Nondeterminism
Thursday, September 22, 2022
UMass Boston Computer Science
Announcements

• HW 1
  • due Sun Sept 25 11:59pm EST
Last Time: Concatenation

Example: Recognizing street addresses

212 Beacon Street

We want this operation to be closed...
allows using DFAs as building blocks
(~ modular programming)
Last Time: Is Concatenation Closed?

FALSE?

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 $A_1 \circ A_2$.

• 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?
Last Time: Deterministic vs Nondeterministic

Deterministic computation

Nondeterministic computation

Non-deterministic computation can be in multiple states at the same time

DFAs

New FA
**Last Time:** Formal Definition of NFAs

**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. \(\delta : Q \times \Sigma \varepsilon \rightarrow P(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

**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**.

*NFA transition may not read input, \(\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}\)***

*Transition results in a set of states***

167
Last Time: NFA Example

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

![NFA Diagram]

**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. \(\delta: Q \times \Sigma_e \longrightarrow \mathcal{P}(Q)\) 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. $F = \{q_4\}$. 

\[\delta: Q \times \Sigma \varepsilon \rightarrow \mathcal{P}(Q)\]
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. “1011”

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. “101”

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
- \( r_0 = q_0 \)
- \( r_i = \delta(r_{i-1}, w_i), \text{ for } i = 1, \ldots, n \)
- \( r_n \in F \)

A non-deterministic 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. \( \delta : Q \times \Sigma \rightarrow P(Q) \) is the transition function,
4. \( q_0 \in Q \) is the start state, and
5. \( F \subseteq Q \) is the set of accept states.
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 \( w = w_1w_2 \cdots w_n \) 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:** \( \hat{\delta}(q, w) = \hat{\delta}(\hat{\delta}(q, w_1), w_2 \cdots w_n) \)
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 \( w = w_1w_2 \cdots w_n \) 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:** \( \hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n) \)

\( \delta : Q \times \Sigma \rightarrow Q \) is the **transition function**

- Empty string
- Recursive call: (smaller argument) computation “so far”
- Non-empty string
- 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
Define **extended transition function:**

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

\[ \hat{\delta} : Q \times \Sigma^* \rightarrow Q \]
\[ \hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \]

Result is set of states

Empty string

Non-empty string

All single transition steps for last char

Recursive call: (smaller argument) computation “so far”
NFA Extended $\delta$ Example

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

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

1. $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$ Stay in start state
2. $\hat{\delta}(q_0, 0) = \hat{\delta}(q_0, 0) = \{q_0, q_1\}$ Same as single step $\delta$
3. $\hat{\delta}(q_0, 00) = \hat{\delta}(q_0, 0) \cup \hat{\delta}(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$ Combine result of recursive call with “last step”
4. $\hat{\delta}(q_0, 001) = \hat{\delta}(q_0, 1) \cup \hat{\delta}(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

We haven’t considered empty transitions!
Adding Empty Transitions

- Define **the set** \( \varepsilon\text{-REACHABLE}(q) \)
  - ... to be all states reachable from \( q \) via zero or more empty transitions

(Defined recursively)

- **Base case:** \( q \in \varepsilon\text{-REACHABLE}(q) \)

- **Inductive case:**

\[
\varepsilon\text{-REACHABLE}(q) = \{ r \mid p \in \varepsilon\text{-REACHABLE}(q) \text{ and } r \in \delta(p, \varepsilon) \}
\]

A state is in the reachable set if ...

... there is an empty transition to it from another state in the reachable set
$\varepsilon$-REACHABLE Example

$\varepsilon$-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 \( w = w_1 w_2 \cdots w_n \) where \( w_i \in \Sigma \)

**Range:**
- Ending set of states

(Defined recursively, on length of input string)

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

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

**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:** \( \hat{\delta}(q, \epsilon) = \{q\} \)
  
  \[
  \varepsilon\text{-REACHABLE}(q) = \\bigcup_{i=1}^{k} \hat{\delta}(q_i, w_n)
  \]

- **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

<table>
<thead>
<tr>
<th>DFAs</th>
<th>NFAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Can only be in <strong>one</strong> state</td>
<td>• Can be in <strong>multiple</strong> states</td>
</tr>
<tr>
<td>• Transition:</td>
<td>• Transition</td>
</tr>
<tr>
<td>• Must read 1 char</td>
<td>• Can read no chars</td>
</tr>
<tr>
<td></td>
<td>• <strong>i.e.</strong>, empty transition</td>
</tr>
<tr>
<td>• Acceptance:</td>
<td>• Acceptance:</td>
</tr>
<tr>
<td>• If final state is accept state</td>
<td>• If <strong>one of</strong> final states is accept state</td>
</tr>
</tbody>
</table>
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

$$A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$$
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 $N_1$
- Moves to $N_2$ to check 2nd part
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 $a \in \Sigma_\varepsilon$, 

![Diagram](image-url)
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 $a \in \Sigma$,

$$\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\
\delta_2(q, a) & q \in Q_2.
\end{cases}$$
Flashback: A DFA’s Language

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

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

- $M$ 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 9/22

On gradescope