DFAs, Informally

- 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 the alphabet.
  - This is the “deterministic” part of DFA.
- There is a unique start state.
- There may be multiple accepting states.
Recognizing Languages with DFAs

$$L = \{ \text{w} \mid \text{w contains 00 as a substring} \}$$
Recognizing Languages with DFAs

$L = \{ w \mid \text{every even character of } w \text{ is a } 0 \}$
Tabular DFAs

<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>
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where

  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta\) is the transition function.
  - \(q_0\) is the start state.
  - \(F\) is a set of accepting states.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.

\(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the \textit{transition function}.
  - \(q_0\) is the start state.
  - \(F\) is a set of accepting states.

\(\delta\) is a function, so there must be exactly one transition defined for each state/symbol pair.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
A Formal Definition of DFAs

- Formally, a DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
A Formal Definition of Acceptance

- Given a DFA \((Q, \Sigma, \delta, q_0, F)\), we want to find some way to formally define what it means for the DFA to accept a string \(w \in \Sigma^*\).

- **Idea:** Define a function \(\delta^* : \Sigma^* \rightarrow Q\) that says what state we end up in if we run the DFA on a given string.

- This function represents the effect of running the computer on a given input.
A Formal Definition of Acceptance

- **Notation**: If $w$ is a string and $a$ is a character, then $wa$ is the string formed by appending $a$ to $w$.
- Given a DFA $(Q, \Sigma, \delta, q_0, F)$, $\delta^*$ is defined recursively.
  - $\delta^*(\varepsilon) = q_0$
    - Running the automaton on $\varepsilon$ ends in the start state.
  - $\delta^*(wa) = \delta(\delta^*(w), a)$
    - Running on $wa$ is equal to running the automaton on $w$, then following the transition for $a$. 
A Formal Definition of Acceptance

- Using our $\delta^*$ function, we can formally define the language of a DFA.
- Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
- Define $\mathcal{L}(D) = \{ w \mid \delta^*(w) \in F \}$
  - The set of strings $w$ that cause the DFA to end up in an accepting state.
So What?

- We now have a mathematically rigorous way of defining whether a DFA accepts a string.
- We can try making changes to DFAs and can formally prove how those changes transform the language of the DFA.
A language $L$ is called a **regular language** iff there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
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 not in $L$.

• Formally:

$$\overline{L} = \{ w \mid w \in \Sigma^* \land w \notin L \}$$
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 not in $L$.
- Formally:
  \[
  \overline{L} = \{ w \mid w \in \Sigma^* \land w \notin L \}
  \]
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 not in $L$.

- Formally:

$$\overline{L} = \Sigma^* - L$$
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 not in $L$.

- Formally:

$$\overline{L} = \Sigma^* - L$$
Complementing Regular Languages

- Recall: A **regular language** is a language accepted by some DFA.

- **Question**: If $L$ is a regular language, is $\overline{L}$ a regular language?

- If the answer is “yes,” then there must be some way to construct a DFA for $\overline{L}$.

- If the answer is “no,” then some language $L$ can be accepted by a DFA, but $\overline{L}$ cannot be accepted by any DFA.
Complementing Regular Languages

$L = \{ w \mid w \text{ contains } 00 \text{ as a substring} \}$
Complementing Regular Languages

\[ L = \{ \text{w | w contains 00 as a substring} \} \]

\[ \overline{L} = \{ \text{w | w does not contain 00 as a substring} \} \]
Complementing Regular Languages

$L = \{ \text{ w } \mid \text{ w contains 00 as a substring } \}$

$L = \{ \text{ w } \mid \text{ w does not contain 00 as a substring } \}$
Complementing Regular Languages

\[ L = \{ w \mid w \text{ is a legal email address} \} \]
Complementing Regular Languages

\[ \overline{L} = \{ w \mid w \text{ is not a legal email address} \} \]
Complementing Regular Languages

\[ L = \{ w \mid w \text{ is not a legal email address} \} \]
Constructions on Automata

- Much of our discussion of automata will consider constructions that transform one automaton into another.
- Exchanging accepting and rejecting states is a simple construction sometimes called the complement construction.
- Does this construction always work?
- How would we prove it?
Theorem: If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $L(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $L(D')$. 

Proof: By definition, $L(D') = \{ w | \delta^*(w) \notin Q - F \}$.

So $\in \subseteq L(D') \iff \delta^*(w) \in Q \land \delta^*(w) \notin F$.

$\in L(D') = \{ w | \delta^*(w) \in Q \} - \{ w | \delta^*(w) \in F \}$.

Since $\delta^*(w) : \Sigma^* \to Q$, any string $w$ satisfies $\delta^*(w) \in Q$.

Thus $\delta^*(w) \in Q$ means that $w \in \Sigma^*$.

So $L(D') = \{ w | w \in \Sigma^* \} - \{ w | \delta^*(w) \in F \}$.

$L(D') = \Sigma^* - L(D)$. ■
Theorem: If D = (Q, Σ, δ, q₀, F) is a DFA with language L(D), then the DFA D' = (Q, Σ, δ, q₀, Q – F) has language L(D).

Proof: By definition, L(D') = \{ w | δ*(w) ∈ Q – F \}. 

Since δ*(w) : Σ* → Q, any string w satisfies δ*(w) ∈ Q. Thus δ*(w) ∈ Q means that w ∈ Σ*. So L(D') = Σ* – L(D).
Theorem: If \( D = (Q, \Sigma, \delta, q_0, F) \) is a DFA with language \( L(D) \), then the DFA \( D' = (Q, \Sigma, \delta, q_0, Q - F) \) has language \( L(D) \).

Proof: By definition, \( L(D') = \{ w \mid \delta^*(w) \in Q - F \} \). So

\[
L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \}
\]
Theorem: If \( D = (Q, \Sigma, \delta, q_0, F) \) is a DFA with language \( L(D) \), then the DFA \( D' = (Q, \Sigma, \delta, q_0, Q - F) \) has language \( L(D) \).

Proof: By definition, \( L(D') = \{ w \mid \delta^*(w) \in Q - F \} \). So

\[
L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \}
\]

\[
L(D') = \{ w \mid \delta^*(w) \in Q \} - \{ w \mid \delta^*(w) \in F \}
\]
Theorem: If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $L(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $L(D)$.

Proof: By definition, $L(D') = \{ w \mid \delta^*(w) \in Q - F \}$. So

$$L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \}$$

$$L(D') = \{ w \mid \delta^*(w) \in Q \} - \{ w \mid \delta^*(w) \in F \}$$

Since $\delta^* : \Sigma^* \rightarrow Q$, any string $w$ satisfies $\delta^*(w) \in Q$. 

Theorem: If \( D = (Q, \Sigma, \delta, q_0, F) \) is a DFA with language \( L(D) \), then the DFA \( D' = (Q, \Sigma, \delta, q_0, Q - F) \) has language \( L(D) \).

Proof: By definition, \( L(D') = \{ w \mid \delta^*(w) \in Q - F \} \). So

\[
L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \}
\]

Thus \( \delta^*(w) \in Q \) means that \( w \in \Sigma^* \).
**Theorem:** If \( D = (Q, \Sigma, \delta, q_0, F) \) is a DFA with language \( L(D) \), then the DFA \( D' = (Q, \Sigma, \delta, q_0, Q - F) \) has language \( L(D) \).

**Proof:** By definition, \( L(D') = \{ w \mid \delta^*(w) \in Q - F \} \). So

\[
L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \} = \{ w \mid \delta^*(w) \in Q \} - \{ w \mid \delta^*(w) \in F \}
\]

Since \( \delta^* : \Sigma^* \rightarrow Q \), any string \( w \) satisfies \( \delta^*(w) \in Q \). Thus \( \delta^*(w) \in Q \) means that \( w \in \Sigma^* \). So

\[
L(D') = \{ w \mid w \in \Sigma^* \} - \{ w \mid \delta^*(w) \in F \}
\]
Theorem: If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $L(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $\overline{L(D)}$.

Proof: By definition, $L(D') = \{ w \mid \delta^*(w) \in Q - F \}$. So

\[
L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \} \\
L(D') = \{ w \mid \delta^*(w) \in Q \} - \{ w \mid \delta^*(w) \in F \}
\]

Since $\delta^* : \Sigma^* \to Q$, any string $w$ satisfies $\delta^*(w) \in Q$. Thus $\delta^*(w) \in Q$ means that $w \in \Sigma^*$. So

\[
L(D') = \{ w \mid w \in \Sigma^* \} - \{ w \mid \delta^*(w) \in F \} \\
L(D') = \Sigma^* - \{ w \mid \delta^*(w) \in F \}
\]
Theorem: If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $L(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $L(D)$.

Proof: By definition, $L(D') = \{ w \mid \delta^*(w) \in Q - F \}$. So

$L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \}$
$L(D') = \{ w \mid \delta^*(w) \in Q \} - \{ w \mid \delta^*(w) \in F \}$

Since $\delta^* : \Sigma^* \to Q$, any string $w$ satisfies $\delta^*(w) \in Q$. Thus $\delta^*(w) \in Q$ means that $w \in \Sigma^*$. So

$L(D') = \{ w \mid w \in \Sigma^* \} - \{ w \mid \delta^*(w) \in F \}$
$L(D') = \Sigma^* - \{ w \mid \delta^*(w) \in F \}$
$L(D') = \Sigma^* - L(D)$
**Theorem:** If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $L(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $L(D)$.

**Proof:** By definition, $L(D') = \{ w \mid \delta^*(w) \in Q - F \}$. So

$$L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \not\in F \}$$

$$L(D') = \{ w \mid \delta^*(w) \in Q \} - \{ w \mid \delta^*(w) \in F \}$$

Since $\delta^*: \Sigma^* \rightarrow Q$, any string $w$ satisfies $\delta^*(w) \in Q$. Thus $\delta^*(w) \in Q$ means that $w \in \Sigma^*$. So

$$L(D') = \{ w \mid w \in \Sigma^* \} - \{ w \mid \delta^*(w) \in F \}$$

$$L(D') = \Sigma^* - \{ w \mid \delta^*(w) \in F \}$$

$$L(D') = \Sigma^* - L(D)$$

$$L(D') = L(D).$$
**Theorem:** If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA with language $L(D)$, then the DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$ has language $\overline{L(D)}$.

**Proof:** By definition, $L(D') = \{ w \mid \delta^*(w) \in Q - F \}$. So

\[
L(D') = \{ w \mid \delta^*(w) \in Q \land \delta^*(w) \notin F \}
\]

Since $\delta^* : \Sigma^* \to Q$, any string $w$ satisfies $\delta^*(w) \in Q$. Thus $\delta^*(w) \in Q$ means that $w \in \Sigma^*$. So

\[
L(D') = \{ w \mid w \in \Sigma^* \} - \{ w \mid \delta^*(w) \in F \}
\]

\[
L(D') = \Sigma^* - \{ w \mid \delta^*(w) \in F \}
\]

\[
L(D') = \Sigma^* - L(D)
\]

\[
L(D') = \overline{L(D)}. \quad \blacksquare
\]
Closure Properties

• If $L$ is a regular language, $\overline{L}$ is a regular language.

• If we begin with a regular language and complement it, we end up with a regular language.

• This is an example of a closure property of regular languages.
  • The regular languages are closed under complementation.
  • We'll see more such properties later on.
NFAs
NFAs

• An **NFA** is a
  • **Nondeterministic**
  • **Finite**
  • **Automaton**

• Conceptually similar to a DFA, but equipped with the vast power of **nondeterminism**.
(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.
A Simple NFA

The diagram represents a non-deterministic finite automaton (NFA) with the following states:

- **Start state**: $q_0$
- **Final states**: $q_2$, $q_3$

Transitions:
- From $q_0$ to $q_1$: on input 1
- From $q_1$ to $q_2$: on input 1
- From $q_0$ to $q_3$: on input 0, 1
- From $q_2$ to $q_0$: on input 0, 1
- From $q_3$ to $q_2$: on input 0, 1
A Simple NFA

$q_0$ has two transitions defined on 1!
A Simple NFA
A Simple NFA

0 1 0 1 1
A Simple NFA

start

$q_0$ 1 $q_1$

$q_1$ 1 $q_2$

$q_2$

$q_3$

0, 1

0, 1

0, 1

0, 1

0, 1

0, 1

0 1 0 1 1
A Simple NFA
A Simple NFA

start

$q_0$ → 1 → $q_1$ → 1 → $q_2$

$q_0$ (0, 1)

$q_1$ (0, 1)

$q_2$

$q_3$ (0, 1)

0 1 0 1 1
A Simple NFA

\[
\text{start} \quad q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{0, 1} q_3 \xrightarrow{0, 1} q_2
\]

Input sequence: 0 1 0 1 1
A Simple NFA

start

$q_0$ 1 $q_1$

0, 1

$q_1$ 1 $q_2$

$q_2$

$q_3$

0, 1

0, 1

0, 1

0 1 0 1 1 1
A Simple NFA
A Simple NFA

0 1 0 1 1
A Simple NFA
A Simple NFA

start

$q_0$ 1 1

$q_1$

$q_2$

$q_3$

$0, 1$

$0, 1$

$0, 1$

$0$

$1$

$0$

$1$

$1$

$1$

0 1 0 1 1
A Simple NFA

Start

$q_0$ -> 1 -> $q_1$ -> 1 -> $q_2$

$q_3$

Input: 010111
A Simple NFA

start

$q_0$ \[\xrightarrow{1} q_1\]

$q_1$ \[\xrightarrow{1} q_2\]

$q_2$ \[\xrightarrow{0} q_3\]

$q_3$ \[\xrightarrow{0} \text{accept}\]

Input: 0 1 0 1 1
A Simple NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

<table>
<thead>
<tr>
<th>State</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>0, 1</td>
</tr>
<tr>
<td>q_1</td>
<td>0, 1</td>
</tr>
<tr>
<td>q_2</td>
<td></td>
</tr>
<tr>
<td>q_3</td>
<td>0, 1</td>
</tr>
</tbody>
</table>

Input string: 0 1 0 1 1 1
A Simple NFA

\[
\begin{array}{c}
\text{start} \\
q_0 \quad 1 \quad q_1 \\
\quad 0, 1 \quad q_3 \\
\quad 0 \\
q_2 \\
\quad 0, 1 \\
\end{array}
\]
A Simple NFA

start

\( q_0 \) 1 \( q_1 \) 1 \( q_2 \)

0, 1

0

0, 1

0, 1

0, 1

0 1 0 1 1
A Simple NFA

- Start state: $q_0$
- Transitions:
  - From $q_0$: 1 to $q_1$, 0, 1 to $q_3$
  - From $q_1$: 1 to $q_2$
  - From $q_3$: 0 to $q_1$, 0, 1 to $q_2$, 0, 1 to $q_3$

Input sequence: 0 1 0 1 1
A Simple NFA

\[ q_0 \xrightarrow{0,1} q_1 \xrightarrow{1} q_2 \xrightarrow{0,1} q_3 \xrightarrow{0,1} q_2 \]

Input sequence: 0 1 0 1 1
A Simple NFA
A Simple NFA

- Start state: \( q_0 \)
- Transitions:
  - From \( q_0 \): 1 to \( q_1 \)
  - From \( q_1 \): 1 to \( q_2 \)
  - From \( q_2 \): 0, 1 to \( q_3 \)
  - From \( q_3 \): 0, 1 to \( q_3 \)

Input string: 0 1 0 1 1 1
A Simple NFA

0 1 0 1 1 1
A Simple NFA

[Diagram of a non-deterministic finite automaton (NFA) with states labeled q₀, q₁, q₂, q₃ and transitions labeled '0', '1', '0, 1']

Input sequence: 0 1 0 1 1 1
A Simple NFA

Transition Table:

<table>
<thead>
<tr>
<th>State</th>
<th>Input 0</th>
<th>Input 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>0, 1</td>
<td>1</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$q_2$</td>
<td>0, 1</td>
<td>0, 1</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0, 1</td>
<td>0, 1</td>
</tr>
</tbody>
</table>

Input String: 0 1 0 1 1 1
A Simple NFA

start

$q_0$ → 1 → $q_1$ → 1 → $q_2$

0, 1

$q_0$ → 0, 1

$q_1$ → 0

$q_3$ → 0, 1

$q_2$ → 0, 1

0 1 0 1 1 1
A Simple NFA

start → \( q_0 \) on 1 → \( q_1 \) on 1 → \( q_2 \)

\( q_0 \) on 0, 1

\( q_1 \) on 0

\( q_3 \) on 0, 1

\( q_2 \) on 0, 1

0 1 0 1 1
A Simple NFA

start

$q_0$ 1 $q_1$

$q_1$ 1 $q_2$

$q_2$

$q_3$ 0, 1

$q_3$ 0, 1

$q_3$ 0, 1
A More Complex NFA
A More Complex NFA

These states don’t have transitions defined on all symbols!
A More Complex NFA

These states don’t have transitions defined on all symbols!

If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.
A More Complex NFA
A More Complex NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

0 1 0 1 1
A More Complex NFA
A More Complex NFA

\[ \begin{align*}
\text{start} & \quad \rightarrow \quad q_0 \quad \rightarrow \quad q_1 \quad \rightarrow \quad q_2 \\
0, 1 & \quad \rightarrow \quad 1 & \quad 1 \\
\end{align*} \]
A More Complex NFA

The diagram shows a non-deterministic finite automaton (NFA) with states $q_0$, $q_1$, and $q_2$. The transitions are:

- From $q_0$ on input 1 to $q_1$.
- From $q_0$ on input 0 or 1 to $q_0$.
- From $q_1$ on input 1 to $q_2$.
- From $q_2$ there is a loop back to $q_2$.

The input sequence $010111$ is shown on the diagram, indicating how the NFA processes the input.
A More Complex NFA
A More Complex NFA

Oh no! There's no transition defined!
A More Complex NFA
A More Complex NFA

start → $q_0$ (0, 1) → $q_1$ (1) → $q_2$
A More Complex NFA

```
0 1 0 1 1
```
A More Complex NFA

```
0 1 0 1 1
```
A More Complex NFA

0 1 0 1 1
A More Complex NFA

0 1 0 1 1
A More Complex NFA

start

$q_0$ 1 $q_1$

0, 1

1 $q_2$

0 1 0 1 1
A More Complex NFA

start

$q_0$ 1 $q_1$ 1 $q_2$

0, 1
A More Complex NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

0 1 1 0 1 1
A More Complex NFA

0 1 0 1 1
A More Complex NFA
Intuiting Nondeterminism

- Nondeterministic machines are a serious departure from physical computers.
- How can we build up an intuition for them?
- Three approaches:
  - Tree computation
  - Perfect guessing
  - Massive parallelism
Tree Computation
Tree Computation

0 1 0 1 0
Tree Computation

0 1 0 1 1 0
Tree Computation

[start] $q_0$ → 0 → $q_1$ → 1 → $q_2$

$1$ → $q_3$ → 0 → $q_4$ → 0 → $q_5$

$0, 1$ → $q_2$
Tree Computation

0 1 0 1 0 1 0
Tree Computation

0 1 0 1 0 1 0
Tree Computation

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_4 \rightarrow q_5 \]

0, 1

0, 1

0, 1
Tree Computation

0 1 0 1 0 1 0

start

q_0 → 0 → q_1 → 1 → q_2

q_0 → 1 → q_3

q_3 → 0 → q_4 → 0 → q_5

q_1 → 1 → q_4

q_4 → 0 → q_5

q_5

q_4

q_2
Tree Computation

0 1 0 1 1 0
Tree Computation

0 1 0 1 0 0
Tree Computation

start

q₀ → 0 → q₁ → 1 → q₂

1 → q₃ → 0 → q₄ → 0 → q₅

0, 1

0 1 0 1 0

q₀ → q₁

q₄ → q₅

q₄ → q₂

q₄

q₄
Tree Computation

0 1 0 1 1 0
Tree Computation

Start

$q_0 \rightarrow 0 \rightarrow q_1 \rightarrow 1 \rightarrow q_2$

$q_3 \rightarrow 1 \rightarrow q_4 \rightarrow 1 \rightarrow q_5$

$0, 1 \rightarrow q_4 \rightarrow 0, 1 \rightarrow q_5$

0 1 0 1 0 0

$q_0 \rightarrow q_1 \rightarrow q_4 \rightarrow q_5 \rightarrow q_4 \rightarrow q_5$

$q_2 \rightarrow q_4 \rightarrow q_5$
Tree Computation

\begin{itemize}
\item \textbf{start: } $q_0$
\item $q_0 \xrightarrow{0} q_1$
\item $q_1 \xrightarrow{1} q_2$
\item $q_2 \xrightarrow{1} q_3$
\item $q_3 \xrightarrow{0} q_4$
\item $q_4 \xrightarrow{0} q_5$
\item $q_5 \xrightarrow{0,1} q_4$
\item \textbf{Input: } 0 1 0 1 0 0
\end{itemize}
Tree Computation
Nondeterminism as a Tree

- At each decision point, the automaton clones itself for each possible decision.
- The series of choices forms a directed, rooted tree.
- At the end, if any active accepting states remain, we accept.
Perfect Guessing

\[ q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \]
\[ q_0 \xrightarrow{1} q_3 \]
\[ q_3 \xrightarrow{0} q_4 \xrightarrow{0} q_5 \]
\[ q_1 \xrightarrow{1} q_4 \]
\[ q_2 \]

\[ q_4 \xrightarrow{0, 1} q_5 \]
Perfect Guessing

- States: $q_0, q_1, q_2, q_3, q_4, q_5$
- Transitions:
  - $q_0$ to $q_1$: 0
  - $q_0$ to $q_3$: 1
  - $q_1$ to $q_2$: 1
  - $q_1$ to $q_4$: 1
  - $q_3$ to $q_4$: 0
  - $q_4$ to $q_5$: 0
  - $q_5$ (loop)
- Initial state: $q_0$
- Accept states: $q_5$

Input sequence: 010110
Perfect Guessing

```
0 1 0 1 1 0
```

\( q_0 \rightarrow q_1 \rightarrow q_2 \)

\( q_3 \rightarrow q_4 \rightarrow q_5 \)
Perfect Guessing

0 1 0 1 0 0 1 0
Perfect Guessing

start

$q_0$ 0 $q_1$ 1 $q_2$

1

$q_3$ 0 $q_4$ 0

1

$q_4$ 0 $q_5$

0, 1

0 1 0 1 0
Perfect Guessing

0 1 0 1 0
Perfect Guessing

\[0, 1\]
Perfect Guessing

```
q_0  0 -> q_1  1 -> q_2
   1   1   
q_3  0 -> q_4  0 -> q_5
   1   0, 1

0 1 0 1 1 0
```
Perfect Guessing

1. $q_0$ -> $q_1$ on 0
2. $q_0$ -> $q_3$ on 1
3. $q_1$ -> $q_4$ on 1
4. $q_4$ -> $q_5$ on 0
5. $q_3$ -> $q_4$ on 0
6. $q_4$ -> $q_5$ on 0, 1
7. Start state: $q_0$
8. Final state: $q_5$
9. Input sequence: 0 1 0 1 0
Perfect Guessing

- Start state: $q_0$
- $q_0$ transitions to $q_1$ on input 0 and $q_2$ on input 1
- $q_1$ transitions to $q_2$ on input 1
- $q_2$ is an accept state
- $q_3$ transitions to $q_4$ on input 1
- $q_3$ transitions to $q_3$ on input 0 or 1
- $q_4$ transitions to $q_5$ on input 0
- $q_5$ is an accept state

Input sequence: 0 1 0 1 0 1 0
Perfect Guessing

0 1 0 1 0
Perfect Guessing

- We can view nondeterministic machines as having magic superpowers that enable them to guess the correct choice of moves to make.
- Idea: Machine can always guess the right choice if one exists.
- No physical analog for something of this sort.
  - (Those of you thinking quantum computing – nondeterminism is more powerful than quantum computation.)
Massive Parallelism

0 1 0 1 0 0
Massive Parallelism
Massive Parallelism

0 1 0 1 0 0
Massive Parallelism

Start

$q_0 \rightarrow 0 \rightarrow q_1 \rightarrow 1 \rightarrow q_2$

$q_3 \rightarrow 0 \rightarrow q_4 \rightarrow 0 \rightarrow q_5$

$q_1 \rightarrow 1 \rightarrow q_2$

$q_4 \rightarrow 0 \rightarrow q_5$

$q_0 \rightarrow 1 \rightarrow q_3$

$q_4 \rightarrow 0, 1 \rightarrow q_4$

Input: 0 1 0 1 1 0
Massive Parallelism

```
0 1 0 1 1 0
```

Diagram:

- Start at $q_0$
- Transition to $q_1$ on input 0
- Transition to $q_2$ on input 1
- $q_2$ loops back on 0,1
- Transition to $q_3$ on input 1
- Transition to $q_4$ on input 0
- Transition to $q_5$ on input 0
- $q_5$ loops back on 0,1
Massive Parallelism

start

$q_0$ → 0 → $q_1$  
$1$ → $q_3$  
$0$ → $q_4$  
$0$ → $q_5$

$q_1$  
$1$ → $q_2$

$q_3$

$q_4$

$q_2$

$q_5$
Massive Parallelism

0 1 0 1 0 1 0
Massive Parallelism

Start

$q_0$ → 0 → $q_1$ → 1 → $q_2$

$q_3$ → 1 → $q_4$ → 1 → $q_5$

$q_4$ also transitions to $q_5$ on inputs 0, 1.

Input sequence: 0 1 0 1 0 0
Massive Parallelism

- From start to $q_0$: 0
- From $q_0$ to $q_1$: 1
- From $q_1$ to $q_2$: 1
- From $q_3$ to $q_4$: 1
- From $q_4$ to $q_5$: 0
- The sequence of symbols: 0 1 0 1 0 0
Massive Parallelism

0 1 0 1 0 0

Diagram:

- Start state: $q_0$
- Transitions:
  - $q_0$ to $q_1$: 0
  - $q_1$ to $q_2$: 1
  - $q_0$ to $q_3$: 1
  - $q_3$ to $q_4$: 0
  - $q_4$ to $q_5$: 0
  - $q_4$ self-loop on 0, 1
  - $q_2$ self-loop

States:
- $q_0$
- $q_1$
- $q_2$
- $q_3$
- $q_4$
- $q_5$
Massive Parallelism

Diagram:

- Start at $q_0$
- $q_0$ to $q_1$: 0
- $q_1$ to $q_2$: 1
- $q_2$ is a loop.
- $q_3$ to $q_4$: 1
- $q_4$ to $q_5$: 0
- $q_5$ is a loop.

Input sequence: 0 1 0 1 0 1 0
Massive Parallelism
Massive Parallelism

• An NFA can be thought of as a DFA that can be in many states at once.
• Each symbol read causes a transition on every active state into each potential state that could be visited.
• Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
  • No fixed limit on processors; makes multicore machines look downright wimpy!
So What?

- We will turn to these three intuitions for nondeterminism more later in the quarter.

- 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 efficiently by a deterministic machine?

- The answers vary from automaton to automaton.
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.
**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.

![Diagram of ε-transitions in an NFA](image_url)
**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.

![Diagram of NFAs with ε-transitions]

```plaintext
0 0 1 0 0
```
**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.
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.
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.

![Diagram of ε-transitions in an NFA]
ε-Transitions

- NFAs have a special type of transition called the \textit{ε-transition}.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
ε-Transitions

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

![Diagram of ε-Transitions](image)
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.
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.
**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.
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.
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.
ε-Transitions

- NFAs have a special type of transition called the \textbf{ε-transition}.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.
A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.

Note the domain allows for \(\epsilon\)-moves
A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where
  - \(Q\) is a set of states.
  - \(\Sigma\) is an alphabet.
  - \(\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \wp(Q)\) is the transition function.
  - \(q_0 \in Q\) is the start state.
  - \(F \subseteq Q\) is a set of accepting states.

Note the domain allows for \(\epsilon\)-moves

Note that the codomain is sets of states to allow for multiple transitions.
Designing NFAs
Designing NFAs

• When designing NFAs, **embrace the nondeterminism**!

• Good model: **Guess-and-check**:
  • Have the machine *guess* what the right choice is.
  • Have the machine *check* that the choice was correct.
Guess-and-Check

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[ L = \{ w \mid w \in \{a, b, c\}^* \text{ and one of } a, b, \text{ or } c \text{ is not in } w \} \]
NFAs and DFAs

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

• Why?
  • Just use the same set of transitions as before.

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

• Surprisingly, the answer is **yes**!
Simulation

- **Simulation** is a key technique in computability theory.

- If we can build an automaton $A'$ whose behavior *simulates* that of another automaton $A$, then we can make a connection between $A$ and $A'$.

- To show that any language accepted by an NFA can be accepted by a DFA, we will show how to make a DFA that *simulates* the execution of an NFA.
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA

\[ q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \]

\[ q_3 \xrightarrow{\varepsilon} q_4 \xrightarrow{0} q_5 \]

\[ q_{14} \xrightarrow{0} q_{03} \xrightarrow{1} q \]

\[ q_2 \xrightarrow{1} q \]

\[ q_1 \xrightarrow{0} q_5 \]
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA

Start

$q_0$ 0 $q_1$

$\varepsilon$

$q_3$ 0 $q_4$ 0 $q_5$

$q_1$ 0 $q_2$

$q_2$ 1 $q_3$

$q_4$ 0 $q_5$

$q_5$

$q_0$ 0 $q_1$

$q_1$ 1 $q_2$

$q_2$ 0 $q_3$

$q_3$ 1 $q_4$

$q_4$ 0 $q_5$

$q_5$ 1 $q_3$
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA

Start

$q_0$ 0 $q_1$ 1 $q_2$

$q_3$ 0 $q_4$ 0 $q_5$

$q_1$ 1

$q_2$ 1 0 0

$q_0$ 0 1

$q_{14}$ 0 $q_{03}$

$q_{14}$ 1 0 0

$q_{14}$ 1

$q_{14}$ 1

$q_{03}$ 1 0

$q_{03}$ 1

$q_3$ 1

$q_3$ 1
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA

Start state: $q_0$

- From $q_0$, on input 0, go to $q_1$.
- From $q_0$, on input 1, go to $q_2$.
- From $q_0$, on input $\varepsilon$, go to $q_3$.
- From $q_1$, on input 0, go to $q_4$.
- From $q_1$, on input 1, go to $q_2$.
- From $q_1$, on input 1, go to self-loop.
- From $q_2$, on input 0, go to $q_3$.
- From $q_2$, on input 1, go to $q_5$.
- From $q_3$, on input 0, go to $q_4$.
- From $q_3$, on input 1, go to $q_5$.
- From $q_4$, on input 0, go to $q_5$.
- From $q_4$, on input 1, go to $q_3$.
- From $q_5$, on input 0, go to $q_3$.
- From $q_5$, on input 1, go to $q_3$.

Start state: $q_{14}$

- From $q_{14}$, on input 0, go to $q_{03}$.
- From $q_{14}$, on input 1, go to $q_{2}$.
- From $q_{03}$, on input 0, go to $q_{5}$.
- From $q_{03}$, on input 1, go to $q_{4}$.
- From $q_{2}$, on input 0, go to $q_{4}$.
- From $q_{2}$, on input 1, go to $q_{5}$.
- From $q_{4}$, on input 0, go to $q_{3}$.
- From $q_{4}$, on input 1, go to $q_{5}$.
- From $q_{3}$, on input 0, go to $q_{4}$.
- From $q_{3}$, on input 1, go to $q_{5}$.

$\varepsilon$-transitions:
- From $q_0$, on input $\varepsilon$, go to $q_3$.
- From $q_1$, on input $\varepsilon$, go to $q_2$.
- From $q_2$, on input $\varepsilon$, go to $q_2$.
- From $q_3$, on input $\varepsilon$, go to $q_4$.
- From $q_4$, on input $\varepsilon$, go to $q_5$.
- From $q_5$, on input $\varepsilon$, go to $q_3$.
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA

0 0 1 0 0 0

start

q₀
q₁
q₂
q₃
q₄
q₅

ε
0
1
1
0
1
0, 1
1
1
0
0
1
1
0
1

q₁₄
q₀₃
q₂
q₁
q₄
q₅

start
Simulating an NFA with a DFA
Simulating an NFA with a DFA

0 0 1 0 0
Simulating an NFA with a DFA
Simulating an NFA with a DFA
Simulating an NFA with a DFA

0 0 1 0 0 0
Simulating an NFA with a DFA
The Subset Construction

• This construction for transforming an NFA into a DFA is called the **subset construction** (or sometimes the **powerset construction**).

• Intuitively:
  • States of the new DFA correspond to **sets of states** of the NFA.
  • The initial state is the start state, plus all states reachable from the start state via $\epsilon$-transitions.
  • Transition on state $S$ on character $a$ is found by following all possible transitions on $a$ for each state in $S$, then taking the set of states reachable from there by $\epsilon$-transitions.
  • Accepting states are any set of states where *some* state is an accepting state.

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

- In converting an NFA to a DFA, the DFA's set of states is equal to the power set of the NFA's states.
- 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.
- Interesting challenge: Find a language for which this worst-case behavior occurs (there are infinitely many of them!)
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

- Constructions on DFAs allowed us to prove that regular languages are closed under complement, intersection, and difference.
- We can now also use constructions on NFAs to prove that regular languages are closed under other properties.
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 $L_1 \cup L_2$?
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 Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the set 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?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the set 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?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the set 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?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the set 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?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the set 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?
Concatenation

- The **concatenation** of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language
  \[ L_1 L_2 = \{ wx \mid w \in L_1 \land x \in L_2 \} \]
- The set of strings that can be split into two pieces: a string in $L_1$ and a string in $L_2$. 
Concatenation Example

• Example: Let $\Sigma = \{ \text{a, b, \ldots, z, A, B, \ldots, Z} \}$
  • Noun = $\{ \text{Velociraptor, Rainbow, Whale, \ldots} \}$
  • Verb = $\{ \text{Eats, Juggles, Loves, \ldots} \}$
  • The = $\{ \text{The} \}$

• The language TheNounVerbTheNoun is
  • $\{ \text{TheVelociraptorEatsTheVelociraptor, TheWhaleLovesTheRainbow, TheRainbowJugglesTheVelociraptor, \ldots} \}$
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 at $x \in L_1$ and $y \in L_2$?

Idea: Run the automaton for $L_1$ on $w$, and whenever $L_1$ reaches an accepting state 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
Concatenating Regular Languages
Concatenating Regular Languages
Concatenating Regular Languages
Concatenating Regular Languages