Finite Automata

Part Two
Recap from Last Time
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

● 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.
A Sample DFA

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
New Stuff!
Tabular DFAs

These stars indicate accepting states.
Tabular DFAs

Since this is the first row, it’s the start state.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*q₀</td>
<td>q₁</td>
<td>q₀</td>
</tr>
<tr>
<td>q₁</td>
<td>q₃</td>
<td>q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>q₃</td>
<td>q₀</td>
</tr>
<tr>
<td>*q₃</td>
<td>q₃</td>
<td>q₃</td>
</tr>
</tbody>
</table>
Tabular DFAs

Question to ponder: Why isn’t there a column here for \( \Sigma \)?

<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>
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 $\mathcal{L}(D) = L$.

If $L$ is a language and $\mathcal{L}(D) = L$, we say that $D$ **recognizes** the language $L$. 
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:

$$\overline{L} = \Sigma^* - L$$

Good proofwriting exercise: prove $\overline{L} = L$ for any language $L$. 
Complementing Regular Languages

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

$\bar{L} = \{ \, w \in \{a, b\}^* \mid w \text{ does not contain } aa \text{ as a substring } \}$
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*.

Question to ponder: are the nonregular languages closed under complementation?
NFAs
Revisiting a Problem

\[
\begin{align*}
q_0 & \xrightarrow{0, 1} q_1 \\
q_1 & \xrightarrow{1} q_2 \\
q_2 & \xrightarrow{0, 1} q_3 \\
q_3 & \xrightarrow{0, 1} q_2
\end{align*}
\]
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.

  • (This sort of nondeterminism is technically called **existential nondeterminism**, the most philosophical-sounding term we’ll introduce all quarter.)
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 does not accept.
The language of an NFA is

\[ \mathcal{L}(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \}. \]

What is the language of this NFA?
(Assume \( \Sigma = \{h, i\} \).)
The language of an NFA is

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

What is the language of this NFA?  
(Assume $\Sigma = \{0, 1\}$.)
The language of an NFA is
\[ \mathcal{L}(N) = \{ \, w \in \Sigma^* \mid N \text{ accepts } w \, \}. \]

What is the language of this NFA?
(Assume \( \Sigma = \{0, 1\} \).)
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.
\(\varepsilon\text{-Transitions}\)

- NFAs have a special type of transition called the \(\varepsilon\text{-transition}\).
- An NFA may follow any number of \(\varepsilon\)-transitions at any time without consuming any input.

Not at all fun or rewarding exercise: what is the language of this NFA?
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.
Time-Out For Announcements!
ALTERNATIVE SPRING BREAK 2019

REWIRING THE ELECTRIC BRAIN:

EXPLORING THE ROLE OF TECH IN LANGUAGE REVITALIZATION

REGISTER FOR ASB BY FRIDAY, NOVEMBER 2, 11:59 PM

ASSU SPECIAL FEES

HTTP://ASB.STANFORD.EDU
Stanford Women in Computer Science

CASUAL DINNER

Tuesday, November 6th from 5-7 PM at Gates 403

Come have dinner with CS students and faculty. Everyone is welcome, especially students just starting out in CS!
Midterms Graded

- Midterms have been graded! If you didn’t pick yours up yet, you can grab it from the Gates building.
  - SCPD students - exams have been sent back to the SCPD distribution office. If you haven’t received yours yet, ping the SCPD distribution office.
- We’ve posted a regrade request form on the course website with instructions about how to ask for a regrade. Regrade requests are due next Wednesday.
Your Questions
“I did bad on the midterm. How can I ask for help to solidify those past topics when in doing so, I'm not doing the PSET?”

Allow me to make a series of analogies. 😊

If you’re shaky on one of the fundamental topics, you will likely end up saving time in the long run by drilling those skills until they’re solidified.
“Favorite Halloween costume you’ve ever seen/you’ve ever come up with?”

This is also a candidate for “most embarrassing thing that’s ever happened to you.”
Back to CS103!
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 positive guessing*
  - *Massive parallelism*
Perfect Positive 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.
- There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!
Massive Parallelism

\[ \Sigma \]

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

- \text{a b a b a b a a}
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input:

\[ a \ b \ a \ b \ a \ b \ a \]

Start state: \( q_0 \)

Final state: \( q_3 \)
Massive Parallelism

\[ q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \]

Input sequence: ababaaba
Massive Parallelism

\[ \Sigma \]

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ ababa \]

\[ \text{start} \]
Massive Parallelism

\[ \sum \]

\[ q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \]

\[ a \ b \ a \ b \ a \ b \ a \]
We're in at least one accepting state, so there's some path that gets us to an accepting state.

\[ \Sigma \]

\[ a \quad b \quad a \quad b \quad a \quad a \]
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 \textit{efficiently} by a deterministic machine?

• The answers vary from automaton to automaton.
Designing NFAs
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

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

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

Nondeterministically guess when the end of the string is coming up.

Deterministically check whether you were correct.
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
Guess-and-Check

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

Nondeterministically guess which character is missing.

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

- **The Powerset Construction**
  - So beautiful. So elegant. So cool!
- **More Closure Properties**
  - Other set-theoretic operations.
- **Language Transformations**
  - What’s the deal with the notation $\Sigma^*$?