CS 420
Nondeterminism
Wednesday, February 8, 2023
UMass Boston Computer Science
Announcements

• HW 1 in
  • due Tues 2/7 11:59pm EST

• HW 2 out
  • due Tues 2/14 11:59pm EST
Last Time: Is Union Closed For Regular Langs?

*In this course, we are interested in closed operations for a set of languages (here the set of regular languages)*

(In general, a set is **closed** under an operation if applying the **operation** to members of the set produces a result in the same set)

The class of regular languages is **closed** under the union operation.

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$. 
Last Time: Is Union Closed For Regular Langs?

The class of regular languages is **closed** under the **union operation**.

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

(In general, a set is **closed** under an operation if applying the **operation** to members of the set produces a result in the same set.)

A member of the set of regular languages is ...

... a regular language, which itself is a set (of strings) ...

... so the operations we’re interested in are **set operations**

Want to prove this statement

Or this (same) statement
Last Time: Is Union Closed For Regular Langs?

**Statements**

1. $A_1$ and $A_2$ are regular languages
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes $A_1$
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes $A_2$
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ (todo)
5. $M$ recognizes $A_1 \cup A_2$ (How to create this? Don’t know what $A_1$ and $A_2$ are!)
6. $A_1 \cup A_2$ is a regular language
7. The class of regular languages is closed under the union operation.

**Justifications**

1. Assumption
2. Def of Regular Language
3. Def of Regular Language
4. Def of DFA
5. See examples
6. Def of Regular Language
7. From stmt #1 and #6

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$. 
A language is called a **regular language** if some finite automaton recognizes it.

**Definition**

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the **states**,
2. $\Sigma$ is a finite set called the **alphabet**,
3. $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**,  
4. $q_0 \in Q$ is the **start state**, and 
5. $F \subseteq Q$ is the **set of accept states**.

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize $A_1$, 
$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize $A_2$, 

If we don’t know what exactly these languages are, we still know these facts...
**Want:** $M$

$M_1$ recognizes $A_1$

$M_2$ recognizes $A_2$

**Rough sketch Idea:**

$M$ is a combination of $M_1$ and $M_2$ that checks whether its input is accepted by either $M_1$ and $M_2$.

But, a DFA can only read its input once!

Need to somehow simulate “being in” both an $M_1$ and $M_2$ state simultaneously.

---

**THEOREM**

The class of regular languages is closed under the union operation.

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$. 

\[
A_1 \cup A_2
\]
Last Time: Union is Closed For Regular Langs

Proof
• Given: $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, recognize $A_1$,
  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognize $A_2$,
• Construct: $M = (Q, \Sigma, \delta, q_0, F)$, using $M_1$ and $M_2$, that recognizes $A_1 \cup A_2$
• states of $M$:

  $Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$

  This set is the Cartesian product of sets $Q_1$ and $Q_2$

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the states,
2. $\Sigma$ is a finite set called the alphabet,
3. $\delta: Q \times \Sigma \rightarrow Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

A state of $M$ is a pair:
- the first part is a state of $M_1$ and
- the second part is a state of $M_2$

So the states of $M$ is all possible combinations of the states of $M_1$ and $M_2$
**Last Time:** Union is Closed For Regular Langs

**Proof**

- Given: 
  \[ M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \text{ recognize } A_1, \]
  \[ M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2), \text{ recognize } A_2, \]
- Construct: \( M = (Q, \Sigma, \delta, q_0, F), \text{ using } M_1 \text{ and } M_2, \text{ that recognizes } A_1 \cup A_2 \)
- states of \( M \): 
  \[ Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2 \]

This set is the **Cartesian product** of sets \( Q_1 \) and \( Q_2 \)

---

A **finite automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

\[
\alpha = (\delta_1(r_1, a), \delta_2(r_2, a))
\]

A step in \( M \) includes both:
- a step in \( M_1 \), and
- a step in \( M_2 \)
Last Time: Union is Closed For Regular Langs

Proof

• Given: \( M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \), recognize \( A_1 \),
  \( M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \), recognize \( A_2 \),

• Construct: \( M = (Q, \Sigma, \delta, q_0, F) \), using \( M_1 \) and \( M_2 \), that recognizes \( A_1 \cup A_2 \)

• states of \( M \): \( Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2 \)
This set is the Cartesian product of sets \( Q_1 \) and \( Q_2 \)

• \( M \) transition fn: \( \delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \)

• \( M \) start state: \( (q_1, q_2) \)  
  \text{Start state of } M \text{ is both start states of } M_1 \text{ and } M_2
**Last Time:** Union is Closed For Regular Langs

**Proof**

- **Given:**
  \[
  M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \text{ recognize } A_1, \\
  M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2), \text{ recognize } A_2,
  \]

- **Construct:** \( M = (Q, \Sigma, \delta, q_0, F) \), using \( M_1 \) and \( M_2 \), that recognizes \( A_1 \cup A_2 \)

- **states of** \( M \):
  \[
  Q = \{ (r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2 \} = Q_1 \times Q_2
  \]
  This set is the *Cartesian product* of sets \( Q_1 \) and \( Q_2 \)

- **\( M \) transition fn:**
  \[
  \delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))
  \]

- **\( M \) start state:**
  \[
  (q_1, q_2)
  \]

- **\( M \) accept states:**
  \[
  F = \{ (r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2 \}
  \]

*Remember: Accept states must be subset of \( Q \)*

(Q.E.D.)
Another operation: Concatenation

Example: Recognizing street addresses

212 Beacon Street

$M_3$: CONCAT

$M_1$: recognize numbers

$M_2$: recognize words

We want this operation to be closed ... allows using DFAs as building blocks (~ modular programming)
Concatenation of Languages

Let the alphabet $\Sigma$ be the standard 26 letters $\{a, b, \ldots, z\}$.
If $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$, then

$$A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$$
Is Concatenation Closed?

**Theorem**

The class of regular languages is closed under the concatenation operation.

In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

- Construct a **new** machine $M$ recognizing $A_1 \circ A_2$? (like union)
  - Using DFA $M_1$ (which recognizes $A_1$),
  - and DFA $M_2$ (which recognizes $A_2$)
Let $M_1$ recognize $A_1$, and $M_2$ recognize $A_2$.

**Want:** Construction of $M$ to recognize $A_1 \circ A_2$.
Overlapping Concatenation Example

• Let $M_1$ recognize language $A = \{\text{j}e\text{n}, \text{j}e\text{n}s\}$
• and $M_2$ recognize language $B = \{\text{smith}\}$
• Want: Construct $M$ to recognize $A \circ B = \{\text{jen}\text{smith}, \text{jen}ss\text{smith}\}$

• If $M$ sees jen ...
• $M$ must decide to either:
Overlapping Concatenation Example

Let $M_1$ recognize language $A = \{\text{jens}, \text{jens}\}$

and $M_2$ recognize language $B = \{\text{smith}\}$

Want: Construct $M$ to recognize $A \circ B = \{\text{jenssmith}, \text{jenssmith}\}$

If $M$ sees $\text{jens}$ ...

$M$ must decide to either:
  - stay in $M_1$ (correct, if full input is $\text{jenssmith}$)
Overlapping Concatenation Example

- Let $M_1$ recognize language $A = \{ \text{jen}, \text{jens} \}$
- and $M_2$ recognize language $B = \{ \text{smith} \}$
- Want: Construct $M$ to recognize $A \circ B = \{ \text{jensmith}, \text{jenssmith} \}$

- If $M$ sees jen ...
  - $M$ must decide to either:
    - stay in $M_1$ (correct, if full input is jenssmith)
    - or switch to $M_2$ (correct, if full input is jenssmith)

- But to recognize $A \circ B$, it needs to handle both cases!!
  - Without backtracking

**Concatenation:** $A \circ B = \{ xy \mid x \in A \text{ and } y \in B \}$
The class of regular languages is closed under the concatenation operation.

In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

- Cannot combine $A_1$ and $A_2$’s machine because:
  - Not clear when to switch machines? (can only read input once)
- Requires a new kind of machine!
- But does this mean concatenation is not closed for regular langs?
Deterministic vs Nondeterministic

Deterministic computation

- start
- ...
- accept or reject

DFAs
Deterministic vs Nondeterministic

Deterministic computation

- start
- ... (transitions)
- accept or reject

Nondeterministic computation

- states
- reject
- ... (transitions)

New FA

DFA

Nondeterministic computation can be in multiple states at the same time
Finite Automata: The Formal Definition

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the states,
2. \(\Sigma\) is a finite set called the alphabet,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

Also called a Deterministic Finite Automata (DFA)
Precise Terminology is Important

• A finite automata or finite state machine (FSM) defines ...
  ... computation with a finite number of states

• There are many kinds of FSMs

• We’ve learned one kind, the Deterministic Finite Automata (DFA)
  • (So up to now, the terms DFA and FSM refer to the same definition)

• But now we learn other kinds, e.g., Nondeterministic Finite Automata (NFA)

• Be careful with terminology!
A **nondeterministic finite automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set of states,
2. \(\Sigma\) is a finite alphabet,
3. \(\delta: Q \times \Sigma \epsilon \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

**Power set, i.e. a transition results in set of states**

**Compare with DFA:**

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called the **states**,
2. \(\Sigma\) is a finite set called the **alphabet**,
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the **transition function**, 
4. \(q_0 \in Q\) is the **start state**, and
5. \(F \subseteq Q\) is the **set of accept states**.
Power Sets

• A **power set** is the set of all subsets of a set

• **Example**: \( S = \{a, b, c\} \)

• Power set of \( S = \)
  • \( \{ \{ \}, \{ a \}, \{ b \}, \{ c \}, \{ a, b \}, \{ a, c \}, \{ b, c \}, \{ a, b, c \} \} \)
  • **Note**: includes the empty set!
**DEFINITION**

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set of states,
2. \(\Sigma\) is a finite alphabet,
3. \(\delta: Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

Transition label can be “empty”, i.e., machine can transition without reading input.

\[\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}\]
NFA Example

• Come up with a formal description of the following NFA:

**DEFINITION**

A *nondeterministic finite automaton* is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set of states,
2. \(\Sigma\) is a finite alphabet,
3. \(\delta: Q \times \Sigma_e \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.
The formal description of $N_1$ is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0, 1\}$,
3. $\delta$ is given as

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\emptyset$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_4}$</td>
<td>${q_4}$</td>
<td>$\emptyset$,</td>
</tr>
</tbody>
</table>

4. $q_1$ is the start state, and
5. $F = \{q_4\}$. 

$\delta: Q \times \Sigma \varepsilon \rightarrow \mathcal{P}(Q)$

- Empty transition (no input read)
- Result of transition is a set
- Empty transition (no input read)
- Multiple 1 transitions
- No 0 transition
In-class Exercise

• Come up with a formal description for the following NFA
  • \( \Sigma = \{ a, b \} \)

**Definition**

A **nondeterministic finite automaton** is a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \), where

1. \( Q \) is a finite set of states,
2. \( \Sigma \) is a finite alphabet,
3. \( \delta: Q \times \Sigma_e \rightarrow \mathcal{P}(Q) \) is the transition function,
4. \( q_0 \in Q \) is the start state, and
5. \( F \subseteq Q \) is the set of accept states.
In-class Exercise Solution

Let \( N = (Q, \Sigma, \delta, q_0, F) \)

- \( Q = \{ q_1, q_2, q_3 \} \)
- \( \Sigma = \{ a, b \} \)
- \( \delta \)
- \( q_0 = q_1 \)
- \( F = \{ q_1 \} \)

\[
\begin{align*}
\delta(q_1, a) &= \{ \} \\
\delta(q_1, b) &= \{ q_2 \} \\
\delta(q_1, \varepsilon) &= \{ q_3 \} \\
\delta(q_2, a) &= \{ q_2, q_3 \} \\
\delta(q_2, b) &= \{ q_3 \} \\
\delta(q_2, \varepsilon) &= \{ \} \\
\delta(q_3, a) &= \{ q_1 \} \\
\delta(q_3, b) &= \{ \} \\
\delta(q_3, \varepsilon) &= \{ \} 
\end{align*}
\]
NFA Computation (JFLAP demo): 010110
NFA Computation Sequence

Symbol read

0

1

0

1

0

NFA accepts input if at least one path ends in accept state

Each step can branch into multiple states at the same time!

So this is an accepting computation
Flashback: DFA Computation Model

Informally

- **Machine** = a DFA
- **Input** = string of chars, e.g. “1101”

Machine “accepts” input if:
- **Start** in “start state”
- **Repeat:**
  - Read 1 char;
  - Change state according to the transition table
- **Result** =
  - Last state is “Accept” state

Formally (i.e., mathematically)

- \( M = (Q, \Sigma, \delta, q_0, F) \)
- \( w = w_1 w_2 \cdots w_n \)

\( M \) accepts \( w \) if
  - sequence of states \( r_0, r_1, \ldots, r_n \) in \( Q \) exists with
    - \( r_0 = q_0 \)
    - \( r_i = \delta(r_{i-1}, w_i) \), for \( i = 1, \ldots, n \)
  - \( r_n \in F \)
Informally

- Machine = a DFA-an NFA
- Input = string of chars, e.g. “1101”

Machine “accepts” input if:
- Start in “start state”

- Repeat:
  - Read 1 char;
  - Change state according to the transition table

- Result =
  - Last state is “Accept” state

Formally (i.e., mathematically)

- \( M = (Q, \Sigma, \delta, q_0, F) \)
- \( w = w_1w_2 \cdots w_n \)

\( M \) accepts \( w \) if
- sequence of states \( r_0, r_1, \ldots, r_n \) in \( Q \) exists with
  - \( r_0 = q_0 \)
  - \( r_i = \delta(r_{i-1}, w_i) \), for \( i = 1, \ldots, n \)
  - \( r_i \in \delta(r_{i-1}, w_i) \) This is now a set
  - \( r_n \in F \)
Flashback: DFA Extended Transition Function

Define extended transition function: 
\[ \hat{\delta} : Q \times \Sigma^* \rightarrow Q \]

**Domain:**
- Beginning state \( q \in Q \) (not necessarily the start state)
- Input string \( w = w_1 w_2 \cdots w_n \) where \( w_i \in \Sigma \)

**Range:**
- Ending state (not necessarily an accept state)

(Defined recursively, on length of input string)

- **Base case:** \( \hat{\delta}(q, \varepsilon) = q \)
- **Recursive case:** \( \hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n) \)
Alternate Extended Transition Function

Define **extended transition function**: \( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \)

- **Domain**:
  - Beginning state \( q \in Q \) (not necessarily the start state)
  - Input string \( w = w_1 w_2 \cdots w_n \) where \( w_i \in \Sigma \)

- **Range**:
  - Ending state (not necessarily an accept state)

(Defined recursively, on length of input string)

- **Base case**: \( \hat{\delta}(q, \varepsilon) = q \)
- **Recursive case**: \( \hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n) \)

\[ \delta(\delta(q, w_1 \cdots w_{n-1}), w_n) \]

- **Empty string**
- **NonEmpty string**
- **Recursive call**: (smaller argument) computation “so far”
- **Single transition step**: on last char
Define **extended transition function**:  \( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \)

- **Domain:**
  - Beginning state \( q \in Q \)
  - Input string \( w = w_1 w_2 \cdots w_n \) where \( w_i \in \Sigma \)

- **Range:**
  - Ending state set of states

**Result is set of states**
Extended Transition Function

Define **extended transition function**: \( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \)

**Domain:**
- Beginning state \( q \in Q \)
- Input string \( w = w_1w_2\cdots w_n \) where \( w_i \in \Sigma \)

**Range:**
- Ending state set of states

(Defined recursively, on length of input string)

- **Base case:** \( \hat{\delta}(q, \epsilon) = \{q\} \)

- **Recursive case:**
Extended Transition Function

Define **extended transition function**: \( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \)

**Domain:**
- Beginning state \( q \in Q \)
- Input string \( w = w_1w_2 \cdots w_n \) where \( w_i \in \Sigma \)

**Range:**
- Ending state set of states

(Defined recursively, on length of input string)

- **Base case:** \( \hat{\delta}(q, \epsilon) = \{q\} \)
- **Recursive case:** \( \hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n) \) where: \( \hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\} \)

Result is set of states

Empty string

All single transition steps for last char

Recursive call: (smaller argument) computation “so far”

nonEmpty string
NFA Extended $\delta$ Example

Base case:
$$\hat{\delta}(q, \epsilon) = \{q\}$$

Recursive case:
$$\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$$
where:
$$\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$$

- $\hat{\delta}(q_0, \epsilon) = \{q_0\}$ (Stay in start state)
- $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$ (Same as single step $\delta$)
- $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

We haven’t considered empty transitions!
Adding Empty Transitions

• Define the set $\varepsilon$-REACHABLE$(q)$
  • ... to be all states reachable from $q$ via zero or more empty transitions

(Defined recursively)

• **Base case:** $q \in \varepsilon$-REACHABLE$(q)$

• **Inductive case:**
  \[
  \varepsilon$-REACHABLE$(q) = \{ r | p \in \varepsilon$-REACHABLE$(q)$ and $r \in \delta(p, \varepsilon) \}\]

A state is in the reachable set if...

... there is an empty transition to it from another state in the reachable set
**ε-REACHABLE Example**

\[
\varepsilon\text{-REACHABLE}(1) = \{1, 2, 3, 4, 6\}
\]
NFA Extended Transition Function

Define **extended transition function**: \( \hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \)

**Domain:**
- Beginning state \( q \in Q \)
- Input string \( w = w_1w_2 \cdots w_n \) where \( w_i \in \Sigma \)

**Range:**
- Ending set of states

(Defined recursively, on length of input string)

- **Base case:** \( \hat{\delta}(q, \varepsilon) = \{q\} \)

- **Recursive case:** \( \hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n) \)

where: \( \hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\} \)
NFA Extended Transition Function

Define **extended transition function**: \( \hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \)

**Domain:**
- Beginning state \( q \in Q \)
- Input string \( w = w_1 w_2 \cdots w_n \) where \( w_i \in \Sigma \)

**Range:**
- Ending set of states

(Defined recursively, on length of input string)

- **Base case:** \( \hat{\delta}(q, \epsilon) = \{q\} \)

- **Recursive case:** \( \hat{\delta}(q, w) = \bigcup_{i=1}^{k} \hat{\delta}(q_i, w_n) \)
  
  where: \( \hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\} \)

"Take single step, then follow all empty transitions"
Summary: NFAs vs DFAs

**DFAs**
- Can only be in **one** state
- Transition:
  - Must read 1 char
- Acceptance:
  - If final state is **accept** state

**NFAs**
- Can be in **multiple** states
- Transition
  - Can read no chars
  - i.e., empty transition
- Acceptance:
  - If **one of final** states is accept state
Last Time: Concatenation of Languages

Let the alphabet $\Sigma$ be the standard 26 letters $\{a, b, \ldots, z\}$. If $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$, then

$$A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$$
Last Time: Concatenation is Closed?

**Theorem**

The class of regular languages is closed under the concatenation operation.

In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

**Proof:** Construct a new machine

- How does it know when to switch machines?
  - Can only read input once
Let $N_1$ recognize $A_1$, and $N_2$ recognize $A_2$.

Want: Construction of $N$ to recognize $A_1 \circ A_2$.

$\varepsilon$ = “empty transition” = reads no input
Allows $N$ to be in both machines at once

$N$ is an NFA! It simultaneously:
- Keeps checking 1st part with $N_1$ and
- Moves to $N_2$ to check 2nd part
Flashback: Is Union Closed For Regular Langs?

Statements
1. $A_1$ and $A_2$ are regular languages
2. A DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes $A_1$
3. A DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes $A_2$
4. Construct DFA $M = (Q, \Sigma, \delta, q_0, F)$
5. $M$ recognizes $A_1 \cup A_2$
6. $A_1 \cup A_2$ is a regular language
7. The class of regular languages is closed under the union operation.

Justifications
1. Assumption
2. Def of Regular Language
3. Def of Regular Language
4. Def of DFA
5. See examples
6. Def of Regular Language
7. From stmt #1 and #6

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$. 
Is Concat Closed For Regular Langs?

**Statements**
1. $A_1$ and $A_2$ are regular languages
2. A NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes $A_1$
3. A NFA $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes $A_2$
4. Construct NFA $N = ???$ (todo)
5. $M$ recognizes $A_1 \cup A_2 A_1 \circ A_2$
6. $A_1 \circ A_2 A_1 \cup A_2$ is a regular language
7. The class of regular languages is closed under the concatenation operation.

**Justifications**
1. Assumption
2. Def of Regular Language
3. Def of Regular Language
4. Def of NFA
5. See examples
6. Def of Regular Language
7. From stmt #1 and #6

In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$. 
Concatenation is Closed for Regular Langs

**Proof**

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $N = (Q, \Sigma, \delta, q_1, F)$ to recognize $A_1 \circ A_2$
1. $Q = Q_1 \cup Q_2$
2. The state $q_1$ is the same as the start state of $N_1$
3. The accept states $F_2$ are the same as the accept states of $N_2$
4. Define $\delta$ so that for any $q \in Q$ and any $a \in \Sigma \varepsilon$, 

![Diagrams showing the construction of $N$ from $N_1$ and $N_2$.]
Concatenation is Closed for Regular Langs

**Proof**

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$
2. The state $q_1$ is the same as the start state of $N_1$
3. The accept states $F_2$ are the same as the accept states of $N_2$
4. Define $\delta$ so that for any $q \in Q$ and any $a \in \Sigma_e$,

$$
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \not\in F_1 \\
\delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\
\delta_2(q, a) & q \in Q_2.
\end{cases}
$$
Flashback: A DFA’s Language

- For DFA $M = (Q, \Sigma, \delta, q_0, F)$

- $M$ accepts $w$ if $\hat{\delta}(q_0, w) \in F$

- $M$ recognizes language $A$ if $A = \{w | M$ accepts $w\}$

- A language is a regular language if a DFA recognizes it
An NFA’s Language

• For NFA  $N = (Q, \Sigma, \delta, q_0, F)$

• $N$ accepts $w$ if $\hat{\delta}(q_0, w) \cap F \neq \emptyset$
  • i.e., if the final states have at least one accept state

• Language of $N = L(N) = \left\{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$

Q: How does an NFA’s language relate to regular languages
• Definition: A language is regular if a DFA recognizes it
Is Concatenation Closed for Reg Langs?

• Concatenation of DFAs produces an NFA

To finish the proof ...
• we must prove that NFAs also recognize regular languages.

Specifically, we must prove:
• NFAs $\Leftrightarrow$ regular languages
Check-in Quiz 2/8

On gradescope