q_20$
- $q_1 = q_00$
- $q_2 = q_01$
- $q_3 = q_10 + q_21 + q_3(0 + 1)$
Algebraic method - Example

\begin{itemize}
  \item \( q_0 = \epsilon + q_1 \cdot 1 + q_2 \cdot 0 \)
  \item \( q_1 = q_0 \cdot 0 \)
  \item \( q_2 = q_0 \cdot 1 \)
  \item \( q_3 = q_1 \cdot 0 + q_2 \cdot 1 + q_3 \cdot (0 + 1) \)
\end{itemize}

Now we simple solve the system of equations for \( q_0 \):

\begin{itemize}
  \item \( q_0 = \epsilon + q_1 \cdot 1 + q_2 \cdot 0 \)
  \item \( q_0 = \epsilon + q_0 \cdot 01 + q_0 \cdot 10 \)
  \item \( q_0 = \epsilon + q_0 \cdot (01 + 10) \)
\end{itemize}

**Theorem (Arden’s Theorem)**
\[ R = Q + RP = QP^* \]
Algebraic method - Example

- \( q_0 = \epsilon + q_1 1 + q_2 0 \)
- \( q_1 = q_0 0 \)
- \( q_2 = q_0 1 \)
- \( q_3 = q_1 0 + q_2 1 + q_3 (0 + 1) \)

Now we simple solve the system of equations for \( q_0 \):

- \( q_0 = \epsilon + q_1 1 + q_2 0 \)
- \( q_0 = \epsilon + q_0 0 1 + q_0 1 0 \)
- \( q_0 = \epsilon + q_0 (0 1 + 1 0) \)
- \( q_0 = (0 1 + 1 0)^* = (0 1 + 1 0)^* \)
Complement language
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

Just flip the state of the states!

For odd number of 0's:
- Start at $q_0$.
- Transition on 1 to $q_1$.
- Transition on 0 to $q_0$.

For even number of 0's:
- Start at $q_0$.
- Transition on 1 to $q_1$.
- Transition on 0 to $q_0$. 

28
Complement

Theorem
Languages accepted by DFAs are closed under complement.

If $L$ is regular then $\bar{L}$ is regular.
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: $L(M') = \overline{L}$. Why?

$\delta^*_M = \delta^*_{M'}$. Thus, for every string $w$, $\delta^*_M(s, w) = \delta^*_{M'}(s, w)$.

$\delta^*_M(s, w) \in A \Rightarrow \delta^*_{M'}(s, w) \notin Q \setminus A$.

$\delta^*_M(s, w) \notin A \Rightarrow \delta^*_{M'}(s, w) \in Q \setminus A$. 

$\square$
Product Construction
Union and Intersection

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

$$L = \{ w \mid w \text{ has odd # of 0's} \}$$

$M_1$ odd # of 0's

$M_2$ odd # of 1's
Union and Intersection

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 then $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$. 
Union and Intersection

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

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 then $w \in L(M_1) \cap L(M_2)$. 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 $\#0 = \text{odd}$

$M_2$ accepts $\#1 = \text{odd}$

Cross-product machine
Product construction for intersection

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

**Theorem**

\[ L(M) = L(M_1) \cap L(M_2). \]

Create \[ M = (Q, \Sigma, \delta, s, A) \] where
Product construction for intersection

\[ M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1) \text{ and } M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2) \]

**Theorem**
\[ L(M) = L(M_1) \cap L(M_2). \]

Create \( M = (Q, \Sigma, \delta, s, A) \) where
\[
\begin{align*}
\cdot & \quad Q = Q_1 \times Q_2 = \{ (q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2 \} \\
\cdot & \quad s = (s_1, s_2) \\
\cdot & \quad \delta : Q \times \Sigma \rightarrow Q \text{ where } \\
& \quad \delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)) \\
\cdot & \quad A = A_1 \times A_2 = \{ (q_1, q_2) \mid q_1 \in A_1, q_2 \in A_2 \} 
\end{align*}
\]
Intersection vs Union

\[ M_1: \]

\[ M_2: \]

\[ M_1 \cap M_2 \]

\[ M_1 \cup M_2 \]
Product construction for union

\[ M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1) \text{ and } M_2 = (Q_1, \Sigma, \delta_2, s_2, A_2) \]

**Theorem**

\[ L(M) = L(M_1) \cup L(M_2). \]

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

- \( Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\} \)
- \( s = (s_1, s_2) \)
- \( \delta : Q \times \Sigma \rightarrow Q \) where \( \delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)) \)
- \( A = \{(q_1, q_2) \mid q_1 \in A_1 \text{ or } q_2 \in A_2\} \)