Pre-lecture brain teaser

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

- Constraint: odd # of zeros
  $$0(00)^*$$

- Generalizability: $1^*01^*(0101^*)^*$
ECE-374 B: Lecture 3 - DFAs

Instructor: Abhishek Kumar Umrawal
January 23, 2024

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

Given $\Sigma = \{0, 1\}$, 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.
Program to check if an input string \(w\) has odd number of 0's

\[
\text{int } n = 0 \\
\text{While input is not finished} \\
\text{read next character } c \\
\text{If (c }\equiv\text{0')} \\
\quad n \leftarrow n + 1 \\
\text{endWhile} \\
\text{If (n is odd) output YES} \\
\text{Else output NO}
\]
A simple program

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

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

```java
bit x = 0
While input is not finished
   read next character c
   If (c == '0')
      $x \leftarrow \text{flip}(x)$
endWhile
If (x = 1) output YES
Else output NO
```
• Machine has input written on a **read-only** tape
• Start in specified **start state**: even
• 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.
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
Graphical representation of DFA
• Directed graph with nodes representing states and edge/arcs 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? \( q_0 \)

- 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.

- Which strings end up in accepting state? All strings with odd number of 0's
• Where does 001 lead? $q_0$
• Where does 10010 lead? $q_1$
• Where does 001 lead? $q_0$
• Where does 10010 lead? $q_1$
• Which strings end up in accepting state? All strings with odd number of 0’s
Graphical Representation

- Where does 001 lead? $q_0$
- Where does 10010 lead? $q_1$
- Which strings end up in accepting state? All strings with odd number of 0’s
- 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.

Computation of a DFA
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: $L(M) = \{w \mid M$ 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

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

- \( Q \): set of all states
- \( \Sigma \): alphabet
- \( \delta \): transition function
- \( S \): start state
- \( A \): set of all accept states
Example

- $Q = \{ q_0, q_1 \}$
- $\Sigma = \{ 0, 1 \}$
- $\delta = \begin{array}{c|cc} & 0 & 1 \\ \hline q_0 & q_1 & q_0 \\ q_1 & q_0 & q_1 \end{array}$
- $s = q_0$
- $A = \{ q_1 \}$

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

$S(q_0, 01) = q_1$
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 $w$

$\delta : Q \times \Sigma \rightarrow Q$

Modify $\delta$ to:

$\delta^* : Q \times \Sigma^* \rightarrow Q$
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:

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

\[
\begin{align*}
101 &= ax \\
1 = a & b = 0 \\
01 &= x & y = 1 \\
= by
\end{align*}
\]
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:

- $\delta^*(q_1, \epsilon) = q_1$
What is:

- \( \delta^*(q_1, \epsilon) = q_1 \)
- \( \delta^*(q_0, 1011) = q_1 \)
What is:

- $\delta^*(q_1, \varepsilon) = q_1$
- $\delta^*(q_0, 1011) = q_1$
- $\delta^*(q_1, 010) = q_1$
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 number of states regardless of its input size.
- The state 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)
Assume $\Sigma = \{0, 1\}$.

1. $L = \emptyset$

![Diagram showing a DFA with states $q_0$ and transitions labeled with 0 and 1.](image)
DFA Construction: Example I: Basic languages

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

1. $L = \emptyset$

2. $L = \Sigma^*$

3. $L = \{x \in \Sigma^* | x = x_1 \cdots x_n \text{ and } x_1 \neq \varepsilon \}$

4. $L = \{x \in \Sigma^* | x = x_1 \cdots x_n \}$
DFA Construction: Example I: Basic languages

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

1. $L = \emptyset$

2. $L = \Sigma^*$

3. $L = \{\epsilon\}$
Assume $\Sigma = \{0, 1\}$.

1. $L = \emptyset$

2. $L = \Sigma^*$

3. $L = \{\epsilon\}$

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

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

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

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

\[
\begin{align*}
M & \quad L(M) & \quad M' \\
\text{Def:} & \quad L(M') = \Sigma^* \setminus L(M)
\end{align*}
\]
Complement

Just flip the state of the states!

odd # of zeros

not odd # of zeros
Complement

Theorem

Languages accepted by DFAs are closed under complement.

Let $M = (Q, \Sigma, \delta, s, A)$ such that $L = L(M)$.

Let $M_0 = (Q, \Sigma, \delta, s, Q \cap A)$. Claim: $L(M_0) = \overline{L}$. Why?

$\ast M = \ast M_0$. Thus, for every string $w$, $\ast M(s, w) = \ast M_0(s, w)$.

$\ast M(s, w) \in A \implies \ast M_0(s, w) \notin Q \cap A$.

$\ast M(s, w) \notin A \implies \ast M_0(s, w) \in Q \cap A$. 

$L(M_0) = \Sigma^* \setminus L(M)$.

$\exists$ a machine (DFA) $M'$ such that $L(M') = \overline{L(M)}$. 

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

Does there exist a DFA $M$ such that $L(M) = L(M_1) \cup L(M_2)$?
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 $w \in L(M_1) \cup L(M_2)$. 

\[ \text{transition diagram} \]
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 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 $\#0 = \text{odd}$

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

$M_1$ accepts $#0 = \text{odd}$

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

Cross-product machine
Product construction for intersection

\[ M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1) \text{ and } M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2) \]

Theorem

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

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

- \( Q = Q_1 \times Q_2 \)
- \( \Sigma = \Sigma \)
- \( s = (s_1, s_2) \)
- \( A = \delta (q_1, q_2) : \ q_1 \in A_1 \text{ and } q_2 \in A_2 \)

Proof by construction

\[ q_r = (q_{r_1}, q_{r_2}) \]

\[ \delta : Q \times \Sigma \rightarrow Q \]

\[ \delta(q_r, a) = (\delta_1(q_{r_1}, a), \delta_2(q_{r_2}, a)) \]
Product construction for intersection

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

**Theorem**

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

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

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

\[ Q = \{ q_i : i = 1, \ldots, k \} \]
Intersection vs Union

$M_1$:

$M_2$:

$M_1 \cap M_2$

$M_1 \cup M_2$
Product construction for union

\[ M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1) \text{ and } M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2) \]

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

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

- \( Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\} \)
- \( s = (s_1, s_2) \)
- \( \delta : Q \times \Sigma \to Q \) where
  \[ \delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)) \]
- \( A = \{(q_1, q_2) \mid q_1 \in A_1 \text{ or } q_2 \in A_2\} \)
Constructing regular expressions
Personal (Prof. Kani) 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 \( q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y) = \delta(q_0, xy) \)

\[ q_0 \xrightarrow{x} q_1 \xrightarrow{y} q_2 \]

\[ q_0 \xrightarrow{xy} q_2 \]
State Removal method - Example

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \]

\[ q_0 \xrightarrow{0} q_1 \]

\[ q_1 \xrightarrow{1} q_2 \]

\[ q_2 \xrightarrow{1} q_1 \]

\[ q_2 \xrightarrow{0} q_0 \]

Input sequence: 0101
State Removal method - Example
State Removal method - Example

```
01
start

q0

1+00

10

0+11

0+11

31

q2
```
State Removal method - Example

State $q_0$ is the start state.

The transitions are as follows:
- From $q_0$ on input $01$, go to $q_0$.
- From $q_0$ on input $1+00$, go to $q_2$.
- From $q_2$ on input $10$, go to $q_2$.
- From $q_2$ on input $0+11$, go to $q_0$.

The regular expression is:

$$(01)^* (1+00)(10)^* (0+11)^*$$

The string $0101$ is accepted by the automaton.
State Removal method - Example

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

\[\text{start} \rightarrow q_0\]

\[(01 + (1 + 00)(10)^*(0 + 11))^*\]
DFAs and regular expressions

The thing to know right now is that DFAs and regular expressions represent the same set of languages!
The End - Reminders

• HW 1 has been assigned. Will be due next week.
• Lab tomorrow will go over DFAs
Extra Slides
Algebraic method

Transition functions are themselves algebraic expressions!

Demarcate states as variables.

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

Solve for accepting state.
Algebraic method - Example

\[
\begin{align*}
q_0 &= q_0 + q_1 + q_2 \\
q_1 &= q_0 q_0 q_3 \\
q_2 &= q_0 q_1 + q_2 q_3 \\
q_3 &= q_0 q_1 + q_2 q_3
\end{align*}
\]
Algebraic method - Example

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

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

Now we simply solve the system of equations for $q_0$:

- $q_0 = \epsilon + q_11 + q_20$
- $q_0 = \epsilon + q_001 + q_010$
- $q_0 = \epsilon + q_0(01 + 10)$

Theorem (Arden’s Theorem)
$R = Q + RP = QP^*$
• $q_0 = \epsilon + q_1 1 + q_2 0$
• $q_1 = q_0 0$
• $q_2 = q_0 1$
• $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Now we simply solve the system of equations for $q_0$:

• $q_0 = \epsilon + q_1 1 + q_2 0$
• $q_0 = \epsilon + q_0 01 + q_0 10$
• $q_0 = \epsilon + q_0 (01 + 10)$
• $q_0 = \epsilon (01 + 10)^* = (01 + 10)^*$