Regular Expressions
Regular Languages

• A language $L$ is a *regular language* if there is a DFA $D$ such that $\mathcal{L}(D) = L$.

• **Theorem:** The following are equivalent:
  • $L$ is a regular language.
  • There is a DFA for $L$.
  • There is an NFA for $L$. 

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**.
Regular Languages

- A language $L$ is a regular language if there is a DFA $D$ such that $\mathcal{L}(D) = L$.

- **Theorem:** The following are equivalent:
  - $L$ is a regular language.
  - There is a DFA for $L$.
  - There is an NFA for $L$. 


Regular Languages

- A language $L$ is a **regular language** if there is a DFA $D$ such that $\mathcal{L}(D) = L$.

- **Theorem:** The following are equivalent:
  - $L$ is a regular language.
  - There is a DFA for $L$.
  - There is an NFA for $L$.
  - $L$ can be formed from other known regular languages, using the closure properties.
Regular Languages

• A language $L$ is a **regular language** if there is a DFA $D$ such that $\mathcal{L}(D) = L$.

• *Theorem:* The following are equivalent:
  • $L$ is a regular language.
  • There is a DFA for $L$.
  • There is an NFA for $L$.
  • $L$ can be formed from other known regular languages, using the closure properties.

Today we will greatly expand on this last idea!
Constructing Regular Languages

• **Idea:** Build up all regular languages as follows:
  
  • Start with a small set of simple languages we already know to be regular.
  
  • Using closure properties, combine these simple languages together to form more elaborate languages.

• A *bottom-up approach to the regular languages.*
Constructing Regular Languages

- **Idea:** Build up all regular languages as follows:
  - Start with a small set of simple languages we already know to be regular.
  - Using closure properties, combine these simple languages to form more elaborate languages.
- A bottom-up approach to the regular languages.
Atomic Regular Expressions

- The regular expressions begin with three simple building blocks.
- The symbol $\emptyset$ is a regular expression that represents the empty language $\emptyset$.
- For any $a \in \Sigma$, the symbol $a$ is a regular expression for the language $\{a\}$.
- The symbol $\varepsilon$ is a regular expression that represents the language $\{\varepsilon\}$.
  - *Remember:* $\{\varepsilon\} \neq \emptyset!$
  - *Remember:* $\{\varepsilon\} \neq \varepsilon!$
Compound Regular Expressions

- If $R_1$ and $R_2$ are regular expressions, $R_1R_2$ is a regular expression for the concatenation of the languages of $R_1$ and $R_2$.
- If $R_1$ and $R_2$ are regular expressions, $R_1 \cup R_2$ is a regular expression for the union of the languages of $R_1$ and $R_2$.
- If $R$ is a regular expression, $R^*$ is a regular expression for the Kleene closure of the language of $R$.
- If $R$ is a regular expression, $(R)$ is a regular expression with the same meaning as $R$. 
**Operator Precedence**

- Here’s the operator precedence for regular expressions, from highest to lowest:
  
  $(R)$
  
  $R^*$
  
  $R_1R_2$
  
  $R_1 \cup R_2$

Consider the regular expression $ab^*c \cup d$

How many of the strings below are in the language described by this regular expression?

- $ababc$
- $abd$
- $ac$
- $abcd$

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then a number.
Operator Precedence

- Here’s the operator precedence for regular expressions, from highest to lowest:
  - $(R)$
  - $R^*$
  - $R_1R_2$
  - $R_1 \cup R_2$

Consider the regular expression $(a(b^*)c) \cup d$.

How many of the strings below are in the language described by this regular expression?

- $ababc$
- $abd$
- $ac$
- $abcd$

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then a number.
Operator Precedence

- Here’s the operator precedence for regular expressions, from highest to lowest:
  - $(R)$
  - $R^*$
  - $R_1R_2$
  - $R_1 \cup R_2$

Consider the regular expression $(a(b^*)c) \cup d$.

How many of the strings below are in the language described by this regular expression?

- $ababc$
- $abd$
- $ac$
- $abcd$

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then a number.
Regular Expression Examples

- The regular expression $\text{cat} \cup \text{dog}$ represents the regular language $\{ \text{cat, dog} \}$.
- The regular expression $\text{booo}^*$ represents the regular language $\{ \text{boo, booo, boooo, ... } \}$.
- The regular expression $(\text{candy!})^*$ represents the regular language $\{ \varepsilon, \text{candy!}, \text{candy!candy!}, \text{candy!candy!candy!}, ... \}$. 
Regular Expressions, Formally

- The **language of a regular expression** is the language described by that regular expression.

- Formally:
  - $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
  - $\mathcal{L}(\emptyset) = \emptyset$
  - $\mathcal{L}(a) = \{a\}$
  - $\mathcal{L}(R_1R_2) = \mathcal{L}(R_1) \mathcal{L}(R_2)$
  - $\mathcal{L}(R_1 \cup R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$
  - $\mathcal{L}(R^*) = \mathcal{L}(R)^*$
  - $\mathcal{L}((R)) = \mathcal{L}(R)$

**Worthwhile activity:** Apply this recursive definition to $a(b \cup c)((d))$ and see what you get.
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid w \text{ contains } aa \text{ as a substring} \}$. 
Designing Regular Expressions

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

\[(a \cup b)^*aa(a \cup b)^*\]
Designing Regular Expressions

• Let $\Sigma = \{a, b\}$.

• Let $L = \{ w \in \Sigma^* \mid w$ contains $aa$ as a substring $\}$.

$$\ (a \cup b)^* aa (a \cup b)^* \$$
Designing Regular Expressions

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

$$(a \cup b)^*aa(a \cup b)^*$$

bbabbbbaabab
aaaa
bbbbbbabbbbaabbbbb
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid w$ contains $aa$ as a substring $\}$.

$(a \cup b)^*aa(a \cup b)^*$

- $bbabbbbaabab$
- $aaaa$
- $bbbbbbbbbaabbbbbbb$
Designing Regular Expressions

• Let $\Sigma = \{a, b\}$.

• Let $L = \{ w \in \Sigma^* \mid w$ contains $aa$ as a substring $\}$.  

\[
\Sigma^*aa\Sigma^* \\
bbabbbbaaabab \\
aaaa \\
bbbbbbabbbbaabbbbbbb
\]
Designing Regular Expressions

• Let $\Sigma = \{\text{a, b}\}$.
• Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$. 
Designing Regular Expressions

Let $\Sigma = \{a, b\}$.

Let $L = \{w \in \Sigma^* \mid |w| = 4\}$.

The length of a string $w$ is denoted $|w|$.
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{w \in \Sigma^* \mid |w| = 4\}$. 
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$. 
Designing Regular Expressions

• Let $\Sigma = \{a, b\}$.

• Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$. 
Designing Regular Expressions

• Let $\Sigma = \{ a, b \}$.
• Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$.

\[
\Sigma \Sigma \Sigma \Sigma
\]

aaaa
baba
bbbbb
baaa
Designing Regular Expressions

• Let $\Sigma = \{a, b\}$.
• Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$. 

\[
\Sigma \Sigma \Sigma \Sigma
\]

aaaaa
baba
bbbb
baaa
Designing Regular Expressions

• Let $\Sigma = \{a, b\}$.
• Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$. 

$\Sigma^4$

aaaaa
baba
bbbb
baaa
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$. 

\[
\Sigma^4 \\
\text{aaaa} \\
\text{baba} \\
\text{babb} \\
\text{bbaa} \\
\]
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one $a \}$. 

Which of the following is a regular expression for $L$?

A. $\Sigma*a\Sigma*$
B. $b*ab* \cup b*$
C. $b*(a \cup \varepsilon)b*$
D. $b*a*b* \cup b*$
E. $b*(a* \cup \varepsilon)b*$
F. None of the above, or two or more of the above.

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then A, B, C, D, E, or F.
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* | w \text{ contains at most one } a \}$.

Which of the following is a regular expression for $L$?

A. $\Sigma^*a\Sigma^*$
B. $b^*ab^* \cup b^*$
C. $b^*(a \cup \varepsilon)b^*$
D. $b^*a*b^* \cup b^*$
E. $b^*(a^* \cup \varepsilon)b^*$
F. None of the above, or two or more of the above.

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then A, B, C, D, E, or F.
Designing Regular Expressions

• Let $\Sigma = \{a, b\}$.

• Let $L = \{ w \in \Sigma^* \mid w \text{ contains at most one } a \}$.

$$b^*(a \cup \varepsilon)b^*$$
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one $a \}$. 

$$b^* (a \cup \varepsilon) b^*$$
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* | w$ contains at most one $a \}$. 

$b^*(a \cup \varepsilon)b^*$

$bbbabbbb$
$bbbbbbbb$
$abbb$
$a$
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one $a \}$. 

$$b^*(a \cup \varepsilon)b^*$$

- $bbbbbabbbb$
- $bbbbbbbb$
- $abbbb$
- $a$
Designing Regular Expressions

- Let $\Sigma = \{a, b\}$.
- Let $L = \{ w \in \Sigma^* \mid w \text{ contains at most one } a \}$. 

$$b*a?b*$$ 

- $bbbbbabbb$
- $bbbbbbb$
- $abbb$
- $a$
A More Elaborate Design

• Let $\Sigma = \{ \, a, \, ., \, @ \, \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.
A More Elaborate Design

• Let $\Sigma = \{a, ., @\}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

  cs103@cs.stanford.edu
  first.middle.last@mail.site.org
  dot.at@dot.com
A More Elaborate Design

• Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

  cs103@cs.stanford.edu
  first.middle.last@mail.site.org
  dot.at@dot.com
A More Elaborate Design

• Let $\Sigma = \{ \texttt{a, ., @} \}$, where $\texttt{a}$ represents “some letter.”

• Let's make a regex for email addresses.

    \texttt{aa*}

    \texttt{cs103@cs.stanford.edu first.middle.last@mail.site.org dot.at@dot.com}
A More Elaborate Design

• Let $\Sigma = \{ \text{a, ., @} \}$, where a represents “some letter.”

• Let's make a regex for email addresses.

```
    aa*
```

  cs103@cs.stanford.edu
  first.middle.last@mail.site.org
  dot.at@dot.com
A More Elaborate Design

• Let $\Sigma = \{ a, \cdot, @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

  $aa*(.aa*)*$

  *cs103@cs.stanford.edu*
  *first.middle.last@mail.site.org*
  *dot.at@dot.com*
A More Elaborate Design

• Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

  $aa*(.aa*)*$

  cs103@cs.stanford.edu
  first.middle.last@mail.site.org
dot.at@dot.com
A More Elaborate Design

• Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

$$aa*(.aa*)*@$$

+ cs103@cs.stanford.edu
+ first.middle.last@mail.site.org
+ dot.at@dot.com
A More Elaborate Design

• Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

  \[ aa^*(.aa^*)*@ \]

  \[
  \text{cs103}@\text{cs.stanford.edu}
  \]

  \[
  \text{first.middle.last}@\text{mail.site.org}
  \]

  \[
  \text{dot.at}@\text{dot.com}
  \]
A More Elaborate Design

• Let $\Sigma = \{ \texttt{a, ., @} \}$, where \texttt{a} represents “some letter.”

• Let's make a regex for email addresses.

   $$\texttt{aa*\.aa*@aa*\.aa*}$$

   \texttt{cs103@cs.stanford.edu}

   \texttt{first.middle.last@mail.site.org}

   \texttt{dot.at@dot.com}
A More Elaborate Design

- Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”
- Let's make a regex for email addresses.

  $$aa*(.aa*)*@aa*..aa*$$

  cs103@cs.stanford.edu
  first.middle.last@mail.site.org
  dot.at@dot.com
A More Elaborate Design

- Let $\Sigma = \{ \text{a, ., @} \}$, where $\text{a}$ represents "some letter."
- Let's make a regex for email addresses.

```regex
aa*(.aa*)*@aa*.aa*(.aa*)*
```

- `cs103@cs.stanford.edu`
- `first.middle.last@mail.site.org`
- `dot.at@dot.com`
A More Elaborate Design

• Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

```
aa*(.aa*)*@aa*.aa*(.aa*)*
```

```
cs103@cs.stanford.edu
first.middle.last@mail.site.org
dot.at@dot.com
```
A More Elaborate Design

• Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

$$a^+ (\text{aa}^*) @ \text{aa}\.\text{aa}^* (\text{aa}^*)^*$$

- cs103@cs.stanford.edu
- first.middle.last@mail.site.org
- dot.at@dot.com
A More Elaborate Design

- Let $\Sigma = \{ \texttt{a, }, \texttt{, @} \}$, where \texttt{a} represents “some letter.”
- Let's make a regex for email addresses.

\[
a^+ (\texttt{.aa}* ) @ \texttt{aa* .aa* (\texttt{.aa* })* }
\]

- \texttt{cs103@cs.stanford.edu}
- \texttt{first.middle.last@mail.site.org}
- \texttt{dot.at@dot.com}
A More Elaborate Design

- Let $\Sigma = \{a, ., @\}$, where $a$ represents “some letter.”
- Let's make a regex for email addresses.

```
a^ (.*a^)* @ a^ .a^ (.*a^)*
```
A More Elaborate Design

- Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”
- Let's make a regex for email addresses.

\[
a^+ (.a^+)* @ a^+.a^+ (.a^+)*
\]

- cs103@cs.stanford.edu
- first.middle.last@mail.site.org
- dot.at@dot.com
A More Elaborate Design

- Let Σ = \{ a, ., @ \}, where a represents “some letter.”
- Let's make a regex for email addresses.

```
  a^ (a^)* @ a^ .a^ (.a^)*
```

- cs103@cs.stanford.edu
- first.middle.last@mail.site.org
- dot.at@dot.com
A More Elaborate Design

- Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”
- Let's make a regex for email addresses.

```
a^ (.a^)* @ a^ .a^ (.a^)*
```

cs103@cs.stanford.edu
first.middle.last@mail.site.org
dot.at@dot.com
A More Elaborate Design

- Let $\Sigma = \{ \text{a, ., @} \}$, where a represents “some letter.”
- Let's make a regex for email addresses.

```regex
a^ (a+)@ a^ ((a+)+)
```

- Examples:
  - `cs103@cs.stanford.edu`
  - `first.middle.last@mail.site.org`
  - `dot.at@dot.com`
A More Elaborate Design

• Let $\Sigma = \{a, \cdot, @\}$, where $a$ represents “some letter.”

• Let's make a regex for email addresses.

$$a^+ (a^+)* @ a^+ (a^+)^+$$

- $\text{cs103@cs.stanford.edu}$
- $\text{first.middle.last@mail.site.org}$
- $\text{dot.at@dot.com}$
A More Elaborate Design

- Let $\Sigma = \{ \text{a}, \text{.}, \text{@} \}$, where \text{a} represents “some letter.”

- Let's make a regex for email addresses.

  $a^+ (a+) ^* @ a^+ (a^+)^+$

  - \text{cs103}@cs.stanford.edu
  - \text{first.middle.last}@mail.site.org
  - \text{dot.at}@dot.com
For Comparison

\[ a^+ (\cdot a^+) * @ a^+ (\cdot a^+) ^+ \]
Shorthand Summary

- $R^n$ is shorthand for $RR \ldots R$ ($n$ times).
  - Edge case: define $R^0 = \varepsilon$.
- $\Sigma$ is shorthand for “any character in $\Sigma$.”
- $R?$ is shorthand for $(R \cup \varepsilon)$, meaning “zero or one copies of $R$."
- $R^+$ is shorthand for $RR^*$, meaning “one or more copies of $R$.”
Are DFAs (or, equivalently, NFAs) equivalent to RegExes?

• To show this, we would need to show:

  \[ L \text{ is Regular} \]

  \[ \iff \]

  You can write a Regular Expression for \( L \)

• Let’s think about each direction of this iff
The Power of Regular Expressions

**Theorem:** If $R$ is a regular expression, then $A(R)$ is regular.

**Proof idea:** Use induction!

- The atomic regular expressions all represent regular languages.
- The combination steps represent closure properties.
- So anything you can make from them must be regular!
Thompson’s Algorithm

• In practice, many regex matchers use an algorithm called *Thompson's algorithm* to convert regular expressions into NFAs (and, from there, to DFAs).
  - Read Sipser if you’re curious!

• **Fun fact:** the “Thompson” here is Ken Thompson, one of the co-inventors of Unix!
The Power of Regular Expressions

**Theorem:** If \( L \) is a regular language, then there is a regular expression for \( L \).

*This is not obvious!*

**Proof idea:** Pick an arbitrary NFA. Show how to convert that NFA into a regular expression.
Generalizing NFAs
Generalizing NFAs
Generalizing NFAs

These are all regular expressions!
Generalizing NFAs

\[
\begin{align*}
q_0 &\xrightarrow{ab \cup b} q_1 \\
q_2 &\xrightarrow{a} q_2 \\
q_2 &\xrightarrow{a*b?a*} q_3 \\
q_1 &\xrightarrow{ab*} q_1 \\
\text{start} &\xrightarrow{ab \cup b} q_1
\end{align*}
\]
Generalizing NFAs

Note: Actual NFAs aren't allowed to have transitions like these. This is just a thought experiment.
Generalizing NFAs

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

Input:

\[
\text{a a a b a a b b b b}
\]
Generalizing NFAs

![Diagram of a non-deterministic finite automaton (NFA)]

- Initial state: $q_0$
- Final states: $q_2$, $q_3$
- Transitions:
  - $q_0 \xrightarrow{ab \cup b} q_1$
  - $q_0 \xrightarrow{a} q_2$
  - $q_2 \xrightarrow{ab^*} q_1$
  - $q_2 \xrightarrow{a^*b?a^*} q_3$
  - $q_1 \xrightarrow{a} q_2$
  - $q_3 \xrightarrow{a} q_2$

Input:

```
| a | a | a | a | b | a | a | b | b | b |
```

This NFA recognizes languages over the alphabet {$a$, $b$}.
Generalizing NFAs

$$q_0 \xrightarrow{ab \cup b} q_1$$

$$q_2 \xrightarrow{a} q_2 \xrightarrow{ab^*} q_3$$

$$q_3 \xrightarrow{a*b?a^*} q_3$$
Generalizing NFAs

\[ q_0 \xrightarrow{ab \cup b} q_1 \]

\[ q_0 \xrightarrow{a} q_2 \quad q_2 \xrightarrow{ab^*} q_3 \]

\[ q_2 \xrightarrow{a*b?a*} q_3 \]

Input: a a a b a a b b b b
Generalizing NFAs
Generalizing NFAs
Generalizing NFAs

\[ q_0 \xrightarrow{ab \cup b} q_1 \]
\[ q_0 \xrightarrow{a} q_2 \quad q_2 \xrightarrow{ab^*} q_3 \]
\[ q_2 \xrightarrow{a*b?a*} q_3 \]

\( a, a, a, b, a, a, b, b, b \)
Generalizing NFAs

**Diagram:**

- **Start state:** $q_0$
- **Transitions:**
  - From $q_0$: $ab \cup b \rightarrow q_1$
  - From $q_2$: $a \rightarrow q_2$, $a*b?a* \rightarrow q_3$
  - From $q_1$: $b \rightarrow q_1$
  - From $q_3$: $ab* \rightarrow q_3$

**Input string:** a a a b a a b b b

**Explanation:**

The diagram illustrates a non-deterministic finite automaton (NFA) that accepts strings based on the rules defined by the transitions. The start state is $q_0$, and the automaton transitions through states $q_0$, $q_1$, $q_2$, and $q_3$ based on the input symbols. The automaton accepts strings that match the pattern defined by the transitions.
Generalizing NFAs

\[ q_0 \xrightarrow{ab \cup b} q_1 \]

\[ q_2 \xrightarrow{a} q_2 \]

\[ q_2 \xrightarrow{ab*} q_3 \]

\[ q_3 \xrightarrow{a*b?a*} q_3 \]

\[ \text{Strings accepted: } a a a a b b a a b b b b \]
Key Idea 1: Imagine that we can label transitions in an NFA with arbitrary regular expressions.
Generalizing NFAs

\[
q_0 \xrightarrow{ab \cup b} q_1
\]

start
Generalizing NFAs

Is there a simple regular expression for the language of this generalized NFA?
Generalizing NFAs

Is there a simple regular expression for the language of this generalized NFA?
Generalizing NFAs

\[ q_0 \xrightarrow{a^+ (a^+)^* @ a^+ (a^+)^+} q_1 \]
Is there a simple regular expression for the language of this generalized NFA?
Generalizing NFAs

Is there a simple regular expression for the language of this generalized NFA?
**Key Idea 2:** If we can convert an NFA into a generalized NFA that looks like this...

...then we can easily read off a regular expression for the original NFA.
From NFAs to Regular Expressions
From NFAs to Regular Expressions

Here, $R_{11}$, $R_{12}$, $R_{21}$, and $R_{22}$ are arbitrary regular expressions.
Question: Can we get a clean regular expression from this NFA?
From NFAs to Regular Expressions

Key Idea 3: Somehow transform this NFA so that it looks like this:

```
q₀  some-regex  q₁
```

```text
Key Idea 3: Somehow transform this NFA so that it looks like this:
```
From NFAs to Regular Expressions

The first step is going to be a bit weird...
From NFAs to Regular Expressions
From NFAs to Regular Expressions

\[
\begin{align*}
q_s & \xrightarrow{\varepsilon} q_1 \\
q_1 & \xrightarrow{R_{11}} q_1 \\
q_1 & \xrightarrow{R_{12}} q_2 \\
q_2 & \xrightarrow{R_{21}} q_2 \\
q_2 & \xrightarrow{R_{22}} q_2 \\
q_2 & \xrightarrow{\varepsilon} q_f \\
q_f & \xrightarrow{\varepsilon} q_f
\end{align*}
\]
From NFAs to Regular Expressions

**Step 1:** Add new start state and new accept state, connect them to old start/accept states via epsilon edges, “demote” old accept states.
From NFAs to Regular Expressions

This NFA got bigger, not simpler as was our goal. :-(

Key Idea 3: Somehow transform this NFA so that it looks like this:

\[ \text{start} \rightarrow q_0 \xrightarrow{\text{some-regex}} q_1 \]
Step 2: Pick one state at a time, and vote it off the island (i.e., remove it from the NFA), until we reach the goal.
From NFAs to Regular Expressions

How could we eliminate this state from the NFA without its changing behavior?
From NFAs to Regular Expressions
From NFAs to Regular Expressions
From NFAs to Regular Expressions

Note: We're using concatenation and Kleene closure in order to skip this state.
From NFAs to Regular Expressions

\[ \varepsilon R_{11} \ast R_{12} \]
From NFAs to Regular Expressions

\[ \varepsilon R_{11} \ast R_{12} \]

\[
\begin{array}{ccl}
\text{start} & \rightarrow & q_s \\
\varepsilon & \rightarrow & q_1 \\
R_{11} & \rightarrow & q_2 \\
R_{12} & \rightarrow & q_f
\end{array}
\]
From NFAs to Regular Expressions

\[ \varepsilon R_{11} \ast R_{12} \]
What regex should go on this edge?

A. $R_{12} R_{21}$  
B. $R_{12} R_{22}^* R_{21}$  
C. $R_{21} R_{12}$  
D. $R_{21} R_{11}^* R_{12}$

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then A, B, C, or D.
From NFAs to Regular Expressions

\[ \varepsilon R_{11} \ast R_{12} \]

\[ R_{21} \ast R_{11} \ast R_{12} \]
From NFAs to Regular Expressions

\[
\begin{align*}
q_s & \xrightarrow{\varepsilon} R_{11} \varepsilon R_{12} \\
q_s & \xrightarrow{R_{11}} q_1 \xrightarrow{R_{21}} q_2 \xrightarrow{R_{22}} q_f \\
q_1 & \xrightarrow{R_{12}} q_2 \\
q_2 & \xrightarrow{R_{11} \varepsilon R_{12}} q_f \\
q_f & \xrightarrow{\varepsilon R_{11} \varepsilon R_{12}} q_f
\end{align*}
\]
From NFAs to Regular Expressions

\[ \varepsilon R_{11} * R_{12} \]

\[ R_{21} R_{11} * R_{12} \]
From NFAs to Regular Expressions

\[ R_{11} \ast R_{12} \]

\[ R_{21} \ast R_{11} \ast R_{12} \]
From NFAs to Regular Expressions

Note: We're using **union** to combine these transitions together.
From NFAs to Regular Expressions

\[ R_{11} \ast R_{12} \]

\[ R_{22} \cup R_{21} R_{11} \ast R_{12} \]
From NFAs to Regular Expressions

\[ R_{22} \cup R_{21} R_{11} \ast R_{12} \]
From NFAs to Regular Expressions

\[
R_{11} \ast R_{12} \cup R_{22} \cup R_{21} R_{11} \ast R_{12}
\]
From NFAs to Regular Expressions

\[
R_{22} \cup R_{21} R_{11} * R_{12}
\]
From NFAs to Regular Expressions

\[ R_{11}^* R_{12} \ (R_{22} \cup R_{21} R_{11}^* R_{12})^* \varepsilon \]
From NFAs to Regular Expressions

\[ R_{11} \ast R_{12} (R_{22} \cup R_{21} R_{11} \ast R_{12}) \ast \varepsilon \]

\[ R_{22} \cup R_{21} R_{11} \ast R_{12} \]
From NFAs to Regular Expressions

\[ R_{11} \ast R_{12} (R_{22} \cup R_{21} R_{11} \ast R_{12}) \ast \varepsilon \]
From NFAs to Regular Expressions

\[ R_{11} \ast R_{12} (R_{22} \cup R_{21} R_{11} \ast R_{12})^\ast \]
From NFAs to Regular Expressions

\[ R_{11}^* R_{12} \left( R_{22} \cup R_{21} R_{11}^* R_{12} \right)^* \]

Key Idea 3: Somehow transform this NFA so that it looks like this:

\[ \text{start} \rightarrow q_0 \xrightarrow{\text{some-regex}} q_1 \]
The Construction at a Glance

• Start with an NFA $N$ for the language $L$.
• Add a new start state $q_s$ and accept state $q_f$ to the NFA.
  • Add an $\varepsilon$-transition from $q_s$ to the old start state of $N$.
  • Add $\varepsilon$-transitions from each accepting state of $N$ to $q_f$, then mark them as not accepting.
• Repeatedly remove states other than $q_s$ and $q_f$ from the NFA by “shortcutting” them until only two states remain: $q_s$ and $q_f$.
• The transition from $q_s$ to $q_f$ is then a regular expression for the NFA.
Eliminating a State

- To eliminate a state \( q \) from the automaton, do the following for each pair of states \( q_0 \) and \( q_1 \), where there's a transition from \( q_0 \) into \( q \) and a transition from \( q \) into \( q_1 \):
  - Let \( R_{in} \) be the regex on the transition from \( q_0 \) to \( q \).
  - Let \( R_{out} \) be the regex on the transition from \( q \) to \( q_1 \).
  - If there is a regular expression \( R_{stay} \) on a transition from \( q \) to itself, add a new transition from \( q_0 \) to \( q_1 \) labeled \(((R_{in})(R_{stay})^*)(R_{out}))\).
    - If there isn't, add a new transition from \( q_0 \) to \( q_1 \) labeled \(((R_{in})(R_{out}))\).
  - If a pair of states has multiple transitions between them labeled \( R_1, R_2, \ldots, R_k \), replace them with a single transition labeled \( R_1 \cup R_2 \cup \ldots \cup R_k \).
Our Transformations

- DFA
- NFA
- Regexp

- is already one
- state elimination
- subset construction
- Thompson's algorithm
**Theorem:** The following are all equivalent:

- $L$ is a regular language.
- There is a DFA $D$ such that $\mathcal{L}(D) = L$.
- There is an NFA $N$ such that $\mathcal{L}(N) = L$.
- There is a regular expression $R$ such that $\mathcal{L}(R) = L$. 
Why This Matters

• The equivalence of regular expressions and finite automata has practical relevance.
  • Tools like grep and flex that use regular expressions capture all the power available via DFAs and NFAs.

• This also is hugely theoretically significant: the regular languages can be assembled “from scratch” using a small number of operations!
Next Time

- **Applications of Regular Languages**
  - Answering “so what?”
- **Intuiting Regular Languages**
  - What makes a language regular?
- **The Myhill-Nerode Theorem**
  - The limits of regular languages.