Finite Automata

Part Three
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>
Tabular DFAs

Since this is the first row, it's the start state.
If $D$ is a DFA, the **language of $D$**, denoted $\mathcal{L}(D)$, is \{ $w \in \Sigma^* \mid D$ accepts $w$ \}.

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
  • Nondeterministic
  • Finite
  • Automaton
• 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 \( \varepsilon \)-transition.

• An NFA may follow any number of \( \varepsilon \)-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?
\[
\begin{array}{c|c|c}
\text{state} & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\end{array}
\]
The Subset Construction

- This procedure for turning an NFA for a language $L$ into a DFA for a language $L$ is called the **subset construction**.
  - It’s sometimes called the **powerset construction**; it’s different names for the same thing!

- Intuitively:
  - Each state in the DFA corresponds to a set of states from the NFA.
  - Each transition in the DFA corresponds to what transitions would be taken in the NFA when using the massive parallel intuition.
  - The accepting states in the DFA correspond to which sets of states would be considered accepting in the NFA when using the massive parallel intuition.

- There’s an online **Guide to the Subset Construction** with a more elaborate example involving $\varepsilon$-transitions and cases where the NFA dies; check that for more details.
The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.
- **Useful fact:** $|\wp(S)| = 2^{|S|}$ 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 if and only if there is some NFA \( N \) such that \( \mathcal{L}(N) = L \).

**Proof Sketch:** Pick a language \( L \). First, assume \( L \) is regular. That means there’s a DFA \( D \) where \( \mathcal{L}(D) = L \). Every DFA is “basically” an NFA, so there’s an NFA \( (D) \) whose language is \( L \).

Next, assume there’s an NFA \( N \) such that \( \mathcal{L}(N) = L \). Using the subset construction, we can build a DFA \( D \) where \( \mathcal{L}(N) = \mathcal{L}(D) \). Then we have that \( \mathcal{L}(D) = L \), so \( L \) is regular. ■-ish
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!
Many of these grades are because folks forgot to list partners - please check to make sure you’re getting credit for the work you’re doing, and let us know if your partner forgot to add you.
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.
  - Design DFAs and NFAs for a range of problems!
  - Explore formal language theory!
  - See some clever applications!
Second Midterm Logistics

• Our second midterm exam is a 49-hour take-home exam that goes out next Friday (November 5\textsuperscript{th}) at 2:30PM and comes due next Sunday (November 7\textsuperscript{th}) at 2:30PM Pacific time.
  • It’s 49 hours long because of the switch to Daylight Saving Time.
• Topic coverage is PS3 – PS5 and lectures 07 – 13 (functions through induction). Later topics (automata, formal languages) won’t be tested. Earlier topics are fair game for the exam, since the material in this class builds on itself.
• We’ve released Extra Practice Problems 2, a collection of 18 problems with solutions, to the course website to help you prepare.
• And always, keep the TAs in the loop! Let us know what we can do to help out.
Three Questions

• What’s something you know now that, at the start of the quarter, you knew you didn’t know?
• What’s something you know now that, at the start of the quarter, you didn’t know you didn’t know?
• What’s something you don’t know now that, at the start of the quarter, you didn’t know you didn’t know?
Your Questions

Next time, because I forgot to set that up today. Oops.
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 $L_1 \cup L_2$?

**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 $wx = \text{quokka}$.

- This is analogous to the + operator for strings in many programming languages.

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

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

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

Let $\Sigma = \{ a, b, ..., z, A, B, ..., Z \}$ and consider these languages over $\Sigma$:

- **Noun** = $\{ \text{Puppy, Rainbow, Whale, ... } \}$
- **Verb** = $\{ \text{Hugs, Juggles, Loves, ... } \}$
- **The** = $\{ \text{The} \}$
- **The** language $\text{TheNounVerbTheNoun}$ is
  $\{ \text{ThePuppyHugsTheWhale, TheWhaleLovesTheRainbow, TheRainbowJugglesTheRainbow, ... } \}$
Concatenation

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

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

- 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 $L_2$**

bookkeeper
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 $L_1$](image1)
![Machine for $L_2$](image2)

$book \quad keeper$
Concatenating Regular Languages

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

**Idea:**

- Run a DFA/NFA for $L_1$ on $w$.
- Whenever it reaches an accepting state, optionally hand the rest of $w$ to a DFA/NFA for $L_2$.
- If the automaton for $L_2$ accepts the rest, $w \in L_1 L_2$.
- If the automaton for $L_2$ rejects the remainder, 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 $L_1 L_2$
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 $L$.
  $$\{ \text{aaaaaaaa, aaaaaaab, aaabaa, aaaaab, aaaaaaa, aabaaa, aabaab, aabbaa, aababb, baaaaaa, baaaaab, baabaa, baabb, bbaaaa, bbaaab, bbbaa, bbbb} \}$$
Language Exponentiation

• We can define what it means to “exponentiate” a language as follows:

• \( L^0 = \{ \varepsilon \} \)
  • Intuition: The only string you can form by gluing no strings together is the empty string.
  • Notice that \( \{ \varepsilon \} \neq \emptyset \). Can you explain why?

• \( L^{n+1} = LL^n \)
  • 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 Star
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, \( L^* \) is the language 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 $L = \{ \text{a, bb} \}$, then $L^* = \{$

$\epsilon$,

$\text{a, bb}$,

$\text{aa, abb, bba, bbb}$,

$\text{aaa, aabb, abba, abbb, bbba, babb, bbbba, bbbbb}$,

$\ldots$

$\}$

Think of $L^*$ as the set of strings you can make if you have a collection of stamps – one for each string in $L$ – and you form every possible string that can be made from those stamps.
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).

• So our earlier argument \( L^* = L^0 \cup L^1 \cup \ldots \) isn’t going to work.

• We need a different line of reasoning.
Idea: Can we directly convert an NFA for language \( L \) to an NFA for language \( L^* \)?
The Kleene Star

Machine for $L$
The Kleene Star

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!