Announcements

• Practice midterm solutions available.
• Second practice midterm available.
• Midterm review session this **Saturday, October 27** at **2PM** in **Gates 104**.
  • Come with questions!
  • Leave with answers!
• Problem Set 3 and Problem Set 4 Checkpoints graded; will be returned at end of lecture.
A Friendly Reminder

∀ goes with →

∃ goes with ∧
Finite Automata
DFAs, Informally

- A DFA is defined relative to some alphabet $\Sigma$.
- For each state in the DFA, there must be exactly one transition defined for each symbol in the alphabet.
  - This is the “deterministic” part of DFA.
- There is a unique start state.
- There may be multiple accepting states.
Recognizing Languages with DFAs

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring} \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* | \text{all even-numbered characters of } w \text{ are } 0 \}$
More Elaborate DFAs

\[ L = \{ \ w \mid w \text{ is a C-style comment} \ \} \]

Suppose the alphabet is

\[ \Sigma = \{ \ a, *, / \} \]

Try designing a DFA for comments!

Some test cases:

\[
\begin{array}{ll}
\text{ACCEPTED} & \text{REJECTED} \\
/*a*/ & /**/ \\
/***/ & /***
\end{array}
\]

/*aaa*/aaa/*/
More Elaborate DFAs

$L = \{ w \mid w \text{ is a C-style comment} \}$
More Elaborate DFAs

$L = \{ w \mid w \text{ is a C-style comment} \}$
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a legal email address} \} \]
Tabular DFAs

The star indicates that this is an accepting state.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_3$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_3$</td>
<td>$q_3$</td>
</tr>
</tbody>
</table>
Code? In a Theory Course?

```cpp
int kTransitionTable[kNumStates][kNumSymbols] = {
    {0, 0, 1, 3, 7, 1, ...},
    ...
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    true,
    ...
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input)
        state = kTransitionTable[state][ch];
    return kAcceptTable[state];
}
```
The Complexity of Addition

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 1 \\
+ & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 1 & 1 & 0
\end{array}
\]
Our Alphabet

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1
\end{bmatrix}
\]
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
A Formal Definition of Acceptance

- Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, we can formally define what it means for $D$ to accept a string $w \in \Sigma^*$.

- **Idea**: Define a function $\delta^* : \Sigma^* \rightarrow Q$ that says what state we end up in if we run the DFA on a given string.

- This function represents the effect of running the computer on a given input.
A Formal Definition of Acceptance

- **Notation**: If $\omega$ is a string and $a$ is a character, then $\omega a$ is the string formed by appending $a$ to $\omega$.

- Given a DFA $(Q, \Sigma, \delta, q_0, F)$, $\delta^*$ is defined recursively.

- $\delta^*(\varepsilon) = q_0$
  - Running the automaton on $\varepsilon$ ends in the start state.

- $\delta^*(\omega a) = \delta(\delta^*(\omega), a)$
  - Running on $\omega a$ is equal to running the automaton on $\omega$, then following the transition for $a$. 
A Formal Definition of Acceptance

- Using our $\delta^*$ function, we can formally define the language of a DFA.
- Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
- Define $\mathcal{L}(D) = \{ w \in \Sigma^* | \delta^*(w) \in F \}$
  - The set of strings $w$ that cause the DFA to end up in an accepting state.
So What?

- We now have a mathematically rigorous way of defining whether a DFA accepts a string.
- We can try making changes to DFAs and can formally prove how those changes transform the language of the DFA.
A language $L$ is called a **regular language** iff there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
The Complement of a Language

• Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ not in $L$.

• Formally:

$$\overline{L} = \Sigma^* - L$$
Complementing Regular Languages

• Recall: A **regular language** is a language accepted by some DFA.

• **Question**: If $L$ is a regular language, is $\overline{L}$ a regular language?

• If the answer is “yes,” then there must be some way to construct a DFA for $\overline{L}$.

• If the answer is “no,” then some language $L$ can be accepted by a DFA, but $\overline{L}$ cannot be accepted by any DFA.
Complementing Regular Languages

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring} \} \]

\[ \bar{L} = \{ \ w \in \{0, 1\}^* \mid w \text{ does not contain 00 as a substring} \} \]
Complementing Regular Languages

\[ \overline{L} = \{ \, w \mid w \text{ is not a legal email address} \, \} \]
Constructions on Automata

- Much of our discussion of automata will consider constructions that transform one automaton into another.
- Exchanging accepting and rejecting states is a simple construction sometimes called the "complement construction".
- Does this construction always work?
- How would we prove it?
Theorem: If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $\mathcal{L}(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $\overline{\mathcal{L}(D)}$.

Proof: By definition, $\mathcal{L}(D') = \{ w \in \Sigma^* | \delta^*(w) \in Q - F \}$. So

$$\mathcal{L}(D') = \{ w \in \Sigma^* | \delta^*(w) \in Q \land \delta^*(w) \notin F \}$$

Since $\delta^*: \Sigma^* \rightarrow Q$, any string $w \in \Sigma^*$ satisfies $\delta^*(w) \in Q$. Thus

$$\mathcal{L}(D') = \{ w \in \Sigma^* | w \in \Sigma^* \} - \{ w \in \Sigma^* | \delta^*(w) \in F \}$$

$$\mathcal{L}(D') = \Sigma^* - \{ w \in \Sigma^* | \delta^*(w) \in F \}$$

$$\mathcal{L}(D') = \Sigma^* - \mathcal{L}(D)$$

$$\mathcal{L}(D') = \overline{\mathcal{L}(D)}. \qed$$
Closure Properties

- If $L$ is a regular language, $\overline{L}$ is a regular language.
- If we begin with a regular language and complement it, we end up with a regular language.
- This is an example of a closure property of regular languages.
  - The regular languages are closed under complementation.
  - We'll see more such properties later on.
NFAs

• An **NFA** is a
  • Nondeterministic
  • Finite
  • Automaton

• Conceptually similar to a DFA, but equipped with the vast power of **nondeterminism**.
(Non)determinism

- A model of computation is **deterministic** if at every point in the computation, there is exactly one choice that can make.
- The machine accepts if that series of choices leads to an accepting state.
- A model of computation is **nondeterministic** if the computing machine may have multiple decisions that it can make at one point.
- The machine accepts if **any** series of choices leads to an accepting state.
A Simple NFA

$q_0$ has two transitions defined on 1!
A More Complex NFA

If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.
Intuiting Nondeterminism

- Nondeterministic machines are a serious departure from physical computers.
- How can we build up an intuition for them?
- Three approaches:
  - Tree computation
  - Perfect guessing
  - Massive parallelism
Tree Computation

0 1 0 1 1 0
Nondeterminism as a Tree

- At each decision point, the automaton clones itself for each possible decision.
- The series of choices forms a directed, rooted tree.
- At the end, if any active accepting states remain, we accept.
Perfect Guessing

• We can view nondeterministic machines as having **Magic Superpowers** that enable them to guess the correct choice of moves to make.

• Idea: Machine can always guess the right choice if one exists.

• No physical analog for something of this sort.
  
  • (Those of you thinking quantum computing – this is not the same thing. We actually don't fully know the relation between quantum and nondeterministic computation.)
Massive Parallelism

- An NFA can be thought of as a DFA that can be in many states at once.
- Each symbol read causes a transition on every active state into each potential state that could be visited.
- Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
  - No fixed limit on processors; makes multicore machines look downright wimpy!
So What?

• We will turn to these three intuitions for nondeterminism more later in the quarter.

• Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:
  • Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?
  • Can any problem that can be solved by a nondeterministic machine be solved **efficiently** by a deterministic machine?

• The answers vary from automaton to automaton.
ε-Transitions

- NFAs have a special type of transition called the \textbf{ε-transition}.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.