NFAs continued, Closure
Properties of Regular Languages

Lecture 5
Wednesday, February 5, 2020
Regular Languages, DFAs, NFAs

Theorem

Languages accepted by **DFA**s, **NFA**s, and regular expressions are the same.
Regular Languages, DFAs, NFAs

Theorem

Languages accepted by **DFAs, NFAs, and regular expressions are the same.**

- DFAs are special cases of NFAs (trivial)
- NFAs accept regular expressions (we saw already)
- DFAs accept languages accepted by NFAs (today)
- Regular expressions for languages accepted by DFAs (later in the course)
Part I

Equivalence of NFAs and DFAs
Theorem

For every NFA $N$ there is a DFA $M$ such that $L(M) = L(N)$.
Formal Tuple Notation for 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 $Q$),
- $s \in Q$ is the **start state**,
- $A \subseteq Q$ is the set of **accepting/final states**.

$\delta(q, a)$ for $a \in \Sigma \cup \{\epsilon\}$ is a subset of $Q$ — a set of states.
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 $\delta^* : Q \times \Sigma^* \rightarrow P(Q)$:
- 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 $w = xa$
  $\delta^*(q, w) = \bigcup_{p \in \delta^*(q, x)} \left( \bigcup_{r \in \delta(p, a) \epsilon$-reach$(r)} \right)$
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

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$
Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA $N$ on input $w$.
- What does it need to store after seeing a prefix $x$ of $w$?

It needs to know at least $\delta^*(s, x)$, the set of states that $N$ could be in after reading $x$.

Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol $a$ in the input.

When should the program accept a string $w$? If $\delta^*(s, w) \cap A \neq \emptyset$.

Key Observation: A DFA $M$ that simulates $N$ should keep in its memory/state the set of states of $N$. Thus the state space of the DFA should be $\mathcal{P}(Q)$.
Simulating an NFA by a DFA

Think of a program with fixed memory that needs to simulate NFA $\mathcal{N}$ on input $w$.

What does it need to store after seeing a prefix $x$ of $w$?

It needs to know at least $\delta^*(s, x)$, the set of states that $\mathcal{N}$ could be in after reading $x$.

Is it sufficient?
Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA $N$ on input $w$.
- What does it need to store after seeing a prefix $x$ of $w$?
- It needs to know at least $\delta^*(s, x)$, the set of states that $N$ could be in after reading $x$.
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol $a$ in the input.
- When should the program accept a string $w$?
Think of a program with fixed memory that needs to simulate NFA $N$ on input $w$.

What does it need to store after seeing a prefix $x$ of $w$?

It needs to know at least $\delta^*(s, x)$, the set of states that $N$ could be in after reading $x$.

Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol $a$ in the input.

When should the program accept a string $w$? If $\delta^*(s, w) \cap A \neq \emptyset$.

**Key Observation:** A DFA $M$ that simulates $N$ should keep in its memory/state the set of states of $N$.

Thus the state space of the DFA should be $\mathcal{P}(Q)$.
NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = \mathcal{P}(Q)$
Subset Construction

NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon\text{reach}(s) = \delta^*(s, \epsilon)$
NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon\text{reach}(s) = \delta^*(s, \epsilon)$
- $A' = \{ X \subseteq Q \mid X \cap A \neq \emptyset \}$
Subset Construction

NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon\text{reach}(s) = \delta^*(s, \epsilon)$
- $A' = \{X \subseteq Q \mid X \cap A \neq \emptyset\}$
- $\delta'(X, a) = \bigcup_{q \in X} \delta^*(q, a)$ for each $X \subseteq Q$, $a \in \Sigma$. 
Example

No $\epsilon$-transitions

$q_0$ \rightarrow 1 \rightarrow q_1$

No $\epsilon$-transitions

$q_0 \rightarrow 0, 1 \rightarrow q_1$

Thus, to simulate the NFA, the DFA only needs to maintain the current set of states of the NFA.
Example

No $\epsilon$-transitions

An example NFA is shown in Figure 4 along with the DFA $\text{det}(N)$ in Figure 5.

We will now prove that the DFA defined above is correct. That is

Lemma 4. $L(N) = L(\text{det}(N))$

Proof. Need to show $\forall w \in \Sigma^*$. $\text{det}(N)$ accepts $w$ if and only if $N$ accepts $w$.

Again for the induction proof to go through we need to strengthen the claim as follows.

$\forall w \in \Sigma^*$. $\text{det}(N)(s_0, w) = \{q_0, q_1\}$ if and only if $N(s, w) = \{q_0, q_1\}$.

In other words, this says that the state of the DFA after reading some string is exactly the set of states the NFA could be in after reading the same string.
Simulating NFA

Example the first revisited

Previous lecture.. Ran

NFA\(^{(N1)}\)

on input \textit{ababa}.

\begin{align*}
t = 0: & \quad A \xrightarrow{a,b} B \xrightarrow{a,b} C \xrightarrow{a,b} D \xrightarrow{a,b} E \\
t = 1: & \quad A \xrightarrow{a,b} B \xrightarrow{a,b} C \xrightarrow{a,b} D \xrightarrow{a,b} E \\
t = 2: & \quad A \xrightarrow{a,b} B \xrightarrow{a,b} C \xrightarrow{a,b} D \xrightarrow{a,b} E \\
t = 3: & \quad A \xrightarrow{a,b} B \xrightarrow{a,b} C \xrightarrow{a,b} D \xrightarrow{a,b} E \\
t = 4: & \quad A \xrightarrow{a,b} B \xrightarrow{a,b} C \xrightarrow{a,b} D \xrightarrow{a,b} E \\
t = 5: & \quad A \xrightarrow{a,b} B \xrightarrow{a,b} C \xrightarrow{a,b} D \xrightarrow{a,b} E
\end{align*}
Example: DFA from NFA

NFA:

\[(N1) \quad A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{a} D \xrightarrow{b} E\]

DFA:
Incremental construction

Only build states reachable from $s' = \varepsilon \text{reach}(s)$ the start state of $M$
Incremental construction

Only build states reachable from \( s' = \varepsilon \text{reach}(s) \) the start state of \( M \)

\[
\delta'(X, a) = \bigcup_{q \in X} \delta^*(q, a)
\]
Incremental algorithm

- Build $M$ beginning with start state $s' == \epsilon \text{reach}(s)$
- For each existing state $X \subseteq Q$ consider each $a \in \Sigma$ and calculate the state $Y = \delta'(X, a) = \bigcup_{q \in X} \delta^*(q, a)$ and add a transition.
- If $Y$ is a new state add it to reachable states that need to be explored.

To compute $\delta^*(q, a)$ - set of all states reached from $q$ on string $a$
- Compute $X = \epsilon \text{reach}(q)$
- Compute $Y = \bigcup_{p \in X} \delta(p, a)$
- Compute $Z = \epsilon \text{reach}(Y) = \bigcup_{r \in Y} \epsilon \text{reach}(r)$
Proof of Correctness

**Theorem**

Let $N = (Q, \Sigma, s, \delta, A)$ be a **NFA** and let $M = (Q', \Sigma, \delta', s', A')$ be a **DFA** constructed from $N$ via the subset construction. Then $L(N) = L(M)$.

**Stronger claim:**

**Lemma**

For every string $w$, $\delta^* N(s, w) = \delta^* M(s', w)$.

**Proof by induction on $|w|$.

**Base case:** $w = \epsilon$.

$\delta^* N(s, \epsilon) = \epsilon$ reach $(s)$.

$\delta^* M(s', \epsilon) = s' = \epsilon$ reach $(s)$ by definition of $s'$.
Proof of Correctness

**Theorem**

Let \( N = (Q, \Sigma, s, \delta, A) \) be a NFA and let \( M = (Q', \Sigma, \delta', s', A') \) be a DFA constructed from \( N \) via the subset construction. Then \( L(N) = L(M) \).

**Stronger claim:**

**Lemma**

For every string \( w \), \( \delta^*_N(s, w) = \delta^*_M(s', w) \).

Proof by induction on \( |w| \).

**Base case:** \( w = \epsilon \).

\( \delta^*_N(s, \epsilon) = \epsilon \text{reach}(s) \).

\( \delta^*_M(s', \epsilon) = s' = \epsilon \text{reach}(s) \) by definition of \( s' \).
Lemma

For every string \( w \), \( \delta_N^*(s, w) = \delta_M^*(s', w) \).

Inductive step: \( w = xa \) (Note: suffix definition of strings)
\[
\delta_N^*(s, xa) = \bigcup_{p \in \delta_N^*(s, x)} \delta_N^*(p, a)
\]
by inductive definition of \( \delta_N^* \)
Lemma

For every string $w$, $\delta^*_N(s, w) = \delta^*_M(s', w)$.

**Inductive step:** $w = xa$ (Note: suffix definition of strings)

$\delta^*_N(s, xa) = \bigcup_{p \in \delta^*_N(s, x)} \delta^*_N(p, a)$ by inductive definition of $\delta^*_N$

$\delta^*_M(s', xa) = \delta_M(\delta^*_M(s, x), a)$ by inductive definition of $\delta^*_M$
Lemma

For every string $w$, $\delta^*_N(s, w) = \delta^*_M(s', w)$.

**Inductive step:** $w = xa$  
(Note: suffix definition of strings) 

$\delta^*_N(s, xa) = \bigcup_{p \in \delta^*_N(s, x)} \delta^*_N(p, a)$ by inductive definition of $\delta^*_N$ 

$\delta^*_M(s', xa) = \delta_M(\delta^*_M(s, x), a)$ by inductive definition of $\delta^*_M$

By inductive hypothesis: $Y = \delta^*_N(s, x) = \delta^*_M(s, x)$
Lemma

For every string $w$, $\delta^*_N(s, w) = \delta^*_M(s', w)$.

Inductive step: $w = xa$  
(Note: suffix definition of strings)

\[ \delta^*_N(s, xa) = \bigcup_{p \in \delta^*_N(s, x)} \delta^*_N(p, a) \]
by inductive definition of $\delta^*_N$

\[ \delta^*_M(s', xa) = \delta_M(\delta^*_M(s, x), a) \]
by inductive definition of $\delta^*_M$

By inductive hypothesis: $Y = \delta^*_N(s, x) = \delta^*_M(s, x)$

Thus $\delta^*_N(s, xa) = \bigcup_{p \in Y} \delta^*_N(p, a) = \delta_M(Y, a)$  
by definition of $\delta_M$. 

Therefore, $\delta^*_N(s, xa) = \delta_M(Y, a) = \delta^*_M(s', xa)$
which is what we need.
Lemma

For every string $w$, $\delta^*_N(s, w) = \delta^*_M(s', w)$.

**Inductive step:** $w = xa$  
(Note: suffix definition of strings)

$\delta^*_N(s, xa) = \bigcup_{p \in \delta^*_N(s, x)} \delta^*_N(p, a)$ by inductive definition of $\delta^*_N$

$\delta^*_M(s', xa) = \delta_M(\delta^*_M(s, x), a)$ by inductive definition of $\delta^*_M$

By inductive hypothesis: $Y = \delta^*_N(s, x) = \delta^*_M(s, x)$

Thus $\delta^*_N(s, xa) = \bigcup_{p \in Y} \delta^*_N(p, a) = \delta_M(Y, a)$ by definition of $\delta_M$.

Therefore, $\delta^*_N(s, xa) = \delta_M(Y, a) = \delta_M(\delta^*_M(s, x), a) = \delta^*_M(s', xa)$

which is what we need.
Example: DFA from NFA

NFA: \((N1)\)

DFA:
Part II

Closure Properties of Regular Languages
Regular Languages

Regular languages have three different characterizations

- Inductive definition via base cases and closure under union, concatenation and Kleene star
- Languages accepted by DFA
- Languages accepted by NFA

Regular language closed under many operations: union, concatenation, Kleene star via inductive definition or NFA complement, union, intersection via DFA homomorphism, inverse homomorphism, reverse, ...

Different representations allow for flexibility in proofs
Regular Languages

Regular languages have three different characterizations

- Inductive definition via base cases and closure under union, concatenation and Kleene star
- Languages accepted by DFAs
- Languages accepted by NFAs

Regular language closed under many operations:

- union, concatenation, Kleene star via inductive definition or NFAs
- complement, union, intersection via DFAs
- homomorphism, inverse homomorphism, reverse, ...

Different representations allow for flexibility in proofs
Examples: PREFIX and SUFFIX

Let $L$ be a language over $\Sigma$.

**Definition**

$\text{PREFIX}(L) = \{ w \mid wx \in L, x \in \Sigma^* \}$

**Definition**

$\text{SUFFIX}(L) = \{ w \mid xw \in L, x \in \Sigma^* \}$
Examples: PREFIX and SUFFIX

Let \( L \) be a language over \( \Sigma \).

**Definition**

\[
\text{PREFIX}(L) = \{ w \mid wx \in L, x \in \Sigma^* \}
\]

**Definition**

\[
\text{SUFFIX}(L) = \{ w \mid xw \in L, x \in \Sigma^* \}
\]

**Theorem**

*If \( L \) is regular then \( \text{PREFIX}(L) \) is regular.*

**Theorem**

*If \( L \) is regular then \( \text{SUFFIX}(L) \) is regular.*
Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes $L$.

Create new DFA/NFA to accept $\text{PREFIX}(L)$ (or $\text{SUFFIX}(L)$).
Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes $L$.

Create new DFA/NFA to accept $\text{PREFIX}(L)$ (or $\text{SUFFIX}(L)$).

$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$

$Y = \{q \in Q \mid q \text{ can reach some state in } A\}$

$Z = X \cap Y$

**Theorem**

Consider DFA $M' = (Q, \Sigma, \delta, s, Z)$. $L(M') = \text{PREFIX}(L)$. 
Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes $L$.
Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes $L$

$X = \{ q \in Q \mid s \text{ can reach } q \text{ in } M \}$
Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes $L$

$X = \{ q \in Q \mid s \text{ can reach } q \text{ in } M \}$

Consider NFA $N = (Q \cup \{s'\}, \Sigma, \delta', s', A)$. Add new start state $s'$ and $\epsilon$-transition from $s'$ to each state in $X$. 
Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes $L$

$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$

Consider NFA $N = (Q \cup \{s'\}, \Sigma, \delta', s', A)$. Add new start state $s'$ and $\epsilon$-transition from $s'$ to each state in $X$.

Claim: $L(N) = \text{SUFFIX}(L)$.
Part III

DFA to Regular Expressions
Theorem

Given a DFA $M = (Q, \Sigma, \delta, s, A)$ there is a regular expression $r$ such that $L(r) = L(M)$. That is, regular expressions are as powerful as DFAs (and hence also NFAs).

- Simple algorithm but formal proof is involved. See notes.
- An easier proof via a more involved algorithm later in course.
Stage 0: Input
Stage 1: Normalizing

2: Normalizing it.
Stage 2: Remove state A
Stage 4: Redrawn without old edges

```
init  a  B  b
 b  a  C
 C  ε  AC
 a + b
```
Stage 4: Removing B

\[ \text{init} \xrightarrow{a} B \xrightarrow{b} \text{init} \]

\[ \text{init} \xrightarrow{b} C \xrightarrow{a + b} \text{AC} \]

\[ \text{init} \xrightarrow{a} B \xrightarrow{b} \text{init} \]

\[ \text{init} \xrightarrow{b} C \xrightarrow{a + b} \text{AC} \]

\[ \text{ab}^*a \]

\[ \text{init} \xrightarrow{a} B \xrightarrow{b} \text{init} \]

\[ \text{init} \xrightarrow{b} C \xrightarrow{a + b} \text{AC} \]
Stage 5: Redraw

\[ \text{init} \xrightarrow{\text{ab}^*a + b} C \xrightarrow{\epsilon} AC \xrightarrow{a + b} \text{init} \]
Stage 6: Removing C

\begin{align*}
\text{init} & \quad \rightarrow \quad ab^*a + b \\
C & \quad \rightarrow \quad \epsilon \\
a + b & \quad \rightarrow \quad AC \\
\end{align*}

\begin{align*}
\text{init} & \quad \rightarrow \quad ab^*a + b \\
C & \quad \rightarrow \quad \epsilon \\
a + b & \quad \rightarrow \quad AC \\
\end{align*}

\[(ab^*a + b)(a + b)^* \epsilon\]
Stage 7: Redraw

\[(ab^*a + b)(a + b)^*\]
Thus, this automata is equivalent to the regular expression 
\[(ab^*a + b)(a + b)^*\].