CS420
Combining Automata & Regular Languages
Monday, January 31, 2022
UMass Boston Computer Science
Announcements

• HW 0 in

• HW 1 out
  • Due Sun 2/6 11:59pm
Last Time: Alphabets, Strings, Languages

• An alphabet is a non-empty finite set of symbols
  \[ \Sigma_1 = \{0,1\} \]
  \[ \Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\} \]

• A string is a finite sequence of symbols from an alphabet
  \[ 01001 \quad \text{abracadabra} \quad \varepsilon \]
  Empty string (length 0)

• A language is a set of strings
  \[ A = \{\text{good, bad}\} \]
  \[ \emptyset \quad \{\} \]
  Empty set is a language
  \[ A = \{w \mid w \text{ contains at least one 1 and an even number of 0s, follow the last 1}\} \]
  Languages can be infinite
  “the set of all ...”
  “such that ...”
Last Time: Computers and Languages

- The **language of a machine** is the set of all strings that it accepts.

E.g.,
- An DFA $M = (Q, \Sigma, \delta, q_0, F)$ **accepts** string $w$ if $\hat{\delta}(q_0, w) \in F$

- $M$ **recognizes** the language $L(M) = \{w \mid M \text{ accepts } w\}$
Last Time: Regular Languages

A language is called a **regular language** if some finite automaton recognizes it.
Last Time: Finite State Automaton, a.k.a. DFAs

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.

**Key characteristic:**
- Has a **finite** number of states
- I.e., a computer or program with access to a single cell of memory,
  - Where: \# states = the possible symbols that can be written to memory

**Often used for text matching**
Combining DFAs?

Password Requirements

» Passwords must have a minimum length of ten (10) characters - but more is better!
» Passwords **must include at least 3** different types of characters:
  » upper-case letters (A-Z)
  » lower-case letters (a-z)
  » symbols or special characters (%, &, *, $, etc.)
  » numbers (0-9)
» Passwords cannot contain all or part of your email address
» Passwords cannot be re-used

To match all requirements, combine smaller DFAs into one big DFA?

https://www.umb.edu/it/password
Password Checker DFAs

- $M_1$: Check special chars
- $M_2$: Check uppercase
- $M_3$: OR
- $M_4$: Check length
- $M_5$: AND

Want to be able to easily combine DFAs

We want:
OR, AND : DFA $\times$ DFA $\rightarrow$ DFA

To combine more than once, operations must be closed!
“Closed” Operations

- Set of Natural numbers = \{0, 1, 2, ...\}
  - Closed under addition:
    - if \(x\) and \(y\) are Natural numbers,
    - then \(z = x + y\) is a Natural number
  - Closed under multiplication?
    - yes
  - Closed under subtraction?
    - no

- Integers = {..., -2, -1, 0, 1, 2, ...}
  - Closed under addition and multiplication
  - Closed under subtraction?
    - yes
  - Closed under division?
    - no

- Rational numbers = \{\(x\mid x = y/z, y \text{ and } z \text{ are Integers}\}\}
  - Closed under division?
    - No?
    - Yes if \(z \neq 0\)

A set is **closed** under an operation if: the result of applying the operation to members of the set is in the same set.
Why Care About Closed Ops on Reg Langs?

- Closed operations preserve “regularness”

- I.e., it preserves the same computation model!

- This way, a “combined” machine can be “combined” again!

We want:
OR, AND : DFA × DFA → DFA
Password Checker: “OR” = “Union”

$M_3$: OR

$M_1$: Check special chars

$M_2$: Check uppercase
Password Checker: “OR” = “Union”

\[ M_3: \text{OR} \]

\[ M_1: \text{Check special chars} \]

\[ M_2: \text{Check uppercase} \]
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

$$A \cup B = \{\text{good, bad, boy, girl}\}$$
A Closed Operation: Union

**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 $A_1 \cup A_2$
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 “runs” its input on both $M_1$ and $M_2$ in parallel.

$M$ needs to be “in” both an $M_1$ and $M_2$ state simultaneously.

And then accept if either accepts.

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: a **new** machine \( M = (Q, \Sigma, \delta, q_0, F) \) using \( M_1 \) and \( M_2 \)

- 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 \( Q_2 \)

---

**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 \( A_1 \cup A_2 \).

**Idea:** \( M \) “runs” its input on both \( M_1 \) and \( M_2 \) in parallel

---

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**.
Union is Closed For Regular Languages

**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 $A_1 \cup A_2$.

**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:** a **new machine** $M = (Q, \Sigma, \delta, q_0, F)$ using $M_1$ and $M_2$

- **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
\[ (q) = (\delta_1(r_1, a), \delta_2(r_2, a)) \]

- a step in $M_1$, a step in $M_2$

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**.
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 $A_1 \cup A_2$.

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: a new machine $M = (Q, \Sigma, \delta, q_0, F)$ using $M_1$ and $M_2$
  
  - 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: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
    a step in $M_1$, a step in $M_2$
  
  - $M$ start state: $(q_1, q_2)$
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: a **new** machine \( M = (Q, \Sigma, \delta, q_0, F) \) using \( M_1 \) and \( M_2 \)

- **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:** \( \delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \)
  
  a step in \( M_1 \), a step in \( M_2 \)

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

- **\( M \) accept states:** \( F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\} \)
  
  Accept if **either** \( M_1 \) or \( M_2 \) accept

**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 \( A_1 \cup A_2 \).
Another operation: Concatenation

Example: Recognizing street addresses

212 Beacon Street

\[ M_3: \text{CONCAT} \]

\[ M_1: \text{recognize numbers} \]

\[ M_2: \text{recognize words} \]
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)
  - From 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 $A = \{ab, abc\}$
- And $M_2$ recognize language $B = \{cde\}$
- Want: Construct $M$ to recognize $A \circ B = \{abcde, abccde\}$

- But if $M$ sees $ab$ as first part of input ...
- $M$ must decide to either:
Overlapping Concatenation Example

- Let $M_1$ recognize language $A = \{ab, abc\}$
- And $M_2$ recognize language $B = \{cde\}$
- Want: Construct $M$ to recognize $A \circ B = \{abcde, abccde\}$

- But if $M$ sees $ab$ as first part of input ...
- $M$ must decide to either:
  - stay in $M_1$ (correct, if full input is $abcde$)
  - or switch to $M_2$ (correct, if full input is $abccde$)
- But it needs to handle both cases!

Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
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

Nondeterministic computation can be in multiple states at the same time

- accept

New FA
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. \(\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)\) 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!
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. \(\delta: Q \times \Sigma_e \rightarrow \mathcal{P}(Q)\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

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

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

\[
\begin{array}{c|ccc}
 & 0 & 1 & \varepsilon \\
\hline
q_1 & \{q_1\} & \{q_1, q_2\} & \emptyset \\
q_2 & \{q_3\} & \emptyset & \{q_3\} \\
q_3 & \emptyset & \{q_4\} & \emptyset \\
q_4 & \{q_4\} & \{q_4\} & \emptyset \\
\end{array}
\]

- **Empty transition** (no input read)
- **Result of transition is a set**

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

- **Multiple 1 transitions**
- **No 0 transition**
Next Time: Running Programs, NFAs (JFLAP demo): 010110
Check-in Quiz 1/31

On gradescope