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: Nickvash Kani
August 31, 2023
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.
Find the regular expression for the language containing all binary strings that do not contain the substring 101010
Find the regular expression for the language containing all binary strings that **do not** contain the substring **101010**
Find the regular expression for the language contains all binary strings whose $\#_0(w) \% 7 = 0$(number of 0’s divisible by 7).
Find the regular expression for the language contains all binary strings whose $\#_0(w) \% 7 = 0$ (number of 0’s divisible by 7).
Pre-lecture brain teaser III

Show that the following string \( w \) is a member of the language that:

- does not contain the subsequence 111000 or
- does not contain the substring 101010 or
- or has a number of 0’s divisible by 7

\[ w = 1001110110111001 \]
\[ 1000010111110010 \]
\[ 0101010011001111 \]
\[ 1001001011111100 \]
Show that the following string \( w \) is a member of the language that:

- does not contain the subsequence \( 111000 \) or
- does not contain the substring \( 101010 \) or
- or has a number of 0’s divisible by 7

\[
\begin{align*}
w &= 10011101110111001 \\
&
\quad 100010111110010 \\
&
\quad 0101010011001111 \\
&
\quad 1001001011111100 \\
\end{align*}
\]

You have 30 seconds.
Pre-lecture brain teaser III

Show that the following string ($w$) is a member of the language that:

- does not contain the subsequence $111000$ or
- does not contain the substring $101010$ or
- or has a number of 0’s divisible by 7

\[ w = 1001110110111001010001011110011001001001011111000100110110111001 \]

You have 30 seconds. Pray, choose a strategy and hope you get lucky.
Does luck allow us to solve unsolvable problems?
Does luck allow us to solve unsolvable problems? New example: 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):

- Initialize for each node $v$, $\text{Dist}(s, v) = d'(s, v) = \infty$
- Initialize $X = \emptyset$, $d'(s, s) = 0$

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):

```
Initialize path = []
path += a
While(not at b)
    take an outgoing edge $(u, v)$ from current node $u$ to $v$
    current = v
    path += v
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.
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.

The notion was first posed by Robert W. Floyd in 1967.
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.
Non-determinism in media

Placeholder slide for youtube.
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.

![Diagram of a non-deterministic finite state automaton](image-url)
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: Wait! what about the $\varepsilon$?!
NFA acceptance: Example

Is $010110$ accepted?
NFA acceptance: Example

Is 010110 accepted?
• Is 010110 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?

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?
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.
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.
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

- $Q$ is a finite set whose elements are called states,
- $\Sigma$ is a finite set called the input alphabet,
- $\delta: Q \times \Sigma \cup \{\epsilon\} \rightarrow P(Q)$ is the transition function (here $P(Q)$ 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,
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,
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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow P(Q)$ is the transition function (here $P(Q)$ 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 $Q$),
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,
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \rightarrow P(Q)$ is the transition function (here $P(Q)$ 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,
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 $q$. 
Extending the transition function to strings

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

![Diagram of an NFA](attachment:diagram.png)

Proofs/oracles.

Finally, we can treat NFAs not as a mechanism for computing something, but as a tool for proving theorems. If an NFA reads a string that is not in its language, it 'dies' in some sense, as it somehow chose a path to an accept state. Hence, the NFA somehow chose a path to an accept state still exist. One slight disadvantage of this metaphor is that if an NFA reads a string that is not in its language, it 'dies' in some sense, as it somehow chose a path to an accept state.
Definition
For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon$-reach($q$) is the set of all states that $q$ can reach using only $\epsilon$-transitions.

Definition
For $X \subseteq Q$: $\epsilon$-reach($X$) = $\bigcup_{x \in X} \epsilon$-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

$\epsilon$\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

$\epsilon$-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 \( \delta^* (q_0, 11) \):

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

![Diagram of a finite automaton with transitions labeled 0, 1, and ε.]

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

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

\[
\epsilon \text{reach}(q_0) = \{q_0\}
\]

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

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

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

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

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

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

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

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

Simplify

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

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

- \( R = \varepsilon\text{reach}(q) \implies \)

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

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

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

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

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

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.
Example

What is:

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

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

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

- \( \delta^*(s, \epsilon) = \)
- \( \delta^*(s, 0) = \)
- \( \delta^*(b, 0) = \)
- \( \delta^*(b, 00) = \)
Constructing generalized 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

$L = \{\text{bitstrings that have a 1 three positions from the end}\}$
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
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 $L(N) = L(N_1) \cup L(N_2)$. 
Closure under concatenation

**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 $L(N) = L(N_1) \cdot L(N_2)$. 
Closure under Kleene star

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

![Diagram of NFA with start state $q_1$ and final state $f_1$]
Closure under Kleene star

**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 $L(N) = (L(N_1))^\ast$.

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

All these examples are examples of language *transformations*. A language transformation is one where you take one class or languages, perform some operation and get a new language that belongs to that same class (closure). Tomorrow’s lab will go over more examples of language transformations.
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.