Recap from Last Time
These stars indicate accepting states.
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
  • **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?
\[
\begin{array}{ccc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\end{array}
\]
\begin{center}
\begin{tikzpicture}
\node[state, initial, accepting] (q0) {$q_0$};
\node[state, right of=q0] (q1) {$q_1$};
\node[state, right of=q1] (q2) {$q_2$};
\node[state, accepting, right of=q2] (q3) {$q_3$};
\draw[->] (q0) edge node {$a$} (q1);
\draw[->] (q1) edge node {$b$} (q2);
\draw[->] (q2) edge node {$a$} (q3);
\draw[->] (q3) edge[loop above] node {$\Sigma$} (q3);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
 & a & b \\
\hline
$\{q_0\}$ & $\{q_0, q_1\}$ & $\{q_0\}$ \\
\hline
$\{q_0, q_1\}$ & $\{q_0, q_1\}$ & $\{q_0, q_2\}$ \\
\hline
$\{q_0, q_2\}$ & $\{q_0, q_1, q_3\}$ & $\{q_0\}$ \\
\hline
$\{q_0, q_1, q_3\}$ & $\{q_0, q_1\}$ & $\{q_0, q_2\}$ \\
\hline
\end{tabular}
\end{center}
\[
\begin{array}{c}
\begin{array}{c}
\text{start} \\
\begin{array}{cc}
q_0 & a \\
q_1 & b \\
q_2 & a \\
q_3 & \\
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\Sigma \\
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\{q_0\} & a & \{q_0, q_1\} & b & \{q_0\} \\
\{q_0, q_1\} & a & \{q_0, q_1\} & b & \{q_0, q_2\} \\
\{q_0, q_2\} & a & \{q_0, q_1, q_3\} & b & \{q_0\} \\
*\{q_0, q_1, q_3\} & a & \{q_0, q_1\} & b & \{q_0, q_2\} \\
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\text{start} \\
\begin{array}{cc}
\{q_0\} & a \\
\{q_0, q_1\} & b \\
\{q_0, q_2\} & a \\
*\{q_0, q_1, q_3\} & b \\
\end{array}
\end{array}
\end{array}
\end{array}
\]
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:** $|\mathcal{P}(S)| = 2^{|S|}$ 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 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.
  • Want to use late days? One late day will extend this deadline to 2:30PM Saturday, and a second will extend it to 2:30PM Sunday.

• 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!
- One quick note: unlike the previous coding questions, we will only run the autograder once the problem set comes due. As a result, make sure to test your solutions thoroughly before submitting!
  
  - Think about edge cases. What are some small strings that might break things? Some longer strings?
  - Pretend you haven’t looked at your automata and just saw the language itself. What would be cases you’d expect would be really tricky?
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 four 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
“How can you "differentiate" yourself as a programmer? Especially, at Stanford since you are one out of so many.”

My first question is why you’d want to differentiate yourself as a programmer – that’s not something you necessarily need to do at this point. I’d focus a lot more on skill acquisition and on finding what makes you happy before worrying about this. There aren’t many times where you need to “stand out” of the crowd as a programmer, and most of them will arise because you’re competent, talented, and easy to work with.

Your personal identity doesn’t have to be tied to your coding skills. You’re a whole person and these skills are just a part of that.
“Why do you like the number 137 so much?”

It’s the reciprocal of the fine structure constant, rounded to the nearest integer. It’s also a great “nothing-up-my-sleeve” number.
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 $L_1 \cup L_2$?
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 \( \epsilon \) is the *identity element* for concatenation:
    \[
    w\epsilon = \epsilon w = w
    \]
  • 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

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

Let \( \Sigma = \{ \text{a, b, ..., z, A, B, ..., Z} \} \) 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
  
  $L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}$

- 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 $y \in L_2$?

**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, $w \in L_1L_2$.
- 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 $L_1L_2$
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 $L$.
  $$\{ \text{aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbaa, bbb} \}$$
- $LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$.
  $$\{ \text{aaaaaaaa, aaaaaaab, aaaaabaa, aaaaabb, aabaaaa, aabaab, aabbaa, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbaaaa, bbaab, bbbaa, bbbb} \}$$
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 \( L = \{ a, bb \} \), then \( L^* = \{ \)

\[ \varepsilon, \]

\[ a, bb, \]

\[ aa, abb, bba, bbbb, \]

\[ aaa, aabb, abba, abbbb, bbba, bbabb, bbbba, bbbbbbb, \]

\[ ... \]

\}
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).
- 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!