Finite Automata
Part One
Computability Theory
What problems can we solve with a computer?
What problems can we solve with a computer?
Computers are Messy

by CC Dharmani
www.dharmanitech.com

microSD/SD Card interface with ATmega32  Ver_2.3

http://www.dharmanitech.com/
Computers are Messy

Fig 2  Covering Everything from PCs to Supercomputers  NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM) added.
Computers are Messy

Computers are Messy

That messiness makes it hard to rigorously say what we intuitively know to be true: that, on some fundamental level, different brands of computers or programming languages are more or less equivalent in what they are capable of doing.

C vs C++

vs Java

vs Python
We need a simpler way of discussing computing machines.
An *automaton* (plural: *automata*) is a mathematical model of a computing device.
Computers are Messy

http://www.intel.com/design/intarch/prodbref/272713.htm
Automata are Clean
Computers are Messy

http://www.dharmanitech.com/
Automata are Clean
Computers are Messy

Fig 2 Covering Everything from PCs to Supercomputers  NVIDIA's CUDA architecture boasts high scalability. The quantity of processor units (SM) can be varied as needed to flexibly provide performance from PC to supercomputer levels. Tesla 10, with 240 SPs, also has double-precision operation units (SM) added.

http://techon.nikkeibp.co.jp/article/HONSHI/20090119/164259/
Automata are Clean

![Automata Diagram]
Computers are Messy

Automata are Clean
Why Build Models?

- **Mathematical simplicity.**
  - It is significantly easier to manipulate our abstract models of computers than it is to manipulate actual computers.

- **Intellectual robustness.**
  - If we pick our models correctly, we can make broad, sweeping claims about huge classes of real computers by arguing that they're just special cases of our more general models.
Why Build Models?

- The models of computation we will explore in this class correspond to different conceptions of what a computer could do.
  
  - *Finite automata* (this week) are an abstraction of computers with finite resource constraints.
    - Provide upper bounds for the computing machines that we can actually build.

  - *Turing machines* (later) are an abstraction of computers with unbounded resources.
    - Provide upper bounds for what we could ever hope to accomplish.
What problems can we solve with a computer?
What problems can we solve with a computer?

What is a “problem?”
Problems with Problems

Before we can talk about what problems we can solve, we need a formal definition of a “problem.”

We want a definition that

• corresponds to the problems we want to solve,
• captures a large class of problems, and
• is mathematically simple to reason about.

No one definition has all three properties.
Formal Language Theory
Strings

• An *alphabet* is a finite, nonempty set of symbols called *characters*.
  • Typically, we use the symbol $\Sigma$ to refer to an alphabet.
• A *string over an alphabet* $\Sigma$ is a finite sequence of characters drawn from $\Sigma$.
• Example: Let $\Sigma = \{a, b\}$. Here are some strings over $\Sigma$:
  
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>aabaaabbabaaabaaabbb</td>
</tr>
<tr>
<td>abbababba</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>aabaaabbabaaabaaabbb</td>
</tr>
<tr>
<td>abbababba</td>
</tr>
</tbody>
</table>

• The *empty string* has no characters and is denoted $\varepsilon$.
• Calling attention to an earlier point: since all strings are finite sequences of characters from $\Sigma$, you cannot have a string of infinite length.
Languages

• A *formal language* is a set of strings.
• We say that $L$ is a *language over* $\Sigma$ if it is a set of strings over $\Sigma$.
• Example: The language of palindromes over $\Sigma = \{a, b, c\}$ is the set
  $$\{\varepsilon, a, b, c, aa, bb, cc, aaa, aba, aca, bab, \ldots\}$$
• The set of all strings composed from letters in $\Sigma$ is denoted $\Sigma^*$.
• Formally, we say that $L$ is a language over $\Sigma$ if $L \subseteq \Sigma^*$.
The Cast of Characters

- **Languages** are sets of strings.
- **Strings** are finite sequences of characters.
- **Characters** are individual symbols.
- **Alphabets** are sets of characters.
Strings and Problems

Given a string \( w \), determine whether \( w \in S \).

• Suppose that \( L \) is the language
  \[
  L = \{ \, "a", "b", "c", \ldots, "z" \, \}
  \]
• This is modeling the problem:
  
  Given a string \( w \), determine whether \( w \) is a single lower-case English letter.
Strings and Problems

Given a string \( w \), determine whether \( w \in S \).

- Suppose that \( L \) is the language
  \[
  L = \{ \ p \mid p \text{ is a legal C++ program} \}
  \]

- This is modeling the problem:
  Given a string \( w \), determine whether \( w \) is a legal C++ program.
The Model

- **Fundamental Question:** For what languages $L$ can you design an automaton that takes as input a string, then determines whether the string is in $L$?

- The answer depends on the choice of $L$, the choice of automaton, and the definition of “determines.”

- In answering this question, we’ll go through a whirlwind tour of models of computation and see how this seemingly abstract question has very real and powerful consequences.
To Summarize

- An **automaton** is an idealized mathematical computing machine.
- A **language** is a set of strings, a **string** is a (finite) sequence of characters, and a **character** is an element of an **alphabet**.
- **Goal**: Figure out in which cases we can build automata for particular languages.
What problems can we solve with a computer?
Finite Automata
It’s time for another round of Mathematicalisthenics!
We will distribute one packet to each row.

When the packet comes to you, follow the directions to help it on its Magical Journey.
If you are holding a packet and the top sheet has a giant star on it, please raise your hand.
What’s going on here?
A **finite automaton** is a simple type of mathematical machine for determining whether a string is contained within some language.
Each finite automaton consists of a set of *states* connected by *transitions*. 
A Simple Finite Automaton

\[ \begin{array}{ccc}
q_0 & \xrightarrow{0} & q_1 \\
0 & \xrightarrow{1} & 1 \\
q_3 & \xrightarrow{0} & q_2 \\
\end{array} \]

\[ \begin{array}{ccc}
q_1 & \xrightarrow{0} & q_2 \\
1 & \xrightarrow{1} & 0 \\
q_2 & \xrightarrow{0} & q_3 \\
\end{array} \]
A Simple Finite Automaton

Each circle represents a state of the automaton.
A Simple Finite Automaton

The diagram shows a finite automaton with four states: $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are as follows:

- From $q_0$ to $q_0$ on input 0.
- From $q_0$ to $q_1$ on input 1.
- From $q_1$ to $q_0$ on input 1.
- From $q_1$ to $q_1$ on input 0.
- From $q_1$ to $q_2$ on input 1.
- From $q_2$ to $q_3$ on input 1.
- From $q_3$ to $q_2$ on input 1.
- From $q_2$ to $q_2$ on input 0.
- From $q_2$ to $q_3$ on input 1.

The start state is $q_0$. The diagram does not explicitly show the transitions from $q_3$ to $q_3$ on input 0 and from $q_3$ to $q_2$ on input 1, but these transitions are implied by the symmetry of the automaton.

The automaton transitions between states based on the input symbols 0 and 1, forming a loop that includes all states.

The automaton is deterministic, as each state has a unique transition for each input symbol.
A Simple Finite Automaton

One special state is designated as the start state.
A Simple Finite Automaton

\[\begin{array}{c}
\text{start} \\
q_0 \\
q_3 \\
q_2 \\
q_1
\end{array}\]

States: \(q_0, q_1, q_2, q_3\)

Transitions:
- \(q_0 \rightarrow q_1 \) on input 0
- \(q_1 \rightarrow q_0 \) on input 0
- \(q_3 \rightarrow q_2 \) on input 0
- \(q_2 \rightarrow q_3 \) on input 0
- \(q_0 \rightarrow q_3 \) on input 1
- \(q_3 \rightarrow q_1 \) on input 1
- \(q_1 \rightarrow q_2 \) on input 1
- \(q_2 \rightarrow q_0 \) on input 1
A Simple Finite Automaton

start

$q_0$ 0 $q_1$

1 1 0

$q_3$

0

$q_2$

0 0 0

0 1 0 1 1 1 0
A Simple Finite Automaton

The automaton is run on an input string and answers "yes" or "no."
A Simple Finite Automaton
A Simple Finite Automaton

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_0 \]

States:
- \( q_0 \) (Start state)
- \( q_1 \)
- \( q_2 \)
- \( q_3 \)

Transitions:
- \( q_0 \) on input 0, transition to \( q_1 \)
- \( q_1 \) on input 0, transition to \( q_2 \)
- \( q_2 \) on input 1, transition to \( q_3 \)
- \( q_3 \) on input 1, transition to \( q_0 \)
A Simple Finite Automaton

The automaton can be in one state at a time. It begins in the start state.
A Simple Finite Automaton

start

$q_0$ 0 0 0 1 1 0 0 0 0 1 1
$q_1$ 0 0 0 1 1 1 0 0 0 1 1
$q_3$ 1 1 1 0 0 0 0 0 0 0 0
$q_2$ 1 1 1 0 0 0 0 0 0 0 0

0 1 0 1 1 0
A Simple Finite Automaton

The automaton now begins processing characters in the order in which they appear.

0 1 0 1 1 1 0
A Simple Finite Automaton

\[ q_0 \rightarrow 0 \rightarrow q_1 \]
\[ q_1 \rightarrow 0 \rightarrow q_2 \]
\[ q_2 \rightarrow 1 \rightarrow q_3 \]
\[ q_3 \rightarrow 1 \rightarrow q_0 \]

Input: 01011110
A Simple Finite Automaton
A Simple Finite Automaton

Each arrow in this diagram represents a transition. The automaton always follows the transition corresponding to the current symbol being read.
A Simple Finite Automaton
A Simple Finite Automaton

start

$q_0$ 0

$q_1$

$q_3$

$q_2$

0 1 0 1 1 1 0
A Simple Finite Automaton
A Simple Finite Automaton

The diagram represents a simple finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are as follows:

- From $q_0$ on input 0, go to $q_1$.
- From $q_0$ on input 1, go to $q_3$.
- From $q_1$ on input 0, go to $q_2$.
- From $q_1$ on input 1, go to $q_2$.
- From $q_2$ on input 0, go to $q_3$.
- From $q_2$ on input 1, go to $q_2$.
- From $q_3$ on input 0, go to $q_3$.
- From $q_3$ on input 1, go to $q_3$.

The start state is $q_0$. The automaton processes the input string "0 1 0 1 1 1 0" and transitions through the states as indicated by the arrows.

The input string is processed as follows:

1. Start at $q_0$.
2. Read 0, transition to $q_1$.
3. Read 1, transition to $q_3$.
4. Read 0, transition to $q_3$.
5. Read 1, transition to $q_2$.
6. Read 0, transition to $q_2$.
7. Read 1, transition to $q_2$.
8. Read 1, transition to $q_2$.
9. Read 0, transition to $q_2$.

The automaton ends at $q_2$ after processing the input string.
A Simple Finite Automaton

After transitioning, the automaton considers the next symbol in the input.

0 1 0 1 1 1 0
A Simple Finite Automaton

\[ q_0 \rightarrow q_1 \]
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton

\[
\begin{array}{c}
q_0 & \xrightarrow{0} & q_1 \\
\xleftarrow{0} & & \xleftarrow{0} \\
q_3 & \xrightarrow{1} & q_2 \\
\xleftarrow{1} & & \xleftarrow{0} \\
\end{array}
\]
A Simple Finite Automaton

\[
\begin{array}{c}
\text{start} \\
q_0 & \xrightarrow{0} & q_1 \\
q_3 & \xrightarrow{1} & q_2 \\
q_2 & \xrightarrow{0} & q_3 \\
q_0 & \xrightarrow{0} & q_1 \\
\end{array}
\]
A Simple Finite Automaton

The diagram shows a finite automaton with the following states:

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

The automaton accepts the string $0101110$.
A Simple Finite Automaton

\[
\begin{array}{c}
q_0 \quad 0 \quad q_1 \\
1 \quad 0 \quad 1 \\
q_3 \quad 0 \quad q_2 \\
0 \quad 0 \quad 0
\end{array}
\]

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

start

$q_0$ 0 0

$q_3$ 1 1

$q_1$ 0 0

$q_2$ 1 1

0 1 0 1 1 1 0
A Simple Finite Automaton

\[ q_0 \rightarrow 0 \rightarrow q_1 \]

\[ q_3 \downarrow 1 \rightarrow q_2 \]

\[ q_1 \rightarrow 0 \rightarrow q_2 \]

\[ 0 1 0 1 1 1 0 \]
A Simple Finite Automaton

start

$\begin{align*}
q_0 & \rightarrow 0 \rightarrow q_1 \\
& \rightarrow 0 \rightarrow q_2 \\
q_3 & \rightarrow 0 \rightarrow q_2 \\
& \rightarrow 0 \rightarrow q_1 \\
\end{align*}$

Input: 0 1 0 1 1 1 0
A Simple Finite Automaton

\( q_0 \leftarrow 0 \rightarrow q_1 \)
\( q_3 \leftarrow 1 \rightarrow q_2 \)
\( q_2 \leftarrow 0 \rightarrow q_3 \)
\( q_1 \leftarrow 0 \rightarrow q_2 \)

\( 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \)
A Simple Finite Automaton

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_0 \]

Start state: \( q_0 \)

Transitions:
- \( q_0 \) on 0 goes to \( q_1 \)
- \( q_0 \) on 1 goes to \( q_3 \)
- \( q_1 \) on 0 goes to \( q_2 \)
- \( q_1 \) on 1 goes to \( q_0 \)
- \( q_3 \) on 0 goes to \( q_2 \)
- \( q_2 \) on 0 goes to \( q_3 \)
- \( q_2 \) on 1 goes to \( q_1 \)

Input sequence: 0 1 0 1 1 1 0
A Simple Finite Automaton
A Simple Finite Automaton

$\begin{array}{c}
\text{start} \\
q_0 \\
q_3 \\
q_2 \\
q_1
\end{array}$

$\begin{array}{c}
0 \\
1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1
\end{array}$
A Simple Finite Automaton
A Simple Finite Automaton

- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{0} q_1$
  - $q_0 \xrightarrow{1} q_3$
  - $q_1 \xrightarrow{0} q_2$
  - $q_1 \xrightarrow{1} q_1$
  - $q_3 \xrightarrow{1} q_2$

Input sequence: 0 1 0 1 1 1 0
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton

Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."
A Simple Finite Automaton

Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

The double circle indicates that this state is an accepting state, so the automaton outputs "yes."

0 1 0 1 1 1 0
A Simple Finite Automaton

Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

The double circle indicates that this state is an accepting state, so the automaton outputs "yes."

0 1 0 1 1 0
Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

The double circle indicates that this state is an accepting state, so the automaton outputs "yes."
A Simple Finite Automaton

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

\[ q_0 \xrightarrow{1} q_3 \xrightarrow{1} q_0 \]

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

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

\[ \text{start} \]
A Simple Finite Automaton

1 0 1 0 0 0
A Simple Finite Automaton

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

start

1 1 1

0 0 0

1 0 1 0 0 0
A Simple Finite Automaton

- Start: $q_0$
- Transitions:
  - $q_0 \xrightarrow{0} q_1$
  - $q_0 \xrightarrow{1} q_3$
  - $q_1 \xrightarrow{0} q_0$
  - $q_1 \xrightarrow{1} q_2$
  - $q_2 \xrightarrow{1} q_1$
  - $q_2 \xrightarrow{1} q_0$
  - $q_3 \xrightarrow{1} q_1$
  - $q_3 \xrightarrow{1} q_0$

Input String: 1 0 1 0 0 0
A Simple Finite Automaton

start

$q_0$

$q_1$

$q_2$

$q_3$

$1 1 1 1 1 1$

$0 0 0$

$0 0 0$

$1 0 1 0 0 0 0$

.
A Simple Finite Automaton

\begin{center}
\begin{tikzpicture}[node distance=2cm, thick, bend angle=45, auto]
    \node[state,initial] (q0) {$q_0$};
    \node[state] (q1) [right of=q0] {$q_1$};
    \node[state,accepting] (q3) [below of=q0] {$q_3$};
    \node[state] (q2) [right of=q3] {$q_2$};

    \path[->]
    (q0) edge [loop left] node {0} ()
    edge [bend left] node {1} (q3)
    edge [bend right] node {1} (q1)
    (q1) edge [loop right] node {0} ()
    edge [bend left] node {1} (q2)
    (q2) edge [loop right] node {0} ()
    edge [bend right] node {1} (q0)
    (q3) edge [loop below] node {0} ()
    edge [bend left] node {1} (q1)
    edge [bend right] node {1} (q2);
\end{tikzpicture}
\end{center}

Input sequence: 1 0 1 0 0 0 0
A Simple Finite Automaton

![Finite Automaton Diagram]

- States: $q_0$, $q_1$, $q_2$, $q_3$
- Transitions:
  - $q_0$ to $q_1$: on input 0
  - $q_0$ to $q_3$: on input 1
  - $q_1$ to $q_2$: on input 0
  - $q_2$ to $q_3$: on input 1
  - $q_3$ to $q_0$: on input 0

Start state: $q_0$

Input sequence: 1 0 1 0 0 0 0
A Simple Finite Automaton
A Simple Finite Automaton

start

$q_0$ 0 $q_1$

1 1 1 1 1

$q_3$ 0 $q_2$

1 0 1 0 0 0

$q_0$ 0 1

$q_1$ 0 0

$q_2$ 1 1 1

$q_3$ 0 0
A Simple Finite Automaton
A Simple Finite Automaton

The diagram shows a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are as follows:

- From $q_0$: On input 0, move to $q_1$.
- From $q_1$: On input 0, move to $q_0$.
- From $q_2$: On input 1, move to $q_3$.
- From $q_3$: On input 1, move to $q_2$.

The automaton starts at state $q_0$. The input sequence is 1010000.
A Simple Finite Automaton
A Simple Finite Automaton

![Diagram of a finite automaton with states q₀, q₁, q₂, q₃ and transitions labeled with 0s and 1s. The start state is q₀ and the accepting state is q₂. The input sequence is 1010000.]
A Simple Finite Automaton

A finite automaton is a mathematical model of computation that recognizes patterns in a sequence of inputs. The automaton has a set of states, an initial state, and a set of transitions between states based on input symbols. In this diagram, the states are labeled $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with 0s and 1s, indicating the symbols that cause a transition between states.

The automaton starts in the state $q_0$ and follows the transitions based on the input sequence. The input sequence is shown at the bottom of the diagram. The automaton accepts the sequence if it ends in a final state.
A Simple Finite Automaton

\begin{center}
\begin{tikzpicture}
  \node[state,initial] (q0) at (0,0) {$q_0$};
  \node[state,accepting] (q1) at (1,0) {$q_1$};
  \node[state] (q3) at (0,-1) {$q_3$};
  \node[state] (q2) at (1,-1) {$q_2$};
  \draw (q0) edge[loop below] node {$0$} (q0);
  \draw (q0) edge[below] node {$0$} (q1);
  \draw (q1) edge[loop above] node {$0$} (q1);
  \draw (q1) edge[above] node {$0$} (q2);
  \draw (q2) edge[loop left] node {$0$} (q2);
  \draw (q2) edge[below] node {$0$} (q3);
  \draw (q3) edge[loop right] node {$0$} (q3);
  \draw (q3) edge[above] node {$1$} (q0);
\end{tikzpicture}
\end{center}
A Simple Finite Automaton
A Simple Finite Automaton

![A Simple Finite Automaton Diagram]
A Simple Finite Automaton

\[ q_0 \rightarrow 0 \rightarrow q_1 \]
\[ q_0 \rightarrow 1 \rightarrow q_3 \]
\[ q_3 \rightarrow 1 \rightarrow q_2 \]
\[ q_2 \rightarrow 1 \rightarrow q_0 \]
\[ q_1 \rightarrow 0 \rightarrow q_0 \]

Input: 1 0 1 0 0 0
A Simple Finite Automaton

start

$q_0$  $q_1$

$q_3$  $q_2$

1 1 1 1 1 1 1 1

1 0 1 0 0 0
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton
A Simple Finite Automaton

![Finite Automaton Diagram]
A Simple Finite Automaton

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

Diagram:

- Start state: $q_0$
- States: $q_0$, $q_1$, $q_2$, $q_3$
- Transitions:
  - From $q_0$: 0 to $q_1$, 1 to $q_3$
  - From $q_1$: 0 to $q_2$ (with an arrow), 1 to $q_0$
  - From $q_2$: 0 to $q_3$, 1 to $q_1$
  - From $q_3$: 1 to $q_0$, 0 to $q_2$

Arrow pointing up to the right at the end of the string.
A Simple Finite Automaton

\[ q_0 \rightarrow q_1 \]

\[ q_3 \rightarrow q_2 \]

1 1 1 0 0 0 0
A Simple Finite Automaton
A Simple Finite Automaton

\[
\begin{array}{ccccccc}
q_0 & \rightarrow & 0 & \rightarrow & q_1 & \rightarrow & 1 & \rightarrow & q_2 & \rightarrow & 0 & \rightarrow & q_3 & \rightarrow & 1 & \rightarrow & q_0 \\
\text{start} & & & & & & & & & & & & &
\end{array}
\]
A Simple Finite Automaton

This state is not an accepting state (it’s a rejecting state), so the automaton says “no.”
A Simple Finite Automaton

This state is not an accepting state (it’s a rejecting state), so the automaton says “no.”

1 0 1 0 0 0 0
A Simple Finite Automaton

This state is not an accepting state (it's a rejecting state), so the automaton says "no."

1 0 1 0 0 0
A Simple Finite Automaton

\begin{center}
\begin{tikzpicture}
  \node[state,initial] (q0) {$q_0$};
  \node[state] (q1) [right of=q0] {$q_1$};
  \node[state] (q2) [below of=q1] {$q_2$};
  \node[state] (q3) [left of=q2] {$q_3$};

  \draw[->] (q0) -- node[auto] {1} (q3);
  \draw[->] (q3) -- node[auto] {1} (q2);
  \draw[->] (q2) -- node[auto] {1} (q1);
  \draw[->] (q1) -- node[auto] {1} (q0);

  \draw[->] (q0) -- node[auto] {0} (q1);
  \draw[<->] (q1) -- node[auto] {0} (q2);
  \draw[<->] (q2) -- node[auto] {0} (q3);
  \draw[<->] (q3) -- node[auto] {0} (q0);
\end{tikzpicture}
\end{center}
A Simple Finite Automaton

Try it yourself! Does this automaton accept or reject this string?
The Story So Far

• A **finite automaton** is a collection of **states** joined by **transitions**.

• Some state is designated as the **start state**.

• Some number of states are designated as **accepting states**.

• The automaton processes a string by beginning in the start state and following the indicated transitions.

• If the automaton ends in an accepting state, it **accepts** the input.

• Otherwise, the automaton **rejects** the input.
Time-Out For Announcements!
Problem Sets

• Problem Set Three solutions are now available online and in hardcopy.
• We’ll aim to get PS3 graded and returned by Thursday morning.
• Problem Set Four is due this Friday at 3:00PM.
  • As always, ask questions if you have them! Office hours and Piazza are great places to start.
Extra Practice Problems

• We’ve posted a set of around 30 practice problems on the course website spanning all the topics we’ve covered so far.

• (Optional but Good) idea: reflect on your performance so far in the course and identify any areas you can continue to improve.

• More generally, the TAs and I are happy to meet with you if you’d like to chat about your progress in the course.
Back to CS103!
Just Passing Through
Just Passing Through

\[
\begin{array}{c}
\text{start} \\
\rightarrow \\
q_0 \\
\downarrow 1 \\
q_1 \\
\uparrow 0 \\
q_2 \\
\downarrow 0 \\
q_3 \\
\downarrow 1 \\
q_4 \\
\downarrow 1 \\
\end{array}
\]

1 1 0 1
Just Passing Through

Diagram of a finite state machine with states $q_0$, $q_1$, $q_2$, $q_3$, and $q_4$, and transitions labeled with 1s and 0s.
Just Passing Through

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto, >=latex]

  \node (q0) [state, fill=yellow] {$q_0$};
  \node (q1) [state] at (q0 -| 2,0) {$q_1$};
  \node (q2) [state] at (q0 -| -2,0) {$q_2$};
  \node (q3) [state] at (q1 -| 2,0) {$q_3$};
  \node (q4) [state] at (q1 -| -2,0) {$q_4$};

  \path[->]
  (start) edge node [left] {$\text{start}$} (q0)
  (q0) edge node [above] {1} (q1)
  (q0) edge node [below] {0} (q2)
  (q1) edge node [above] {1} (q3)
  (q1) edge node [below] {1} (q4)
  (q2) edge node [above] {0} (q3)
  (q2) edge node [below] {0} (q4)
  (q3) edge node [above] {0} (q0)
  (q3) edge node [below] {1} (q2)
  (q4) edge node [above] {1} (q0)
  (q4) edge node [below] {0} (q2);
\end{tikzpicture}
\end{center}

\begin{align*}
1 & 1 & 0 & 1
\end{align*}
Just Passing Through

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1
\end{array}
\]
Just Passing Through
Just Passing Through

```
start

\[ q_0 \]

\[ q_1 \]
\[ q_2 \]

\[ q_3 \]
\[ q_4 \]

1 1 0 1
```

1 1 0 1
Just Passing Through

Diagram of a finite state machine with states $q_0$, $q_1$, $q_2$, $q_3$, and $q_4$. The initial state is $q_0$ and the input sequence is 1101.
Just Passing Through
Just Passing Through
Just Passing Through
Just Passing Through

The diagram shows a finite state machine (FSM) with states $q_0$, $q_1$, $q_2$, $q_3$, and $q_4$. The initial state is $q_0$. Transitions are labeled with input symbols (1 or 0). The sequence 1101 is shown on the right side of the diagram.
Just Passing Through

$\begin{align*}
\text{start} & \rightarrow q_0 \\
q_1 & \rightarrow q_0 \\
q_2 & \rightarrow q_0 \\
q_3 & \rightarrow q_0 \\
q_4 & \rightarrow q_0 \\
\end{align*}$

1 1 0 1
Just Passing Through

start

$q_0$

$q_1$

$q_2$

$q_3$

$q_4$

1 1 0 1

1 1 0 1
A finite automaton does *not* accept as soon as it enters an accepting state.

A finite automaton accepts if it *ends* in an accepting state.
What Does This Accept?
What Does This Accept?
What Does This Accept?
What Does This Accept?

![Diagram of a finite automaton](image-url)
What Does This Accept?

Diagram:
- Start state: \( q_0 \)
- Transitions:
  - From \( q_0 \) to \( q_1 \) on input 1
  - From \( q_0 \) to \( q_2 \) on input 0
  - From \( q_1 \) to \( q_0 \) on input 0
  - From \( q_1 \) to \( q_3 \) on input 1
  - From \( q_2 \) to \( q_4 \) on input 0
  - From \( q_3 \) to \( q_1 \) on input 1
  - From \( q_3 \) to \( q_4 \) on input 0
  - From \( q_4 \) to \( q_2 \) on input 0

States:
- \( q_0 \)
- \( q_1 \)
- \( q_2 \)
- \( q_3 \)
- \( q_4 \)
What Does This Accept?
What Does This Accept?
No matter where we start in the automaton, after seeing two 1's, we end up in accepting state $q_3$. 
What Does This Accept?
What Does This Accept?
What Does This Accept?
What Does This Accept?
What Does This Accept?
What Does This Accept?

![Diagram of a finite automaton with states and transitions labeled with 0s and 1s. The start state is labeled as $q_0$. Transitions include $q_0$ to $q_1$ on input 1, $q_1$ to $q_2$ on input 0, $q_2$ to $q_3$ on input 1, and $q_3$ to $q_4$ on input 0. The state $q_4$ is shaded.]
What Does This Accept?

No matter where we start in the automaton, after seeing two 0's, we end up in accepting state $q_4$. 
What Does This Accept?
What Does This Accept?

This automaton accepts a string in \( \{0, 1\}^* \) iff the string ends in 00 or 11.
The *language of an automaton* is the set of strings that it accepts.

If $D$ is an automaton that processes characters from the alphabet $\Sigma$, then $\mathcal{L}(D)$ is formally defined as

$$\mathcal{L}(D) = \{ w \in \Sigma^* \mid D \text{ accepts } w \}$$
A Small Problem

\begin{center}
\begin{tikzpicture}
  \node[draw, circle] (q0) at (0,0) {$q_0$};
  \node[draw, circle] (q1) at (2,0) {$q_1$};
  \node[draw, circle] (q2) at (0,-2) {$q_2$};
  \draw[->] (q0) -- node[above] {\text{start}} (q0);
  \draw[->] (q0) -- node[left] {0} (q2);
  \draw[->] (q0) -- node[right] {0} (q1);
  \draw[->] (q1) -- node[below] {1} (q2);
\end{tikzpicture}
\end{center}
A Small Problem

\begin{align*}
\text{start} & \rightarrow q_0 \\
0 & \rightarrow q_0 \\
0 & \rightarrow q_0 \\
1 & \rightarrow q_1 \\
q_2 & \rightarrow 1
\end{align*}

0 1 1 0
A Small Problem
A Small Problem

\[
\begin{array}{c}
\text{start} \\
\downarrow \\
q_0 \\
\downarrow \\
q_2 \\
\downarrow \\
1 \\
\rightarrow \\
q_1 \\
\end{array}
\]

Input: 0 1 1 0
A Small Problem
A Small Problem

\[\text{start} \rightarrow q_0 \rightarrow q_1 \rightarrow q_2\]

Sequence: 0110
A Small Problem

\[
\begin{array}{c}
\text{start} \\
q_0 \\
0 \\
0 \\
q_2 \\
1 \\
q_1 \\
01110
\end{array}
\]
A Small Problem

0 1 1 0

0 1

start

\( q_0 \)

\( q_1 \)

\( q_2 \)
A Small Problem

\[
\begin{align*}
\text{start} & \rightarrow q_0 \\
q_0 & \rightarrow 0, 0 \\
q_2 & \rightarrow 1 \\
q_1 & \rightarrow 0
\end{align*}
\]
Another Small Problem
Another Small Problem

\[ q_0 \overset{0, 1}{\longrightarrow} q_1 \overset{0}{\rightarrow} q_2 \]

Initial state: \( q_0 \)

Final state: \( q_1 \)

Transition: \( 0, 1 \)
Another Small Problem
Another Small Problem

Start

q_0 \xrightarrow{0, 1} q_1 \xrightarrow{0, 1} q_2

0, 1

0

\begin{array}{cccc}
0 & 0 & 0 & 0
\end{array}
Another Small Problem

Diagram:

- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{0, 1} q_1$
  - $q_1 \xrightarrow{0} q_2$
  - $q_2 \xrightarrow{0, 1} q_1$

Input sequence: 0 0 0
Another Small Problem

\[
\begin{array}{c}
\text{start} \\
q_0 \quad 0, 1 \\
\quad 0, 1 \\
q_1 \quad 0, 1 \\
\quad 0 \\
q_2 \quad 0, 1 \\
\end{array}
\]
Another Small Problem
Another Small Problem

Diagram:

- Start at $q_0$
- From $q_0$ to $q_1$: 0, 1
- From $q_1$ to $q_1$: 0, 1
- From $q_1$ to $q_2$: 0
- From $q_2$: 0, 1
The Need for Formalism

- In order to reason about the limits of what finite automata can and cannot do, we need to formally specify their behavior in all cases.

- All of the following need to be defined or disallowed:
  - What happens if there is no transition out of a state on some input?
  - What happens if there are multiple transitions out of a state on some input?
DFAs

• A *DFA* is a
  • Deterministic
  • Finite
  • Automaton

• DFAs are the simplest type of automaton that we will see in this course.
DFAs

• 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 $\Sigma$.
  
  • This is the “deterministic” part of DFA.

• There is a unique start state.

• There are zero or more accepting states.
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \{0, 1\}?
Is this a DFA over \(\{0, 1\}\)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA over \{0, 1\}?
Is this a DFA over \( \{0, 1\} \)?
Is this a DFA?
Is this a DFA?
Is this a DFA?

Drinking Family of Aardvarks
Designing DFAs

• At each point in its execution, the DFA can only remember what state it is in.

• **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember.
  
  • Each state acts as a “memento” of what you're supposed to do next.
  
  • Only finitely many different states means only finitely many different things the machine can remember.
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* | \text{the number of } b\text{'s in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* | \text{the number of } b\text{'s in } w \text{ is congruent to two modulo three} \}$
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* | \text{the number of b's in } w \text{ is congruent to two modulo three } \}$
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* | \text{the number of } b's \text{ in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* | \text{the number of } b's \text{ in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

\[ L = \{ \ w \in \{a, b\}^* | \text{the number of } b \text{'s in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* | \text{the number of b's in } w \text{ is congruent to two modulo three} \}$
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid \text{the number of } b\text{'s in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* | \text{the number of b's in } w \text{ is congruent to two modulo three} \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* | \text{the number of b's in } w \text{ is congruent to two modulo three} \}$

Each state remembers the remainder of the number of bs seen so far modulo three.
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ \ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring } \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{ a, b \}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* | w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
More Elaborate DFAs

$L = \{ \ w \in \{a, *, /\}* \mid w \text{ represents a C-style comment} \ \}$

Let's have the \texttt{a} symbol be a placeholder for "some character that isn't a star or slash."

Try designing a DFA for comments! Here's some test cases to help you check your work:

**Accepted:**

/*a*/
/**/
/***/
/*aaa*aaa*/
/*a/a*/

**Rejected:**

/***
/***/a/*aa*/
aaa/***/aa
/*
/****
//aaaa
More Elaborate DFAs

$L = \{ \, w \in \{ a, *, / \}^* \mid w \text{ represents a C-style comment} \, \}$
The Regular Languages
A language \( L \) is called a **regular language** if there exists a DFA \( D \) such that \( \mathcal{L}(D) = L \).

If \( L \) is a language and \( \mathcal{L}(D) = L \), we say that \( D \) **recognizes** the language \( L \).
Let’s take a five minute break!
Revisiting a Problem

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

$q_0$, $q_1$, $q_2$, $q_3$
NFAs

• An **NFA** is a
  • **N**ondeterministic
  • **F**inite
  • **A**utomaton

• Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation.
(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.
  - (This sort of nondeterminism is technically called **existential nondeterminism**, the most philosophical-sounding term we’ll introduce all quarter.)
A Simple NFA

Start

$q_0$ 1 $q_1$

$q_1$ 1 $q_2$

$q_2$

$q_3$

0, 1

0, 1
A Simple NFA

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

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

- Start state: $q_0$
- Accept states: $q_2, q_3$
- Transitions:
  - $q_0$ to $q_1$: on 1
  - $q_1$ to $q_2$: on 1
  - $q_0$ to $q_3$: on 0, 1
  - $q_3$ to $q_2$: on 0, 1

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

\[ \begin{align*}
&\text{start} \\
&\rightarrow q_0 \\
&\quad \rightarrow q_1 \\
&\quad \rightarrow q_2 \\
&\quad \rightarrow q_3 \\
&\quad \rightarrow \circ \\
\end{align*} \]

\[ \begin{align*}
&0, 1 \\
&0, 1 \\
&0, 1 \\
&0, 1 \\
\end{align*} \]

\[ \begin{align*}
&0 \\
&1 \\
&0 \\
&1 \\
&1 \\
\end{align*} \]
A Simple NFA

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

0 1 0 1 1
A Simple NFA

start

$q_0$ 1 $q_1$

$q_1$ 1 $q_2$

$q_3$ 0 $q_2$

$q_3$ 0, 1 $q_2$

$q_3$ 0, 1 $q_2$

$q_3$ 0, 1 $q_2$

0 1 0 1 1
A Simple NFA

\begin{center}
\begin{tikzpicture}
  \node[state, fill=yellow, initial] (q0) at (0,0) {$q_0$};
  \node[state] (q1) at (2,0) {$q_1$};
  \node[state, accepting] (q2) at (4,0) {$q_2$};
  \node[state] (q3) at (4,-2) {$q_3$};

  \draw[->] (q0) edge node {1} (q1);
  \draw[->] (q1) edge node {1} (q2);
  \draw[->] (q1) edge [loop above] node {0, 1} (q1);
  \draw[->] (q0) edge [loop below] node {0, 1} (q0);
  \draw[->] (q2) edge [loop below] node {0, 1} (q2);
  \draw[->] (q3) edge node {0} (q1);
  \draw[->] (q3) edge node {0, 1} (q2);
  \draw[->] (q3) edge [loop below] node {0, 1} (q3);

\end{tikzpicture}
\end{center}

\begin{array}{cccccc}
0 & 1 & 0 & 1 & 1 & 1
\end{array}
A Simple NFA

\[
\begin{array}{c}
\text{start} \\
q_0 \quad 1 \\
q_1 \quad 1 \\
q_2 \\
q_3 \\
\end{array}
\]

\[
\begin{array}{c}
0, 1 \\
0 \\
0, 1 \\
0, 1 \\
\end{array}
\]

\[
\begin{array}{c}
0 \\
1 \\
0 \\
1 \\
1 \\
\end{array}
\]
A Simple NFA

start → $q_0$ (0, 1) → $q_1$ (1) → $q_2$ (1) → $q_3$ (0, 1) → $q_3$ (0, 1) → $q_2$ (1) → $q_1$ (1) → $q_0$ (0, 1)

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

Start
$q_0$ → $q_1$: 1
$q_1$ → $q_2$: 1
$q_1$ → $q_3$: 0, 1
$q_3$ → $q_2$: 0, 1

Input:

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

\[
\begin{align*}
q_0 & \xrightarrow{1} q_1 & q_1 & \xrightarrow{1} q_2 \\
q_0 & \xrightarrow{0, 1} q_3 & q_3 & \xrightarrow{0, 1} q_3 \\
q_3 & \xrightarrow{0} q_3 & q_3 & \xrightarrow{0, 1} q_3 \\
\end{align*}
\]
A Simple NFA

0 1 0 1 1
A Simple NFA

```
0 1 0 1 1
```

Diagram:
- Start state: $q_0$
- Transitions:
  - $q_0$ to $q_1$: 1
  - $q_0$ to $q_3$: 0, 1
  - $q_1$ to $q_2$: 1
  - $q_2$ to $q_3$: 0, 1
  - $q_3$: Self loop on 0, 1
A Simple NFA
A Simple NFA
A Simple NFA
A Simple NFA
A Simple NFA
A Simple NFA

- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{1} q_1$
  - $q_1 \xrightarrow{1} q_2$
  - $q_2 \xrightarrow{0,1} q_3$
  - $q_3 \xrightarrow{0,1} q_2$

Input string: 010111
A Simple NFA

start → $q_0$ 1 $q_1$ 1 $q_2$

$q_0$ 0, 1

$q_1$ 0

$q_2$ 0, 1

$q_3$ 0, 1

0 1 0 1 1
A Simple NFA
A Simple NFA

Start: $q_0$

Transitions:
- $q_0 \xrightarrow{1} q_1$
- $q_1 \xrightarrow{1} q_2$
- $q_3 \xrightarrow{0} q_3$
- $q_3 \xrightarrow{0,1} q_2$
- $q_3 \xrightarrow{0,1} q_1$

Input string: 0 1 0 1 1

States:
- $q_0$
- $q_1$
- $q_2$
- $q_3$

Final state: $q_2$
A Simple NFA

start

$q_0$ 1 $q_1$

$q_1$ 1 $q_2$

$q_0$ 0, 1 $q_3$

$q_3$ 0 $q_2$

$q_3$ 0, 1

$q_2$ 0, 1

0 1 0 1 1
A Simple NFA

Start: $q_0$

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

States: $q_0, q_1, q_2, q_3$

Accepting State: $q_2$
A More Complex NFA

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

If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path does not accept.
A More Complex NFA

0 1 0 1 1
A More Complex NFA

start $\rightarrow q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$

$0, 1$

$0 \ 1 \ 0 \ 1 \ 1 \ 1$
A More Complex NFA
A More Complex NFA

![NFA Diagram]

- **Start State**: $q_0$
- **States**: $q_0$, $q_1$, $q_2$
- **Transitions**:
  - $q_0$ to $q_1$: 1
  - $q_1$ to $q_2$: 1
  - $q_0$: 0, 1 (loop)

Input String: 0 1 0 1 1
A More Complex NFA
A More Complex NFA
A More Complex NFA

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

start

$q_0$ 1 $q_1$ 1 $q_2$

0, 1

0 1 0 1 1
A More Complex NFA

Start

$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$

$0, 1$

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

\[
\begin{array}{c}
\text{start} \\
q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2
\end{array}
\]

\[
0, 1
\]

\[
0 \quad 1 \quad 0 \quad 1 \quad 1
\]
A More Complex NFA

Start state $q_0$ transitions to $q_1$ on input 1. $q_1$ transitions to $q_2$ on input 1. $q_2$ loops back to itself on input 0, 1.

Input sequence: 0 1 0 1 1
A More Complex NFA

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

Input sequence: \[010111\]
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 $q_0$ to $q_1$ on input 1
- From $q_1$ to $q_2$ on input 1
- From $q_0$ to $q_1$ on input 0 and 1
- The loop on $q_2$ is not labeled.

The input sequence is 0 1 0 1 1.
A More Complex NFA

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

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

Input sequence: \( 0 \ 1 \ 0 \ 1 \ 1 \)
A More Complex NFA

![Diagram of a more complex NFA with states q₀, q₁, and q₂ connected by transitions labeled 0, 1, and 0, 1, respectively. A string of 01011 is shown as an example input.]
A More Complex NFA

A NFA with states $q_0$, $q_1$, and $q_2$. The transitions are as follows:
- From $q_0$ to $q_1$ with input 1
- From $q_1$ to $q_2$ with input 1

The initial state is $q_0$, and $q_2$ is the final state.