CS 154, Lecture 3: DFA ≡ NFA, Regular Expressions
Homework 1 is coming out ...
Deterministic Finite Automata

Computation with finite memory
Non-Deterministic Finite Automata

Computation with finite memory and “verified guessing”
From NFAs to DFAs

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

To learn if an NFA accepts, we could do the computation in parallel, maintaining the set of all possible states that can be reached.

Idea:

Set $Q' = 2^Q$
From NFAs to DFAs: Subset Construction

Input: NFA \( N = (Q, \Sigma, \delta, Q_0, F) \)

Output: DFA \( M = (Q', \Sigma, \delta', q_0', F') \)

\[
\begin{align*}
Q' &= 2^Q \\
\delta' : Q' \times \Sigma &\rightarrow Q' \\
\delta'(R, \sigma) &= \bigcup_{r \in R} \varepsilon(\delta(r, \sigma)) \\
q_0' &= \varepsilon(Q_0) \\
F' &= \{ R \in Q' \mid \exists f \in R \text{ for some } f \in F \}
\end{align*}
\]

For \( S \subseteq Q \), the \( \varepsilon \)-closure of \( S \) is

\[\varepsilon(S) = \{ q \mid q \text{ reachable from some } s \in S \text{ by taking 0 or more } \varepsilon \text{ transitions} \} \]
Example of the $\varepsilon$-closure

$\varepsilon(\{q_0\}) = \{q_0, q_1, q_2\}$

$\varepsilon(\{q_1\}) = \{q_1, q_2\}$

$\varepsilon(\{q_2\}) = \{q_2\}$
Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\} )$

Construct: Equivalent DFA $M = (2^{\{1,2,3\}}, \{a,b\}, \delta', \{1,3\}, ...)$

$\varepsilon(\{1\}) = \{1,3\}$

$\{1\}, \{1,2\} ?$
Reverse Theorem for Regular Languages

Theorem: The reverse of a regular language is also a regular language

If a language can be recognized by a DFA that reads strings from right to left, then there is an “normal” DFA that accepts the same language

Proof?

Given a DFA for a language L, “reverse” its arrows and flip its start and accept states, getting an NFA.
Convert that NFA back to a DFA!
Using NFAs in place of DFAs can make proofs about regular languages much easier!

Remember this on homework/exams!
Union Theorem using NFAs?
Regular Languages are closed under concatenation

**Concatenation:** $A \cdot B = \{vw \mid v \in A \text{ and } w \in B\}$

Given DFAs $M_1$ and $M_2$, connect the accept states of $M_1$ to the start states of $M_2$

$L(N) = L(M_1) \cdot L(M_2)$
Regular Languages are closed under star

\[ A^* = \{ s_1 ... s_k \mid k \geq 0 \text{ and each } s_i \in A \} \]

Let \( M \) be a DFA, and let \( L = L(M) \)

We can construct an NFA \( N \) that recognizes \( L^* \)
Formally, the construction is:

Input: DFA \( M = (Q, \Sigma, \delta, q_1, F) \)

Output: NFA \( N = (Q', \Sigma, \delta', q_0, F') \)

- \( Q' = Q \cup \{q_0\} \)
- \( F' = F \cup \{q_0\} \)

\[
\delta'(q,a) = \begin{cases} 
\delta(q,a) & \text{if } q \in Q \text{ and } a \neq \varepsilon \\
\{q_1\} & \text{if } q \in F \text{ and } a = \varepsilon \\
\{q_1\} & \text{if } q = q_0 \text{ and } a = \varepsilon \\
\emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon \\
\emptyset & \text{else}
\end{cases}
\]
Regular Languages are Closed Under Star

How would we prove that this NFA construction works?

Want to show: \( L(N) = L^* \)

1. \( L(N) \supseteq L^* \)
2. \( L(N) \subseteq L^* \)
1. \( L(N) \supseteq L^* \)

Assume \( w = w_1 \ldots w_k \) is in \( L^* \) where \( w_1, \ldots, w_k \in L \)

We show \( N \) accepts \( w \) by induction on \( k \)

Base Cases:
- \( k = 0 \) (\( w = \varepsilon \))
- \( k = 1 \) (\( w \in L \))

Inductive Step:
Assume \( N \) accepts all strings \( v = v_1 \ldots v_k \in L^* \), \( v_i \in L \)
Let \( u = u_1 \ldots u_k u_{k+1} \in L^* \), \( u_j \in L \)

Since \( N \) accepts \( u_1 \ldots u_k \) (by induction) and \( M \) accepts \( u_{k+1} \), \( N \) also accepts \( u \) (by construction)
Assume $w$ is accepted by $N$; we want to show $w \in L^*$

If $w = \varepsilon$, then $w \in L^*$

I.H. $N$ accepts $u$ and takes at most $k$ $\varepsilon$-transitions

$\Rightarrow u \in L^*$

Let $w$ be accepted by $N$ with $k+1$ $\varepsilon$-transitions.

Write $w$ as $w = uv$, where $v$ is the substring read after the last $\varepsilon$-transition

$u \in L(N)$, so by I.H.

$u \in L^*$

$v \in L$

$w = uv \in L^*$
Closure Properties for Regular Languages

✓ Union: $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$

✓ Intersection: $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$

✓ Complement: $\neg A = \{ w \in \Sigma^* \mid w \notin A \}$

✓ Reverse: $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A, w_i \in \Sigma \}$

✓ Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

✓ Star: $A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \}$

Theorem: if $A$ and $B$ are regular then so are: $A \cup B, A \cap B, \neg A, A^R, A \cdot B,$ and $A^*$
Regular Expressions

Computation as simple, logical description

A totally different way of thinking about computation:
What is the complexity of describing the strings in the language?
Inductive Definition of Regexp

Let $\Sigma$ be an alphabet. We define the regular expressions over $\Sigma$ inductively:

For all $\sigma \in \Sigma$, $\sigma$ is a regexp

$\epsilon$ is a regexp

$\emptyset$ is a regexp

If $R_1$ and $R_2$ are both regexps, then

$(R_1R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regexps
Precedence Order:

* then \cdot then +

Example: \( R_1 \cdot R_2 + R_3 = ((R_1 \cdot R_2) + R_3 \)
Definition: Regexps Represent Languages

The regexp $\sigma \in \Sigma$ represents the language \{\sigma\}

The regexp $\varepsilon$ represents \{\varepsilon\}

The regexp $\emptyset$ represents $\emptyset$

If $R_1$ and $R_2$ are regular expressions representing $L_1$ and $L_2$ then:

$(R_1 R_2)$ represents $L_1 \cdot L_2$

$(R_1 + R_2)$ represents $L_1 \cup L_2$

$(R_1)^*$ represents $L_1^*$
Regexps Represent Languages

For every regexp \( R \), define \( L(R) \) to be the language that \( R \) represents.

A string \( w \in \Sigma^* \) is *accepted by* \( R \) (or, \( w \) *matches* \( R \)) if \( w \in L(R) \).

Examples: 0, 010, and 01010 match \((01)^*0\)
110101110100100 matches \((0+1)^*0\)
Assume $\Sigma = \{0,1\}$

$\{ w \mid w \text{ has exactly a single 1} \}$

$0^*10^*$

$\{ w \mid w \text{ contains 001} \}$

$(0+1)^*001(0+1)^*$
Assume $\Sigma = \{0,1\}$

What language does the regexp $\emptyset^*$ represent?

$\{\varepsilon\}$
Assume $\Sigma = \{0,1\}$

\[
\{ w \mid \text{w has length \(\geq 3\) and its 3rd symbol is 0} \} \\
(0+1)(0+1)0(0+1)^* 
\]
Assume $\Sigma = \{0, 1\}$

\[
\{ \text{w} \mid \text{every odd position in w is a 1} \}
\]

\[
(1(0 + 1))^*(1 + \varepsilon)
\]
Assume $\Sigma = \{0,1\}$

$$\{ w \mid w \text{ has equal number of occurrences of } 01 \text{ and } 10 \}$$

$$= \{ w \mid w = 1, w = 0, \text{ or } w = \varepsilon, \text{ or }$$

$$w \text{ starts with a } 0 \text{ and ends with a } 0, \text{ or }$$

$$w \text{ starts with a } 1 \text{ and ends with a } 1 \}$$

Claim:
A string $w$ has equal occurrences of $01$ and $10$  
$\Leftrightarrow$ $w$ starts and ends with the same bit.

$$1 + 0 + \varepsilon + 0(0+1)^*0 + 1(0+1)^*1$$
DFAs $\equiv$ NFAs $\equiv$ Regular Expressions!

$L$ can be represented by some regexp

$\iff$ $L$ is regular
L can be represented by some regexp

\[\Rightarrow L \text{ is regular}\]
Given any regexp $R$, we will construct an NFA $N$ s.t. $N$ accepts exactly the strings accepted by $R$.

**Proof by induction on the length of the regexp $R$**

**Base Cases** ($R$ has length 1):
- Given any regexp $R$, we will construct an NFA $N$ s.t. $N$ accepts exactly the strings accepted by $R$.
- $R = \sigma$
- $R = \varepsilon$
- $R = \emptyset$

**Proof by induction on the length of the regexp $R**
Induction Step: Suppose every regexp of length \(< k\) represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

\[ R = R_1 + R_2 \]
\[ R = R_1 R_2 \]
\[ R = (R_1)^* \]
Induction Step: Suppose every regexp of length \( k \) represents some regular language.

Consider a regexp \( R \) of length \( k > 1 \)

Three possibilities for \( R \):

\[
R = R_1 + R_2 \quad \text{By induction, } R_1 \text{ and } R_2 \text{ represent some regular languages, } L_1 \text{ and } L_2
\]

\[
R = R_1 R_2 \quad \text{But } L(R) = L(R_1 + R_2) = L_1 \cup L_2
\]

\[
R = (R_1)^* \quad \text{so } L(R) \text{ is regular, by the union theorem!}
\]
Induction Step: Suppose every regexp of length \(< k\) represents some regular language.

Consider a regexp R of length \(k > 1\)

Three possibilities for R:

\[
\begin{align*}
R &= R_1 + R_2 & \text{By induction, } R_1 \text{ and } R_2 \text{ represent some regular languages, } L_1 \text{ and } L_2 \\
R &= R_1 R_2 & \text{But } L(R) = L(R_1 R_2) = L_1 \cdot L_2 \\
R &= (R_1)^* & \text{so } L(R) \text{ is regular by the concatenation theorem}
\end{align*}
\]
Induction Step: Suppose every regexp of length < \( k \) represents some regular language.

Consider a regexp \( R \) of length \( k > 1 \)

Three possibilities for \( R \):

\[
R = R_1 + R_2 \quad \text{By induction, } R_1 \text{ and } R_2 \text{ represent some regular languages, } L_1 \text{ and } L_2
\]

\[
R = R_1 R_2 \quad \text{But } L(R) = L(R_1^*) = L_1^*
\]

\[
R = (R_1)^* \quad \text{so } L(R) \text{ is regular, by the star theorem}
\]
Induction Step: Suppose every regexp of length < $k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

1. $R = R_1 + R_2$  
   By induction, $R_1$ and $R_2$ represent some regular languages, $L_1$ and $L_2$

2. $R = R_1 R_2$  
   But $L(R) = L(R_1^*) = L_1^*$  
   so $L(R)$ is regular, by the star theorem

3. $R = (R_1)^*$

Therefore: If $L$ is represented by a regexp, then $L$ is regular
Give an NFA that accepts the language represented by \((1(0+1))^*\)

Regular expression: \((1(0+1))^*\)
Generalized NFAs (GNFA)

L can be represented by a regexp

\[ \iff L \text{ is a regular language} \]

Idea: Transform an NFA for L into a regular expression by removing states and re-labeling the arcs with regular expressions

Rather than reading in just letters from the string on a step, we can read in entire substrings
This GNFA recognizes $L(a^*b(cb)^*a)$

Is $aaabcbbcba$ accepted or rejected?
Is $bba$ accepted or rejected?
Is $bcba$ accepted or rejected?

This GNFA recognizes $L(a^*b(cb)^*a)$
Add unique start and accept states
While the machine has more than 2 states:

Pick an internal state, rip it out and re-label the arrows with regexps, to account for paths through the missing state.
In general:

\[ R(q_1, q_2) R(q_2, q_2)^* R(q_2, q_3)^* R(q_1, q_3) \]

While the machine has more than 2 states:

In general:
\[ R(q_0, q_3) = (a \cdot b)(a + b)^* \] represents \( L(N) \)
DFAs ↔ NFAs

Regular Languages

DEFINITION

Regular Expressions
Parting thoughts:
Regular Languages can be defined by their closure properties
NFA=DFA, does it mean that non-determinism is free for Finite Automata?

Questions?