Pre-lecture brain teaser

Find the regular expression for the language containing all binary strings that do not contain the subsequence 111000.
ECE-374-B: Lecture 3 - NFAs

Instructor: Abhishek Kumar Umrawal
January 25, 2024

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

Find the regular expression for the language containing all binary strings that do not contain the subsequence 111000.

\[
0^* + 0^*10^* + 0^*10^*10^* + 0^*10^*10^*1^* + 0^*10^*10^*101^* + 0^*10^*10^*101^*01^*
\]
Does luck allow us to solve unsolvable problems?

- Regular
- Context-Free
- Context-sensitive
- Recursively-enumerable
- Non-recursively-enumerable
Does luck allow us to solve unsolvable problems? Consider two machines: $M_1$ and $M_2$

- $M_1$ is a classic deterministic machine.
- $M_2$ is a “lucky” machine that will always make the right choice.
Problem: Find shortest path from $a$ to $b$

Program on $M_1$ (Dijkstra's algorithm):

1. Initialize for each node $v$, $\text{Dist}(s, v) = d'(s, v) = \infty$
2. Initialize $X = \emptyset$, $d'(s, s) = 0$
3. For $i = 1$ to $|V|$ do
   - Let $v$ be node realizing $d'(s, v) = \min_{u \in V - X} d'(s, u)$
   - $\text{Dist}(s, v) = d'(s, v)$
   - $X = X \cup \{v\}$
   - Update $d'(s, u)$ for each $u$ in $V - X$ as follows:
     $$d'(s, u) = \min\left(d'(s, u), \text{Dist}(s, v) + \ell(v, u)\right)$$
Problem: Find shortest path from $a$ to $b$

Program on $M_2$ (Blind luck):

```python
path = []
current = a
While(not at b)
    take an outgoing edge from current node
    current = new location
    path += current
return path
```
Does luck allow us to solve unsolvable problems?
Consider two machines: $M_1$ and $M_2$

- $M_1$ is a classic deterministic machine.
- $M_2$ is a “lucky” machine that will always make the right choice.

Question:
Does luck allow us to solve unsolvable problems?
Consider two machines: $M_1$ and $M_2$

- $M_1$ is a classic deterministic machine.
- $M_2$ is a “lucky” machine that will always make the right choice.

**Question:** Are there problems which $M_2$ can solve that $M_1$ cannot.
Non-determinism in computing

In computer science, a nondeterministic machine is a theoretical device that can have more than one output for the same input.

A machine 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.
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.
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.
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: $L(N) = \{w \mid N$ accepts $w\}$. 
• Is $010110$ accepted?
NFA acceptance: Wait! what about the $\epsilon$?!
Is 010110 accepted?
NFA acceptance: Example

Is 010110 accepted?
• Is 010110 accepted? Yes
NFA acceptance: Example

- Is 010110 accepted? Yes
- Is 010 accepted? No
- Is 101 accepted? Yes
- Is 10011 accepted? Yes
- What is the language accepted by $N$? All strings with 101 or 11 as a substring.

Comment: Unlike DFAs, it is easier in NFAs to show that a string is accepted than to show that a string is not accepted.
• Is 010110 accepted? **Yes**
• Is 010 accepted? **No**
• Is 101 accepted? **Yes**
NFA acceptance: Example

- Is 010110 accepted? **Yes**
- Is 010 accepted? **No**
- Is 101 accepted? **Yes**
- Is 10011 accepted? **Yes**

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? Yes
- Is 010 accepted? No
- Is 101 accepted? Yes
- Is 10011 accepted? Yes
- What is the language accepted by $N$? All strings with 101 or 11 as a sub string.
NFA acceptance: Example

• Is 010110 accepted? **Yes**
• Is 010 accepted? **No**
• Is 101 accepted? **Yes**
• Is 10011 accepted? **Yes**
• What is the language accepted by $N$? **All strings with 101 or 11 as a sub string.**

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? Yes
- Is 010 accepted? No
- Is 101 accepted? Yes
- Is 10011 accepted? Yes
- What is the language accepted by $N$? All strings with 101 or 11 as a sub string.

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
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,
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,
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 $Q$),
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**

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

\[
\mathcal{P}(Q) = \left\{ \begin{array}{c}
\{1, 2, 3, 4\}, \\
\{2, 3, 4\}, \{1, 3, 4\}, \{1, 2, 4\}, \{1, 2, 3\}, \\
\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \\
\{1\}, \{2\}, \{3\}, \{4\}, \\
\{}\end{array} \right\}
\]
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 $Q$),
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

- $Q = \{ q_0, q_1, q_2, q_3 \}$
- $\Sigma = \{ 0, 1 \}$

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>${ q_0 }$</td>
<td>${ q_0 }$</td>
<td>${ q_0, q_1 }$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>${ q_1, q_2 }$</td>
<td>${ q_2 }$</td>
<td>${ } \varepsilon$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${ q_2 }$</td>
<td>${ } \varepsilon$</td>
<td>${ q_3 }$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>${ q_3 }$</td>
<td>${ q_3 }$</td>
<td>${ q_3 }$</td>
</tr>
</tbody>
</table>

- $s = q_0$
- $A = \{ q_3 \}$
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 $\varepsilon$-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.

![NFA diagram](attachment:image.png)

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

$\varepsilon$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 $w = \varepsilon$, $\delta^*(q, w) = \varepsilon$reach(q)
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($q$)
- if $w = a$ where $a \in \Sigma$:
  
  $\delta^*(q, a) = \epsilon$reach$\left( \bigcup_{p \in \epsilon$reach($q$)} \delta(p, a) \right)$
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$(q)$
- if $w = a$ where $a \in \Sigma$:
  \[
  \delta^*(q, a) = \epsilon\text{reach} \left( \bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a) \right)
  \]
- if $w = ax$:
  \[
  \delta^*(q, w) = \epsilon\text{reach} \left( \bigcup_{p \in \epsilon\text{reach}(q)} \left( \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right) \right)
  \]
Example of extended transition function

Find $\delta^* (q_0, 11)$:
Example of extended transition function

Find $\delta^*(q_0, 11)$:

$$\delta^*(q, w) = \epsilon\text{reach}\left( \bigcup_{p \in \epsilon\text{reach}(q)} \left( \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right) \right)$$
Example of extended transition function

We know \( w = 11 = ax \) so \( a = 1 \) and \( x = 1 \)

\[
\delta^*(q_0, 11) = \varepsilon\text{reach}\left( \bigcup_{p \in \varepsilon\text{reach}(q_0)} \left( \bigcup_{r \in \delta^*(p, 1)} \delta^*(r, 1) \right) \right)
\]
Example of extended transition function

$\varepsilon \text{reach}(q_0) = \{q_0\}$

$\delta^*(q_0, 11) = \varepsilon \text{reach} \left( \bigcup_{p \in \{q_0\}} \left( \bigcup_{r \in \delta^*(p, 1)} \delta^*(r, 1) \right) \right)$
Example of extended transition function

\[ \delta^*(q_0, 11) = \varepsilon \text{reach} \left( \bigcup_{r \in \delta^* \left( \{q_0\}, 1 \right)} \delta^* (r, 1) \right) \]
Example of extended transition function

Need $\delta^*(q_0, 1) = \epsilon\text{reach}(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a)) = \epsilon\text{reach}(\delta(q_0, 1))$: 

$= \epsilon\text{reach}(\{q_0, q_1\}) = \{q_0, q_1, q_2\}$

$\delta^*(q_0, 11) = \epsilon\text{reach}\left(\bigcup_{r \in \delta^*(\{q_0\}, 1)} \delta^*(r, 1)\right)$
Example of extended transition function

Need
\[
\delta^*(q_0, 1) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q_0)} \delta(p, a)\right) = \epsilon\text{reach}(\delta(q_0, 1)):
\]
\[
= \epsilon\text{reach}\{q_0, q_1\} = \{q_0, q_1, q_2\}
\]
\[
\delta^*(q_0, 11) = \epsilon\text{reach}\left(\bigcup_{r \in \{q_0, q_1, q_2\}} \delta^*(r, 1)\right)
\]
Example of extended transition function

\[ \delta^*(q_0, 11) = \varepsilon \text{reach}(\delta^*(q_0, 1) \cup \delta^*(q_1, 1) \cup \delta^*(q_2, 1)) \]
Transition for strings: \( w = ax \)

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

- \( R = \epsilon\text{reach}(q) \)

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

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

- \( \delta^*(q, w) = \epsilon\text{reach}\left( \bigcup_{r \in N} \delta^*(r, x) \right) \)
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

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

- \( \delta^*(s, \epsilon) = \{s, d, a\} \)
Example

What is:

- \( \delta^*(s, \varepsilon) = \{s, d, a\} \)
- \( \delta^*(s, 0) = \{s, d, a, b\} \)
What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 0) = \{d, a, c, g\}$
What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 0) = \{d, a, c, g\}$
- $\delta^*(b, 00) = \{b, g\}$
Constructing generalized 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
Strings that represent decimal numbers.
Examples: 154, 345.75332, 534677567.1
Example

\[ L = \{ \text{bitstrings that have a 1 three positions from the end} \} \]
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 $L = 0^* + 1^*$. 

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 $L(N) = L(N_1) \cup L(N_2)$.

True. Single start state with $\varepsilon$-transition to $q_1$ and $q_2$. 
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 $L(N) = L(N_1) \cdot L(N_2)$.

True. $f_1$ connected to $q_2$ and $f_2$ as accept state.
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 $L(N) = (L(N_1))^*$. 

Does not work! Why?
Because Kleene star must include $\varepsilon$ but if we turn the initial state into an accept state, we are inserting a $\varepsilon$ into $L(N)$ where there might not be one. Imagine:

![Diagram of a simple DFA](image-url)
Theorem
For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^*$. 
NFAs capture Regular Languages
(ε+0)(1+10)*
Final NFA simplified slightly to reduce states

Example
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.