Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings that do not contain the subsequence 111000.
Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings that do not contain the subsequence 111000
What if we draw the above figure as:

What does this mean?
What if we draw the above figure as:
Simplifying DFAs

What if we draw the above figure as:

What does this mean?
Non-deterministic finite automata (NFA) Introduction
When you come to a fork in the road, take it.
When you come to a fork in the road, take it.

Today we’ll talk about automata whose logic **is not** deterministic.
When you come to a fork in the road, take it.

Today we’ll talk about automata whose logic is not deterministic.

But first.... what the heck is non-determinism?
Non-determinism in computing

Non-determinism is a special property of algorithms.

An algorithm that is capable of taking multiple states concurrently. Whenever it reaches a choice, it takes both paths.

If there is a path for the string to be accepted by the machine, then the string is part of the language.
**Informal definition:** An NFA $N$ *accepts a string* $w$ iff some accepting state is reached by $N$ from the start state on input $w$. 

---
Informal definition: An NFA $N$ accepts a string $w$ iff some accepting state is reached by $N$ from the start state on input $w$.

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

- Is 010110 accepted?
NFA acceptance: Example

Is 010110 accepted?
• Is $010110$ accepted?
• Is 010110 accepted?
• Is 010 accepted?
NFA acceptance: Example

- Is **010110** accepted?
- Is **010** accepted?
- Is **101** accepted?
NFA acceptance: Example

- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted?

Comment:
Unlike DFAs, it is easier in NFAs to show that a string is accepted than to show that a string is not accepted.
NFA acceptance: Example

- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted?
- What is the language accepted by $N$?
• Is 010110 accepted?
• Is 010 accepted?
• Is 101 accepted?
• Is 10011 accepted?
• What is the language accepted by $N$?
NFA acceptance: Example

- Is 010110 accepted?
- Is 010 accepted?
- Is 101 accepted?
- Is 10011 accepted?
- What is the language accepted by $N$?

Comment: Unlike DFAs, it is easier in NFAs to show that a string is accepted than to show that a string is not accepted.
Formal definition of NFA
Definition
A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where
Definition
A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- $Q$ is a finite set whose elements are called states,
Definition
A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- $Q$ is a finite set whose elements are called states,
- $\Sigma$ is a finite set called the input alphabet,
Definition
A non-deterministic finite automata (NFA) \( N = (Q, \Sigma, \delta, s, A) \) is a five tuple where

- \( Q \) is a finite set whose elements are called states,
- \( \Sigma \) is a finite set called the input alphabet,
- \( \delta : Q \times \Sigma \cup \{\varepsilon\} \to \mathcal{P}(Q) \) is the transition function (here \( \mathcal{P}(Q) \) is the power set of \( Q \)),
**Definition**

A **non-deterministic finite automata (NFA)** $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- $Q$ is a finite set whose elements are called **states**,
- $\Sigma$ is a finite set called the **input alphabet**,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the **transition function** (here $\mathcal{P}(Q)$ is the power set of $Q$),

\[ \mathcal{P}(Q) \]
Reminder: Power set

$Q$: a set. Power set of $Q$ is: $\mathcal{P}(Q) = 2^Q = \{X \mid X \subseteq Q\}$ is set of all subsets of $Q$.

**Example**

$Q = \{1, 2, 3, 4\}$

$\mathcal{P}(Q) = \{\{1, 2, 3, 4\}, \{2, 3, 4\}, \{1, 3, 4\}, \{1, 2, 4\}, \{1, 2, 3\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1\}, \{2\}, \{3\}, \{4\}, \{\}\}$
Definition
A non-deterministic finite automata (NFA) \( N = (Q, \Sigma, \delta, s, A) \) is a five tuple where

- \( Q \) is a finite set whose elements are called states,
- \( \Sigma \) is a finite set called the input alphabet,
- \( \delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q) \) is the transition function (here \( \mathcal{P}(Q) \) is the power set of \( Q \)).
Formal Tuple Notation

Definition
A non-deterministic finite automata (NFA) \( N = (Q, \Sigma, \delta, s, A) \) is a five tuple where

- \( Q \) is a finite set whose elements are called states,
- \( \Sigma \) is a finite set called the input alphabet,
- \( \delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q) \) is the transition function (here \( \mathcal{P}(Q) \) is the power set of \( Q \)),
- \( s \in Q \) is the start state,
Formal Tuple Notation

Definition
A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- $Q$ is a finite set whose elements are called states,
- $\Sigma$ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of $Q$),
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

$\delta(q, a)$ for $a \in \Sigma \cup \{\varepsilon\}$ is a subset of $Q$ — a set of states.
Example

- $Q =$
- $\Sigma =$
- $\delta =$

- $s =$
- $A =$
Extending the transition function to strings
Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$
Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$
- $\delta(q, a)$: set of states that $N$ can go to from $q$ on reading $a \in \Sigma \cup \{\varepsilon\}$. 
Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$
- $\delta(q, a)$: set of states that $N$ can go to from $q$ on reading $a \in \Sigma \cup \{\varepsilon\}$.
- Want transition function $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$
Extending the transition function to strings

- NFA $N = (Q, \Sigma, \delta, s, A)$
- $\delta(q, a)$: set of states that $N$ can go to from $q$ on reading $a \in \Sigma \cup \{\varepsilon\}$.
- Want transition function $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$
- $\delta^*(q, w)$: set of states reachable on input $w$ starting in state $q$. 


Definition
For NFA \( N = (Q, \Sigma, \delta, s, A) \) and \( q \in Q \) the \( \varepsilon \text{reach}(q) \) is the set of all states that \( q \) can reach using only \( \varepsilon \)-transitions.
Definition
For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\varepsilon$-reach($q$) is the set of all states that $q$ can reach using only $\varepsilon$-transitions.

Definition
For $X \subseteq Q$: $\varepsilon$-reach($X$) = $\bigcup_{x \in X} \varepsilon$-reach($x$).
Extending the transition function to strings

$\epsilon$reach($q$): set of all states that $q$ can reach using only $\epsilon$-transitions.

**Definition**
Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon$reach($q$)
Extending the transition function to strings

$\epsilon\text{reach}(q)$: set of all states that $q$ can reach using only $\epsilon$-transitions.

**Definition**
Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$:

$$\delta^*(q, a) = \epsilon\text{reach} \left( \bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a) \right)$$
Extending the transition function to strings

$\varepsilon\text{reach}(q)$: set of all states that $q$ can reach using only $\varepsilon$-transitions.

**Definition**

Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \varepsilon$, $\delta^*(q, w) = \varepsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$:
  \[
  \delta^*(q, a) = \varepsilon\text{reach}\left( \bigcup_{p \in \varepsilon\text{reach}(q)} \delta(p, a) \right)
  \]
- if $w = ax$:
  \[
  \delta^*(q, w) = \varepsilon\text{reach}\left( \bigcup_{p \in \varepsilon\text{reach}(q)} \left( \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right) \right)
  \]
Transition for strings: \( w = ax \)

\[
\delta^*(q, w) = \epsilonreach \left( \bigcup_{p \in \epsilonreach(q)} \left( \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right) \right)
\]

- \( R = \epsilonreach(q) \Rightarrow \)

\[
\delta^*(q, w) = \epsilonreach \left( \bigcup_{p \in R} \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right)
\]

- \( N = \bigcup_{p \in R} \delta^*(p, a) \): All the states reachable from \( q \) with the letter \( a \).

- \( \delta^*(q, w) = \epsilonreach \left( \bigcup_{r \in N} \delta^*(r, x) \right) \)
Definition
A string $w$ is accepted by NFA $N$ if $\delta^*_N(s, w) \cap A \neq \emptyset$.

Definition
The language $L(N)$ accepted by a NFA $N = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$
Formal definition of language accepted by $N$

**Definition**
A string $w$ is accepted by NFA $N$ if $\delta^*_N(s, w) \cap A \neq \emptyset$.

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

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

**Important:** Formal definition of the language of NFA above uses $\delta^*$ and not $\delta$. As such, one does not need to include $\varepsilon$-transitions closure when specifying $\delta$, since $\delta^*$ takes care of that.
What is:

\[ \delta^* (s, \epsilon) = \]

Example
What is:

- \( \delta^*(s, \epsilon) = \)
- \( \delta^*(s, 0) = \)
What is:

\[ \delta^*(s, \epsilon) = \]
\[ \delta^*(s, 0) = \]
\[ \delta^*(b, 0) = \]
What is:

- $\delta^*(s, \epsilon) =$
- $\delta^*(s, 0) =$
- $\delta^*(b, 0) =$
- $\delta^*(b, 00) =$
Why non-determinism?

- Non-determinism adds power to the model; richer programming language and hence (much) easier to “design” programs
- Fundamental in theory to prove many theorems
- Very important in practice directly and indirectly
- Many deep connections to various fields in Computer Science and Mathematics

Many interpretations of non-determinism. Hard to understand at the outset. Get used to it and then you will appreciate it slowly.
Constructing NFAs
DFAs and NFAs

- Every DFA is a NFA so NFAs are at least as powerful as DFAs.
- NFAs prove ability to “guess and verify” which simplifies design and reduces number of states
- Easy proofs of some closure properties
Example

Strings that represent decimal numbers.
Examples: 154, 345.75332, 534677567.1
Strings that represent valid C comments.

Examples:
\* Comment 1 *\ 
\\Comment 2
Example

$L_3 = \{\text{bitstrings that have a 1 three positions from the end}\}$
A simple transformation

**Theorem**
For every NFA $N$ there is another NFA $N'$ such that $L(N) = L(N')$ and such that $N'$ has the following two properties:

- $N'$ has single final state $f$ that has no outgoing transitions
- The start state $s$ of $N$ is different from $f$
Theorem
For every NFA $N$ there is another NFA $N'$ such that $L(N) = L(N')$ and such that $N'$ has the following two properties:

- $N'$ has single final state $f$ that has no outgoing transitions
- The start state $s$ of $N$ is different from $f$

Why couldn’t we say this for DFA’s?
A simple transformation

**Hint:** Consider the $L = 01 + 10$. 
Closure Properties of NFAs
Closure properties of NFAs

Are the class of languages accepted by NFAs closed under the following operations?

- union
- intersection
- concatenation
- Kleene star
- complement
Theorem
For any two NFAs $N_1$ and $N_2$ there is a NFA $N$ such that $L(N) = L(N_1) \cup L(N_2)$. 
**Theorem**
For any two NFAs $N_1$ and $N_2$ there is a NFA $N$ such that
$L(N) = L(N_1) \cup L(N_2)$. 

\[ q_1 \quad N_1 \quad f_1 \]
\[ q_2 \quad N_2 \quad f_2 \]
Theorem
For any two NFAs $N_1$ and $N_2$ there is a NFA $N$ such that $L(N) = L(N_1) \cdot L(N_2)$. 
Theorem
For any two NFAs $N_1$ and $N_2$ there is a NFA $N$ such that $L(N) = L(N_1) \cdot L(N_2)$.
Theorem

For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^*$. 
Theorem
For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^*$. 

Closure under Kleene star
Theorem
For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^*$. 

Does not work! Why?
Theorem
For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^*$. 
NFAs capture Regular Languages
Example

\((\varepsilon+0)(1+10)^*\)
Example

\[(1+10) \times \varepsilon \times 1 \times 10 \times \varepsilon \]
Example

Final NFA simplified slightly to reduce states
Last thought
Do all NFAs have a corresponding DFA?

Yes but it likely won't be pretty.
Do all NFAs have a corresponding DFA?

Yes but it likely won’t be pretty.