Finite Automata
Part Three
Hello Condensed Slide Readers!

The first half of this lecture consists almost exclusively of animations of the subset construction, which aren’t present here in the condensed version of these slides. You may want to see the full version of the slides for more context.

Enjoy!

-Keith
Recap from Last Time
Tabular DFAs

These stars indicate accepting states.

<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>
Since this is the first row, it's the start state.
A language $L$ is called a regular language if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Can have missing transitions or multiple transitions defined on the same input symbol.
- Accepts if *any possible series of choices* leads to an accepting state.
**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.
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.

• The NFA accepts if *any* of the states that are active at the end are accepting states. It rejects otherwise.
Just how powerful are NFAs?
New Stuff!
NFAs and DFAs

• Any language that can be accepted by a DFA can be accepted by an NFA.

• Why?
  • Every DFA essentially already is an NFA!

• Question: Can any language accepted by an NFA also be accepted by a DFA?

• Surprisingly, the answer is yes!
Thought Experiment:
How would you simulate an NFA in software?
\begin{align*}
\Sigma & \quad \rightarrow a \quad q_0 \quad \rightarrow b \quad q_1 \quad \rightarrow a \quad q_2 \quad \rightarrow a \quad q_3 \\
\{q_0\} & \quad \rightarrow a \quad \{q_0, q_1\} \quad \rightarrow b \quad \{q_0\} \\
\{q_0, q_1\} & \quad \rightarrow a \quad \{q_0, q_1\} \quad \rightarrow b \quad \{q_0, q_2\} \\
\{q_0, q_2\} & \quad \rightarrow a \quad \{q_0, q_1, q_3\} \quad \rightarrow b \quad \{q_0\} \\
\{q_0, q_1, q_3\} & \quad \rightarrow a \quad \{q_0, q_1\} \quad \rightarrow b \quad \{q_0, q_2\} \\
\end{align*}
Once More, With Epsilons!

\[
\begin{array}{c}
\text{start} \\
q_0 \\
q_3 \\
\qquad \Sigma \quad \epsilon \\
q_2 \\
q_1 \\
q_4 \\
\end{array}
\]

\[
\begin{array}{c|cc|}
 & \text{a} & \text{b} \\
\hline
\{q_0, q_3\} & \{q_1, q_4\} & \{q_4\} \\
\{q_1, q_4\} & \emptyset & \{q_2, q_3\} \\
\{q_4\} & \emptyset & \{q_3\} \\
\{q_0, q_3, q_4\} & \{q_0, q_3, q_4\} & \emptyset \\
\{q_3\} & \{q_4\} & \{q_4\} \\
\{q_0, q_3, q_4\} & \{q_1, q_4\} & \{q_3, q_4\} \\
\{q_3, q_4\} & \{q_4\} & \{q_3, q_4\} \\
\emptyset & \emptyset & \emptyset \\
\end{array}
\]
The Subset Construction

- This construction for transforming an NFA into a DFA is called the *subset construction* (or sometimes the *powerset construction*).
  - Each state in the DFA is associated with a set of states in the NFA.
  - The start state in the DFA corresponds to the start state of the NFA, plus all states reachable via \(\varepsilon\)-transitions.
  - If a state \(q\) in the DFA corresponds to a set of states \(S\) in the NFA, then the transition from state \(q\) on a character \(a\) is found as follows:
    - Let \(S'\) be the set of states in the NFA that can be reached by following a transition labeled \(a\) from any of the states in \(S\). (*This set may be empty.*)
    - Let \(S''\) be the set of states in the NFA reachable from some state in \(S'\) by following zero or more epsilon transitions.
    - The state \(q\) in the DFA transitions on \(a\) to a DFA state corresponding to the set of states \(S''\).

- *Read Sipser for a formal account.*
The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.
- **Useful fact:** $|\mathcal{P}(S)| = 2^{|S|}$ for any finite set $S$.
- In the worst-case, the construction can result in a DFA that is *exponentially larger* than the original NFA.
- **Question to ponder:** Can you find a family of languages that have NFAs of size $n$, but no DFAs of size less than $2^n$?
A language $L$ is called a **regular language** if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
An Important Result

**Theorem:** A language $L$ is regular iff there is some NFA $N$ such that $\mathcal{L}(N) = L$.

**Proof Sketch:** If $L$ is regular, there exists some DFA for it, which we can easily convert into an NFA. If $L$ is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so $L$ is regular. ■
Why This Matters

- We now have two perspectives on regular languages:
  - Regular languages are languages accepted by DFAs.
  - Regular languages are languages accepted by NFAs.
- We can now reason about the regular languages in two different ways.
Time-Out for Announcements!
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!
Problem Set Six

- Problem Set Five was due at 2:30PM today.
- Problem Set Six goes out today. It’s due next Friday at 2:30PM.
  - Play around with DFAs, NFAs, language transformations, and their properties!
  - Explore how all the discrete math topics we’ve talked about so far come into play!
DFA/NFA Editor

- We have an online DFA/NFA editor you’ll use to answer and submit some of the questions for PS6.
- This tool will let you design and test your automata on a number of different inputs.
- You can also use it to explore on your own!
Looking for a Partner?

- I’ve heard from many of you that you’re now looking for a problem set partner.
- Don’t forget that Piazza has a lovely “Search for Teammates” feature that you can use to do this.
- It’s like speed dating for theory!
Midterm Practice Problems

• If you’d like to get a jump on studying for the second midterm, feel free to work through the five practice exams we’ve posted to the course website.

• There’s also Extra Practice Problems 2 to work through.

• We’ll be holding a practice midterm exam next Wednesday evening from 7PM - 10PM, location TBA. It’ll use an exam that’s not yet posted to the course website.
Your Questions
“What do you think about working for companies with interesting technical challenges but questionable ethics (ie writing software to support bad governments)?”

You ultimately need to be able to sleep at night doing what you’re doing. If you can’t comfortably do that, you should consider finding work elsewhere.

I would caution that the issue of “questionable ethics” is a lot more subtle than it might initially appear.
“How do you talk to professors?”

Professors are people, and they’re usually really approachable! Crash their office hours and ask them about their work. If you can do a little background reading first, great! Now you have something to talk about.

Another option: find their PhD students and talk to them about their experiences. They might make an introduction for you!
Back to CS103!
Properties of Regular Languages
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.

- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$ regular?

**Question to ponder:** where have you seen this idea before?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?

Hey, it's De Morgan's laws!
Concatenation
String Concatenation

• If $w \in \Sigma^*$ and $x \in \Sigma^*$, the **concatenation** of $w$ and $x$, denoted $wx$, is the string formed by tacking all the characters of $x$ onto the end of $w$.

• Example: if $w = \text{quo}$ and $x = \text{kka}$, the concatenation $wx = \text{quokka}$.

• Analogous to the $+$ operator for strings in many programming languages.

• Some facts about concatenation:
  • The empty string $\varepsilon$ is the **identity element** for concatenation:
    $$we = \varepsilon w = w$$
  • Concatenation is **associative**:
    $$wxy = w(xy) = (wx)y$$
Concatenation

The concatenation of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

$$L_1L_2 = \{ wx \in \Sigma^* | w \in L_1 \land x \in L_2 \}$$
Concatenation Example

- Let $\Sigma = \{ a, b, ..., z, A, B, ..., Z \}$ and consider these languages over $\Sigma$:
  - $Noun = \{ Puppy, Rainbow, Whale, ... \}$
  - $Verb = \{ Hugs, Juggles, Loves, ... \}$
  - $The = \{ The \}$
  - The language $TheNounVerbTheNoun$ is
    - $\{ ThePuppyHugsTheWhale, TheWhaleLovesTheRainbow, TheRainbowJugglesTheRainbow, ... \}$
Concatenation

- The **concatenation** of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

  \[ L_1L_2 = \{ wx \in \Sigma^* | w \in L_1 \land x \in L_2 \} \]

- Two views of $L_1L_2$:
  - The set of all strings that can be made by concatenating a string in $L_1$ with a string in $L_2$.
  - The set of strings that can be split into two pieces: a piece from $L_1$ and a piece from $L_2$.

This is closely related to, but different than, the Cartesian product.

**Question to ponder:** In what ways are concatenations similar to Cartesian products? In what ways are they different?
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

Machine for $L_1$

Machine for $L_2$

book

keeper
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?
- **Idea**: Run the automaton for $L_1$ on $w$, and whenever $L_1$ reaches an accepting state, optionally hand the rest off $w$ to $L_2$.
  - If $L_2$ accepts the remainder, then $L_1$ accepted the first part and the string is in $L_1L_2$.
  - If $L_2$ rejects the remainder, then the split was incorrect.
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$

Machine for $L_1L_2$
Lots and Lots of Concatenation

- Consider the language \( L = \{ \text{aa, b} \} \)
- \( LL \) is the set of strings formed by concatenating pairs of strings in \( L \).
  \[ \{ \text{aaaa, aab, baa, bb} \} \]
- \( LLL \) is the set of strings formed by concatenating triples of strings in \( L \).
  \[ \{ \text{aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbbaa, bbb} \} \]
- \( LLLL \) is the set of strings formed by concatenating quadruples of strings in \( L \).
  \[ \{ \text{aaaaaaaaa, aaaaaaab, aaaaabaa, aaaaabb, aabaaaa, aabaab, aabbaa, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbaaaaa, bbaab, bbbbaa, bbbb} \} \]
Language Exponentiation

- We can define what it means to “exponentiate” a language as follows:
  - $L^0 = \{ \varepsilon \}$
    - The set containing just the empty string.
    - Idea: Any string formed by concatenating zero strings together is the empty string.
  - $L^{n+1} = LL^n$
    - Idea: Concatenating $(n+1)$ strings together works by concatenating $n$ strings, then concatenating one more.
- **Question to ponder:** Why define $L^0 = \{ \varepsilon \}$?
- **Question to ponder:** What is $\emptyset^0$?
The Kleene Closure

• An important operation on languages is the **Kleene Closure**, which is defined as

\[ L^* = \{ w \in \Sigma^* \mid \exists n \in \mathbb{N}. w \in L^n \} \]

• Mathematically:

\[ w \in L^* \iff \exists n \in \mathbb{N}. w \in L^n \]

• Intuitively, all possible ways of concatenating zero or more strings in \( L \) together, possibly with repetition.

• **Question to ponder:** What is \( \emptyset^* \)?
The Kleene Closure

If \( L = \{ a, bb \} \), then \( L^* = \{ \)

\[ \varepsilon, \]
\[ a, bb, \]
\[ aa, abb, bba, bbbb, \]
\[ aaa, aabb, abba, abbb, bbbaa, bbabb, bbbba, bbbbbbb, \]
\[ ... \]

\}
Reasoning about Infinity

• If $L$ is regular, is $L^*$ necessarily regular?

• ⚠ A Bad Line of Reasoning: ⚠
  
  • $L^0 = \{ \varepsilon \}$ is regular.
  • $L^1 = L$ is regular.
  • $L^2 = LL$ is regular
  • $L^3 = L(LL)$ is regular
  • ...

• Regular languages are closed under union.
• So the union of all these languages is regular.
Reasoning About the Infinite

• If a series of finite objects all have some property, the “limit” of that process does not necessarily have that property.

• In general, it is not safe to conclude that some property that always holds in the finite case must hold in the infinite case.
  • (This is why calculus is interesting).
Idea: Can we directly convert an NFA for language $L$ to an NFA for language $L^*$?
The Kleene Star

Machine for $L$
Question: Why add the new state out front? Why not just make the old start state accepting?
Closure Properties

- **Theorem:** If $L_1$ and $L_2$ are regular languages over an alphabet $\Sigma$, then so are the following languages:
  - $\overline{L_1}$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1L_2$
  - $L_1^*$

- These properties are called **closure properties of the regular languages**.
Next Time

• **Regular Expressions**
  • Building languages from the ground up!

• **Thompson’s Algorithm**
  • A UNIX Programmer in Theoryland.

• **Kleene’s Theorem**
  • From machines to programs!