CS420
Combining Automata & Closed Operations
Tuesday, September 20, 2022
UMass Boston Computer Science
Announcements

• HW 1
  • Due Sun 9/25 11:59pm EST
  • Get started early!
  • Questions asked late on Sunday are less likely to be answered

• HW 0 grades returned
  • Use gradecope re-grade request for questions and/or complaints
Last Time: Tips on How to Create Finite Automata

Analogies for this class:
- Automata \sim Programs :: Designing Automata \sim Programming!

1. **Confirm understanding** of the problem
   - Create examples ... and expected results (accept / reject)

2. Decide **information to “remember”**
   - These are the machine states: some are accept states; one is start state

3. Determine **transitions** between states

4. **Test** machine behaves as expected
   - Use your examples; create additional ones if needed
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).

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.
Last Time: Union 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

$$A \cup B = \{\text{good, bad, boy, girl}\}$$
Last Time: Is Union Closed For Regular Langs?

**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 $A_1 \cup A_2$.

- **How do we prove that a language is regular?**
  - Create a DFA recognizing it!

- **So to prove this theorem ... create** a DFA that recognizes $A_1 \cup A_2$
  - But! We **don’t know** what $A_1$ and $A_2$ are!
  - What **do** we know about $A_1$ and $A_2$??

A language is called a **regular language** if some finite automaton recognizes it.
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**.

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

$M_2$ recognizes $A_2$

Want: $M$

Recognizes $A_1 \cup A_2$

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 $A_1 \cup A_2$. 
Union is Closed For Regular Languages

**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 \( Q_2 \)

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**,\(^1\)
4. \( q_0 \in Q \) is the **start state**, and
5. \( F \subseteq Q \) is the set of accept states.

**Want:** \( M \) that can simultaneously be in both an \( M_1 \) and \( M_2 \) state

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 \( M_2 \)
Union is Closed For Regular Languages

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 \( Q_2 \)

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. \( \delta: Q \times \Sigma \to Q \) 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 step in \( M \) includes both:
- a step in \( M_1 \), and
- a step in \( M_2 \)
Union is Closed For Regular Languages

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 \( M \):
  \[ Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2 \]

  This set is the \textit{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:
  \[ (q_1, q_2) \]

  Start state of \( M \) is both start states of \( M_1 \) and \( M_2 \)
Union is Closed For Regular Languages

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 $Q_2$

• $M$ 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: $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Remember: Accept states must be subset of $Q$

Accept if either $M_1$ or $M_2$ accept

(Q.E.D.)

132
Another operation: Concatenation

Example: Recognizing street addresses

212 Beacon Street

$M_3$: CONCAT

$M_1$: recognize numbers

$M_2$: recognize words
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 $A_1 \circ A_2$.

**Problem:** Can only read input once, can’t backtrack.

Need to switch machines at some point, but when?
Overlapping Concatenation Example

- Let $M_1$ recognize language $A = \{ \text{j}en, \text{j}ens \}$
- and $M_2$ recognize language $B = \{ \text{s}mith \}$
- Want: Construct $M$ to recognize $A \circ B = \{ \text{j}ens\text{s}mith, \text{j}ens\text{s}smith \}$

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

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

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

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

- If $M$ sees $jen$ ...
- $M$ must decide to either:
  - stay in $M_1$ (correct, if full input is $jens\text{smith}$)
  - or switch to $M_2$ (correct, if full input is $jens\text{smith}$)

- But to recognize $A \circ B$, it needs to handle both cases!!
  - Without backtracking
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:
  - Need to switch from $A_1$ to $A_2$ at some point ...
  - ... but we don’t know when! (we can only read input once)

- This requires a **new kind of machine**!

- But does this mean concatenation is not **closed** for regular langs?
Nondeterminism
Deterministic vs Nondeterministic

Deterministic computation

- start

- states

- ...

- accept or reject

DFAs
Deterministic vs Nondeterministic

Deterministic computation

- start
- ... states
- accept or reject

DFAs

Nondeterministic computation

- reject
- accept

New FA

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

**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 currently, the terms DFA and FSM refer to the same definition)

• We will learn **other kinds**, e.g., **Nondeterministic Finite Automata (NFA)**

• **Be careful with terminology!**
Nondeterministic Finite Automata (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. \(\delta: Q \times \Sigma \varepsilon \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

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*.

Difference

Power set, i.e. a transition results in *set of 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!
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. \(\delta: Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

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

Transition label can be “empty”, i.e., machine can transition without reading input.
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. \(\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)\) 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. $F = \{q_4\}$. 

\[\delta : Q \times \Sigma \varepsilon \rightarrow \mathcal{P}(Q)\]
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. $\delta: Q \times \Sigma_e \rightarrow P(Q)$ 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 \( N = (Q, \Sigma, \delta, q_0, F) \)

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

\[
\begin{align*}
\delta(q_1, a) &= \{ \} \\
\delta(q_1, b) &= \{ q_2 \} \\
\delta(q_1, \varepsilon) &= \{ q_3 \} \\
\delta(q_2, a) &= \{ q_2, q_3 \} \\
\delta(q_2, b) &= \{ q_3 \} \\
\delta(q_2, \varepsilon) &= \{ \} \\
\delta(q_3, a) &= \{ q_1 \} \\
\delta(q_3, b) &= \{ \} \\
\delta(q_3, \varepsilon) &= \{ \}
\end{align*}
\]
Next Time: Running Programs, NFAs (JFLAP demo): 010110
Check-in Quiz 9/20

On gradescope