Non-deterministic Finite Automata (NFAs)

Lecture 4
Thursday, January 24, 2019
Part I

NFA Introduction
Non-deterministic Finite State Automata (NFAs)

Differences from DFA
- From state $q$ on same letter $a \in \Sigma$ multiple possible states
- No transitions from $q$ on some letters
- $\varepsilon$-transitions!

Questions:
- Is this a “real” machine?
- What does it do?
Non-deterministic Finite State Automata (NFAs)

Differences from DFA
- From state $q$ on same letter $a \in \Sigma$ multiple possible states
- No transitions from $q$ on some letters
- $\varepsilon$-transitions!

Questions:
- Is this a “real” machine?
- What does it do?
Non-deterministic Finite State Automata (NFAs)

Differences from DFA
- From state $q$ on same letter $a \in \Sigma$ multiple possible states
- No transitions from $q$ on some letters
- $\epsilon$-transitions!

Questions:
- Is this a “real” machine?
- What does it do?
NFA behavior

Machine on input string $w$ from state $q$ can lead to set of states (could be empty)

- From $q_\varepsilon$ on 1
- From $q_\varepsilon$ on 0
- From $q_0$ on $\varepsilon$
- From $q_\varepsilon$ on 01
- From $q_{00}$ on 00
Machine on input string $w$ from state $q$ can lead to set of states (could be empty)

- From $q_{\epsilon}$ on 1
- From $q_{\epsilon}$ on 0
- From $q_0$ on $\epsilon$
- From $q_{\epsilon}$ on 01
- From $q_{00}$ on 00
NFA behavior

Machine on input string $w$ from state $q$ can lead to set of states (could be empty)

- From $q_\varepsilon$ on 1
- From $q_\varepsilon$ on 0
- From $q_0$ on $\varepsilon$
- From $q_\varepsilon$ on 01
- From $q_{00}$ on 00
Machine on input string $w$ from state $q$ can lead to set of states (could be empty):

- From $q_\epsilon$ on 1
- From $q_\epsilon$ on 0
- From $q_0$ on $\epsilon$
- From $q_\epsilon$ on 01
- From $q_{00}$ on 00
Machine on input string $w$ from state $q$ can lead to set of states (could be empty)

- From $q_\varepsilon$ on $1$
- From $q_\varepsilon$ on $0$
- From $q_0$ on $\varepsilon$
- From $q_\varepsilon$ on $01$
- From $q_{00}$ on $00$
NFA behavior

Machine on input string $w$ from state $q$ can lead to set of states (could be empty)

- From $q_\varepsilon$ on 1
- From $q_\varepsilon$ on 0
- From $q_0$ on $\varepsilon$
- From $q_\varepsilon$ on 01
- From $q_{00}$ on 00
**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: $L(N) = \{w \mid N \text{ accepts } w\}$. 
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 $w \}$. 
NFA acceptance: example

- Is 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in 1*01 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 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in $1^*01$ 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.
Is 01 accepted?
Is 001 accepted?
Is 100 accepted?
Are all strings in 1*01 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 01 accepted?
Is 001 accepted?
Is 100 accepted?
Are all strings in 1*01 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.
Is 01 accepted?
Is 001 accepted?
Is 100 accepted?
Are all strings in 1*01 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 01 accepted?
Is 001 accepted?
Is 100 accepted?
Are all strings in 1*01 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 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in 1*01 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.
Run it on input \textit{ababa}.

Idea: Keep track of the states where the NFA might be at any given time.
Simulating NFA

Example the first

$t = 0$:

Remaining input: \textit{ababa}.
Simulating NFA

Example the first

$t = 0$:

Remaining input: $ababa$.

$t = 1$:

Remaining input: $baba$. 
Simulating NFA

Example the first

$t = 1$:

Remaining input: $baba$. 

\[
\begin{array}{c}
A \xrightarrow{a,b} B \xrightarrow{a} C \xrightarrow{a} D \xrightarrow{b} E
\end{array}
\]
Simulating NFA

Example the first

$t = 1$:

Remaining input: $baba$.

$t = 2$:

Remaining input: $aba$. 

Chan, Har-Peled, Hassanieh (UIUC) CS374 Spring 2019 7/39
Simulating NFA

Example the first

$t = 2$:

Remaining input: $aba$. 
Simulating NFA

Example the first

$t = 2$:

Remaining input: \textit{aba}.

$t = 3$:

Remaining input: \textit{ba}.
Simulating NFA

Example the first

$t = 3$: 

\[ \begin{array}{c}
A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{a} D \xrightarrow{b} E \\
\end{array} \]

Remaining input: \textit{ba}.
Simulating NFA

Example the first

$t = 3$:

Remaining input: $ba$.

$t = 4$:

Remaining input: $a$. 
Simulating NFA

Example the first

$t = 4$:

Remaining input: $a$. 
Simulating NFA

Example the first

$t = 4$:

Remaining input: $a$.

$t = 5$:

Remaining input: $\varepsilon$. 
Simulating NFA

Example the first

$t = 5$:

Remaining input: $\varepsilon$.

Accepts: $ababa$. 
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.
Reminder: Power set

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

Example

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

$\mathcal{P}(Q) = \{ \{1, 2, 3, 4\}, \{2, 3, 4\}, \{1, 3, 4\}, \{1, 2, 4\}, \{1, 2, 3\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1\}, \{2\}, \{3\}, \{4\}, \{\} \}$
\[ Q = \{ q_\varepsilon, q_0, q_{00}, q_p \} \]
\[ \Sigma = \{ 0, 1 \} \]
\[ \delta \]
\[ s = q_\varepsilon \]
\[ A = \{ q_p \} \]
Example

- $Q = \{q_\varepsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- $s = q_\varepsilon$
- $A = \{q_p\}$
Example

Q = \{q_\varepsilon, q_0, q_{00}, q_p\}

\(\Sigma = \{0, 1\}\)

\(s = q_\varepsilon\)

A = \{q_p\}
Example

\[ Q = \{ q_\epsilon, q_0, q_{00}, q_p \} \]

\[ \Sigma = \{0, 1\} \]

\[ \delta \]

\[ s = q_\epsilon \]

\[ A = \{ q_p \} \]
Example

- $Q = \{q_\varepsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- $\delta$
- $s = q_\varepsilon$
- $A = \{q_p\}$
Example

- \( Q = \{q_\varepsilon, q_0, q_{00}, q_p\} \)
- \( \Sigma = \{0, 1\} \)
- \( \delta \)
- \( s = q_\varepsilon \)
- \( A = \{q_p\} \)
Example

\[ Q = \{q_\varepsilon, q_0, q_{00}, q_p\} \]

\[ \Sigma = \{0, 1\} \]

\[ \delta \]

\[ s = q_\varepsilon \]

\[ A = \{q_p\} \]
Example

- \( Q = \{ q_\epsilon, q_0, q_{00}, q_p \} \)
- \( \Sigma = \{ 0, 1 \} \)
- \( \delta \)
- \( s = q_\epsilon \)
- \( A = \{ q_p \} \)
Example

- $Q = \{q_\varepsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- $\delta$
- $s = q_\varepsilon$
- $A = \{q_p\}$
Example

Transition function in detail...

\[
\delta(q_\varepsilon, \varepsilon) = \{ q_\varepsilon \}
\]
\[
\delta(q_\varepsilon, 0) = \{ q_\varepsilon, q_0 \}
\]
\[
\delta(q_\varepsilon, 1) = \{ q_\varepsilon \}
\]
\[
\delta(q_{00}, \varepsilon) = \{ q_{00} \}
\]
\[
\delta(q_{00}, 0) = \{ \}
\]
\[
\delta(q_{00}, 1) = \{ q_p \}
\]

\[
\delta(q_0, \varepsilon) = \{ q_0, q_{00} \}
\]
\[
\delta(q_0, 0) = \{ q_{00} \}
\]
\[
\delta(q_0, 1) = \{ \}
\]

\[
\delta(q_{p}, \varepsilon) = \{ q_p \}
\]
\[
\delta(q_{p}, 0) = \{ q_p \}
\]
\[
\delta(q_{p}, 1) = \{ q_p \}
\]
Extending the transition function to strings

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

1. **NFA** $N = (Q, \Sigma, \delta, s, A)$

2. $\delta(q, a)$: set of states that $N$ can go to from $q$ on reading $a \in \Sigma \cup \{\varepsilon\}$.

3. Want transition function $\delta^*: Q \times \Sigma^* \to \mathcal{P}(Q)$

4. $\delta^*(q, w)$: set of states reachable on input $w$ starting in state $q$. 

Chan, Har-Peled, Hassanieh (UIUC)
Extending the transition function to strings

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

1. **NFA** \( N = (Q, \Sigma, \delta, s, A) \)

2. \( \delta(q, a) \): set of states that \( N \) can go to from \( q \) on reading \( a \in \Sigma \cup \{\varepsilon\} \).

3. Want transition function \( \delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \)

4. \( \delta^*(q, w) \): 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 $\epsilon\text{reach}(q)$ is the set of all states that $q$ can reach using only $\epsilon$-transitions.
Extending the transition function to strings

**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**
Inductive definition of $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:
- if $w = \varepsilon$, $\delta^*(q, w) = \varepsilon$reach$(q)$
- if $w = a$ where $a \in \Sigma$
  $\delta^*(q, a) = \bigcup_{p \in \varepsilon$reach$(q)} \big( \bigcup_{r \in \delta(p,a)} \varepsilon$reach$(r) \big)$
- if $w = ax$
  $\delta^*(q, w) = \bigcup_{p \in \varepsilon$reach$(q)} \big( \bigcup_{r \in \delta(p,a)} \delta^*(r, x) \big)$
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.

**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 $a \in \Sigma$
  
  $$\delta^*(q, a) = \bigcup_{p \in \epsilon$-reach($q$)} \left( \bigcup_{r \in \delta(p, a)} \epsilon$-reach($r$) \right)$$
- If $w = ax$,
  
  $$\delta^*(q, w) = \bigcup_{p \in \epsilon$-reach($q$)} \left( \bigcup_{r \in \delta(p, a)} \delta^*(r, x) \right)$$
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.

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 $a \in \Sigma$
  $\delta^*(q, a) = \bigcup_{p \in \epsilon$-reach($q$)} \left( \bigcup_{r \in \delta(p, a) \epsilon$-reach($r$)} \right)$
- if $w = ax$, $\delta^*(q, w) = \bigcup_{p \in \epsilon$-reach($q$)} \left( \bigcup_{r \in \delta(p, a) \delta^*(r, x)} \right)$
Formal definition of language accepted by $\mathcal{N}$

**Definition**

A string $w$ is accepted by NFA $\mathcal{N}$ if $\delta^*_\mathcal{N}(s, w) \cap A \neq \emptyset$.

**Definition**

The language $L(\mathcal{N})$ accepted by a NFA $\mathcal{N} = (Q, \Sigma, \delta, s, A)$ 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.
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:

- $\delta^*(s, \epsilon)$
- $\delta^*(s, 0)$
- $\delta^*(c, 0)$
- $\delta^*(b, 00)$
What is:

- $\delta^*(s, \epsilon)$
- $\delta^*(s, 0)$
- $\delta^*(c, 0)$
- $\delta^*(b, 00)$
Example

What is:

- \( \delta^*(s, \epsilon) \)
- \( \delta^*(s, 0) \)
- \( \delta^*(c, 0) \)
- \( \delta^*(b, 00) \)
What is:

- $\delta^*(s, \epsilon)$
- $\delta^*(s, 0)$
- $\delta^*(c, 0)$
- $\delta^*(b, 00)$
Another definition of computation

**Definition**

$q \xrightarrow{w}_N p$: State $p$ of NFA $N$ is **reachable** from $q$ on $w$ if there exists a sequence of states $r_0, r_1, \ldots, r_k$ and a sequence $x_1, x_2, \ldots, x_k$ where $x_i \in \Sigma \cup \{\varepsilon\}$, for each $i$, such that:

- $r_0 = q$,
- for each $i$, $r_{i+1} \in \delta(r_i, x_{i+1})$,
- $r_k = p$, and
- $w = x_1 x_2 x_3 \cdots x_k$.

**Definition**

$\delta^* N(q, w) = \left\{ p \in Q \mid q \xrightarrow{w}_N p \right\}$.
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.
Part II

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.
Strings that represent decimal numbers.
Example

- \{\text{strings that contain CS374 as a substring}\}
- \{\text{strings that contain CS374 or CS473 as a substring}\}
- \{\text{strings that contain CS374 and CS473 as substrings}\}
Example

- \{\text{strings that contain CS374 as a substring}\}
- \{\text{strings that contain CS374 or CS473 as a substring}\}
- \{\text{strings that contain CS374 and CS473 as substrings}\}
Example

- \{\text{strings that contain CS374 as a substring}\}
- \{\text{strings that contain CS374 or CS473 as a substring}\}
- \{\text{strings that contain CS374 and CS473 as substrings}\}
$L_k = \{\text{bitstrings that have a 1 \(k\) 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 $f$
Part III

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
Closure under union

Theorem

For any two NFA's $N_1$ and $N_2$ there is a NFA $N$ such that
$L(N) = L(N_1) \cup L(N_2)$.
Closure under union

**Theorem**

For any two NFA $N_1$ 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 NFA\(s\) \(N_1\) and \(N_2\) there is a NFA \(N\) such that \(L(N) = L(N_1) \cdot L(N_2)\).
**Theorem**

For any two NFA's $N_1$ and $N_2$ there is a NFA $N$ such that $L(N) = L(N_1) \cdot L(N_2)$. 

![Diagram of NFA's $N_1$ and $N_2$](image)
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?
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?
Closure under Kleene star

Theorem

For any NFA $N_1$ there is a NFA $N$ such that $L(N) = (L(N_1))^\ast$.
Part IV

NFA's capture Regular Languages
Regular Languages Recap

### Regular Languages

- $\emptyset$ regular
- $\{\epsilon\}$ regular
- $\{a\}$ regular for $a \in \Sigma$
- $R_1 \cup R_2$ regular if both are
- $R_1 R_2$ regular if both are
- $R^*$ is regular if $R$ is

### Regular Expressions

- $\emptyset$ denotes $\emptyset$
- $\epsilon$ denotes $\{\epsilon\}$
- $a$ denote $\{a\}$
- $r_1 + r_2$ denotes $R_1 \cup R_2$
- $r_1 r_2$ denotes $R_1 R_2$
- $r^*$ denote $R^*$

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language.
Theorem

For every regular language $L$ there is an NFA $N$ such that $L = L(N)$.

Proof strategy:

- For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$
- Induction on length of $r$
For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$

Induction on length of $r$

Base cases: $\emptyset$, $\{\varepsilon\}$, $\{a\}$ for $a \in \Sigma$. 

NFA\textsc{s} and Regular Language

- For every regular expression $r$ show that there is a \textbf{NFA} $N$ such that $L(r) = L(N)$
- Induction on length of $r$

\textbf{Inductive cases:}

- $r_1, r_2$ regular expressions and $r = r_1 + r_2$.
  - By induction there are \textbf{NFAs} $N_1, N_2$ s.t $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is \textbf{NFA} $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$
- $r = r_1 \cdot r_2$. Use closure of \textbf{NFA} languages under concatenation
- $r = (r_1)^*$. Use closure of \textbf{NFA} languages under Kleene star
NFAs and Regular Language

- For every regular expression \( r \) show that there is a NFA \( N \) such that \( L(r) = L(N) \)

- Induction on length of \( r \)

**Inductive cases:**

- \( r_1, r_2 \) regular expressions and \( r = r_1 + r_2 \).
  By induction there are NFAs \( N_1, N_2 \) s.t \( L(N_1) = L(r_1) \) and \( L(N_2) = L(r_2) \). We have already seen that there is NFA \( N \) s.t \( L(N) = L(N_1) \cup L(N_2) \), hence \( L(N) = L(r) \)

- \( r = r_1 \cdot r_2 \). Use closure of NFA languages under concatenation

- \( r = (r_1)^* \). Use closure of NFA languages under Kleene star
NFA s and Regular Language

For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$

**Induction on length of $r$**

**Inductive cases:**

- $r_1, r_2$ regular expressions and $r = r_1 + r_2$.
  
  By induction there are NFA s $N_1, N_2$ s.t $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$

- $r = r_1 \cdot r_2$. Use closure of NFA languages under concatenation

- $r = (r_1)^*$. Use closure of NFA languages under Kleene star
NFA$s and Regular Language

- For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$
- Induction on length of $r$

**Inductive cases:**

- $r_1, r_2$ regular expressions and $r = r_1 + r_2$. By induction there are NFA$s $ $N_1, N_2$ s.t $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$

- $r = r_1 \cdot r_2$. Use closure of NFA languages under concatenation

- $r = (r_1)^*$. Use closure of NFA languages under Kleene star
NFA and Regular Language

For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$

Induction on length of $r$

Inductive cases:

- $r_1, r_2$ regular expressions and $r = r_1 + r_2$. By induction there are NFA $N_1, N_2$ s.t $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$

- $r = r_1 \cdot r_2$. Use closure of NFA languages under concatenation

- $r = (r_1)^*$. Use closure of NFA languages under Kleene star
For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$

Induction on length of $r$

Inductive cases:

- $r_1, r_2$ regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs $N_1, N_2$ s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$

- $r = r_1 \cdot r_2$. Use closure of NFA languages under concatenation

- $r = (r_1)^*$. Use closure of NFA languages under Kleene star
For every regular expression $r$ show that there is a NFA $N$ such that $L(r) = L(N)$

Inductive cases:

- $r_1, r_2$ regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs $N_1, N_2$ s.t $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA $N$ s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$

- $r = r_1 \cdot r_2$. Use closure of NFA languages under concatenation

- $r = (r_1)^*$. Use closure of NFA languages under Kleene star
Example

\[(ε+0)(1+10)^*\]
Example

\[ (1+10)^* \]
Example

Final NFA simplified slightly to reduce states