Finite Automata

Part Two
Recap from Last Time
Strings

• An *alphabet* is a finite, nonempty set of symbols called *characters*.
  • Typically, we use the symbol $\Sigma$ to refer to an alphabet.
• A *string over an alphabet* $\Sigma$ is a finite sequence of characters drawn from $\Sigma$.
• Example: If $\Sigma = \{a, b\}$, here are some valid strings over $\Sigma$:
  
  a    aabaaabbabaaabaaaabbb    abbababba

• The *empty string* has no characters and is denoted $\varepsilon$.
• Calling attention to an earlier point: since all strings are finite sequences of characters from $\Sigma$, you cannot have a string of infinite length.
Languages

- A **formal language** is a set of strings.
- We say that $L$ is a **language over** $\Sigma$ if it is a set of strings over $\Sigma$.
- Example: The language of palindromes over $\Sigma = \{a, b, c\}$ is the set
  - $\{\varepsilon, a, b, c, aa, bb, cc, aaa, aba, aca, bab, \ldots \}$
- The set of all strings composed from letters in $\Sigma$ is denoted $\Sigma^*$.
- Formally, we say that $L$ is a language over $\Sigma$ if $L \subseteq \Sigma^*$. 
A Simple Finite Automaton

\[
\begin{align*}
q_0 & \xrightarrow{0} q_1 \\
q_1 & \xrightarrow{0} q_0 \\
q_3 & \xrightarrow{0} q_2 \\
q_2 & \xrightarrow{0} q_3
\end{align*}
\]

\[
\text{start} \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0
\]
A Simple Finite Automaton

\[
\begin{array}{c}
\text{start} \\
q_0 \quad 0 \quad q_1 \\
1 \quad 1 \quad 0 \quad 1 \quad 1 \\
q_3 \quad 0 \quad q_2 \\
0 \quad 0 \quad 0 \\
\end{array}
\]
The **language of an automaton** is the set of strings that it accepts.

If $D$ is an automaton, we denote the language of $D$ as $\mathcal{L}(D)$.

$$\mathcal{L}(D) = \{ w \in \Sigma^* \mid D \text{ accepts } w \}$$
DFAs

• A **DFA** is a
  • **D**eterministic
  • **F**inite
  • **A**utomaton

• DFAs are the simplest type of automaton that we will see in this course.
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 $\Sigma$.
  - This is the “deterministic” part of DFA.
- There is a unique start state.
- There are zero or more accepting states.
Designing DFAs

- At each point in its execution, the DFA can only remember what state it is in.
- **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states $\approx$ only finitely many different things the machine can remember.
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* | \text{the number of } b\text{'s in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \} \]
New Stuff!
Tabular DFAs

These stars indicate accepting states.
Since this is the first row, it's the start state.
My Turn to Code Things Up!

```cpp
int kTransitionTable[kNumStates][kNumSymbols] = {
    {0, 0, 1, 3, 7, 1, ...},
    ...
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    ...
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input) {
        state = kTransitionTable[state][ch];
    }
    return kAcceptTable[state];
}
```
The Regular Languages
A language $L$ is called a *regular language* if 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 $\Sigma^*$ that aren't in $L$.

- Formally:

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

• As we saw a few minutes ago, a regular language is a language accepted by some DFA.

• Question: If \( L \) is a regular language, is \( \overline{L} \) necessarily a regular language?

• If the answer is “yes,” then if there is a way to construct a DFA for \( L \), 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 some DFA, but \( \overline{L} \) cannot be accepted by any DFA.
Computational Device for $L$

Computational Device for $\overline{L}$
Complementing Regular Languages

\[ L = \{ \ w \in \{a, b\}^* \ | \ w \text{ contains } aa \text{ as a substring } \} \]

\[ \bar{L} = \{ \ w \in \{a, b\}^* \ | \ w \text{ does not contain } aa \text{ as a substring } \} \]
More Elaborate DFAs

$L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \}$
More Elaborate DFAs

\[ \overline{L} = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
Closure Properties

- **Theorem:** If $L$ is a regular language, then $\overline{L}$ is also a regular language.
- As a result, we say that the regular languages are *closed under complementation*.
Time-Out For Announcements!
CODE2040 INFO SESSION
FRIDAY, NOV 3 | 7PM - 8PM
OLD UNION, ROOM 200
Talk to Your Provost

● Provost Persis Drell will be holding office hours in Lathrop 143 next Monday, November 6, from 4PM – 6PM.

● Have any suggestions for the university? Want to change anything? Stop on by to chat!
CS Career Panel

• Greg Ramel, our wonderful CS course advisor, is organizing a CS career panel.

• It’s tomorrow (Thursday, November 2\textsuperscript{nd}) in Gates 219 and runs from 5:45PM – 7:00PM.

• Please RSVP using \textcolor{red}{this link}.

• There’s a great mix of panelists. Highly recommended!
Problem Set Four Graded

75th Percentile: 68 / 72 (94%)
50th Percentile: 61 / 72 (85%)
25th Percentile: 53 / 72 (74%)
Extra Practice Problems 2

- We’ve just released another set of extra practice problems to the course website.
- Need to review some concepts? Want more practice? Try these questions out! There’s a ton of variety.
- Solutions will go out on Friday.
Your Questions
“I've struggled as a public speaker for as long as I can remember and watching your lectures, I'm amazed by how good you are. How do you do it?”

When I first started teaching I was so terrified of speaking to crowds that I literally memorized everything I was going to say and rehearsed for like ten hours each time. After I realized that most people are nice and won’t eat you if you make a mistake, I started backing down from that and just worked out a general game plan for each lecture. Essentially, I just slowly stepped up the amount of improvisation I had in each lecture until I got the hang of it. Right now there’s a balance between making things up as I go and falling back on things I know work well.
“Any movie / TV recommendations?”


Unsorted top TV shows: “The Wire.”
Back to CS103!
NFAs

• An **NFA** is a
  • **N**ondeterministic
  • **F**inite
  • **A**utomaton

• Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation.
(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.
A More Complex NFA

Question to ponder: What does this NFA accept?
NFA Acceptance

• An NFA $N$ accepts a string $w$ if there is some series of choices that lead to an accepting state.

• Consequently, an NFA $N$ rejects a string $w$ if no possible series of choices lead it into an accepting state.

• It's easier to show that an NFA does accept something than to show that it doesn't
**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.
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.

• NFAs are not required to follow ε-transitions. It's simply another option at the machine's disposal.
Intuiting Nondeterminism

• Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?

• There are two particularly useful frameworks for interpreting nondeterminism:
  • *Perfect guessing*
  • *Massive parallelism*
Perfect Guessing
Perfect Guessing

- We can view nondeterministic machines as having *Magic Superpowers* that enable them to guess choices that lead to an accepting state.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the wrong guesses.
- No known physical analog for this style of computation – this is totally new!
Massive Parallelism

\[ q_3 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0 \rightarrow \Sigma \]

\[ a \quad b \quad a \quad b \quad a \quad b \quad a \quad a \]
Massive Parallelism
Massive Parallelism
Massive Parallelism

\[
\begin{align*}
\Sigma & \rightarrow q_0 \\
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} \rightarrow q_3
\end{align*}
\]

Input:

a b a b a b a
Massive Parallelism

We’re in at least one accepting state, so there’s some path that gets us to an accepting state. Therefore, we accept!
Massive Parallelism

• An NFA can be thought of as a DFA that can be in many states at once.
• At each point in time, when the NFA needs to follow a transition, it tries all the options at the same time.
• (Here's a rigorous explanation about how this works; read this on your own time).
  • Start off in the set of all states formed by taking the start state and including each state that can be reached by zero or more ε-transitions.
  • When you read a symbol a in a set of states S:
    - Form the set S’ of states that can be reached by following a single a transition from some state in S.
    - Your new set of states is the set of states in S’, plus the states reachable from S’ by following zero or more ε-transitions.
So What?

- Each intuition of nondeterminism is useful in a different setting:
  - Perfect guessing is a great way to think about how to design a machine.
  - Massive parallelism is a great way to test machines – and has nice theoretical implications.
- 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.
Designing NFAs
Designing NFAs

• When designing NFAs, *embrace the nondeterminism!*

• Good model: *Guess-and-check*:
  
  • Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  
  • Then, have the machine *deterministically check* that the choice was correct.

• The *guess* phase corresponds to trying lots of different options.

• The *check* phase corresponds to filtering out bad guesses or wrong options.
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]

Nondeterministically guess when to leave the start state.
Deterministically check whether that was the right time to do so.
Guess-and-Check

$L = \{ \ w \in \{a, b, c\}^* \ | \ \text{at least one of } a, b, \text{ or } c \ \text{is not in } w \ \}$

Nondeterministically guess which character is missing.

Deterministically check whether that character is indeed missing.
Just how powerful are NFAs?