Deterministic Finite Automata (DFA)

A Simple Computing Machine: States and Transitions
What Problems Can It Solve: Regular Languages
Is There A Problem It Can’t Solve?
Computing Problems and Their Difficulty

- Computing Problem
- Decision Problem
Computing Problems and Their Difficulty

Language $\mathcal{L}$: \{YES\}-set of finite binary strings

Computing Problem

Decision Problem

Deterministic Finite Automata (DFA): 2 / 14
Computing Problems and Their Difficulty

- Computing Problem
- Decision Problem
- How hard is the problem?

Language $\mathcal{L}$: \{YES\}-set of finite binary strings
Computing Problems and Their Difficulty

Language $L$: \{YES\}-set of finite binary strings

- How hard is the problem?
- How complex is $L$?
- How hard is it to test membership in $L$?
A problem can be harder in two ways.
Computing Problems and Their Difficulty

A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.
Computing Problems and Their Difficulty

A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.

2. The problem needs a different kind of computing machine, with superior capabilities.
A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.

2. The problem needs a different *kind* of computing machine, with superior capabilities.

The first type of “harder” is the focus of a follow-on algorithms course.
A problem can be harder in two ways.

1. The problem needs more resources. For example, the problem can be solved with a similar machine to ours, except with more states.

2. The problem needs a different *kind* of computing machine, with superior capabilities.

The first type of “harder” is the focus of a follow-on algorithms course.

We focus on what *can and can’t be solved* on a particular kind of machine.
Today: Deterministic Finite Automata (DFA)

1. A simple computing machine.
   - States.
   - Transitions.
   - No scratch paper.

2. What computing problems can this simple machine solve?
   - Vending machine.

3. Regular languages.
   - Closed under all the set operations: union, intersection, complement, concatenation, Kleene-star.

4. Are there problems that cannot be solved?
A Simple Computing Machine
A Simple Computing Machine

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

Running the Machine
A Simple Computing Machine
A Simple Computing Machine

Running the Machine
A Simple Computing Machine

Running the Machine

### Deterministic Finite Automata (DFA): 4 / 14

Creator: Malik Magdon-Ismail
A Simple Computing Machine

Deterministic Finite Automata (DFA): 4 / 14

Running the Machine ➔
A Simple Computing Machine

In state $q_0$, if you read 0, transition to $q_1$.

In state $q_0$, if you read 0, transition to $q_1$.

Transitions:

1: $q_0 \xrightarrow{0} q_1$
2: $q_0 \xrightarrow{1} q_2$
A Simple Computing Machine

Transitions:
1: $q_0 \rightarrow q_1$  
2: $q_0 \rightarrow q_2$  
3: $q_1 \rightarrow q_1$

In state $q_0$, if you read 0, transition to $q_1$.  

In state $q_1$, if you read 1, transition to $q_2$.  

In state $q_1$, if you read 0, transition to $q_1$.  

Running the Machine
A Simple Computing Machine

Running the Machine

transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>State 1</th>
<th>Input</th>
<th>State 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q₀</td>
<td>0</td>
<td>q₁</td>
</tr>
<tr>
<td>2</td>
<td>q₀</td>
<td>1</td>
<td>q₂</td>
</tr>
<tr>
<td>3</td>
<td>q₁</td>
<td>0</td>
<td>q₁</td>
</tr>
<tr>
<td>4</td>
<td>q₁</td>
<td>1</td>
<td>q₂</td>
</tr>
</tbody>
</table>

In state q₀, if you read 0, transition to q₁.
A Simple Computing Machine

![Diagram of a finite automaton]

Transitions:

<table>
<thead>
<tr>
<th>Transition</th>
<th>State 0</th>
<th>Input</th>
<th>State 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q₀</td>
<td>0</td>
<td>q₁</td>
</tr>
<tr>
<td>2</td>
<td>q₀</td>
<td>1</td>
<td>q₂</td>
</tr>
<tr>
<td>3</td>
<td>q₁</td>
<td>0</td>
<td>q₁</td>
</tr>
<tr>
<td>4</td>
<td>q₁</td>
<td>1</td>
<td>q₂</td>
</tr>
<tr>
<td>5</td>
<td>q₂</td>
<td>0</td>
<td>q₂</td>
</tr>
<tr>
<td>6</td>
<td>q₂</td>
<td>1</td>
<td>q₂</td>
</tr>
</tbody>
</table>

In state q₀, if you read 0, transition to q₁.
A Simple Computing Machine

Deterministic Finite Automata (DFA):

Running the Machine →

---

Transitions:

<table>
<thead>
<tr>
<th>Number</th>
<th>Input</th>
<th>Current State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>$q_1$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

In state $q_0$, if you read 0, transition to $q_1$. 

---
A Simple Computing Machine

Deterministic Finite Automata (DFA): 4 / 14

Running the Machine →

Creator: Malik Magdon-Ismail
A Simple Computing Machine

1: Process the input string (left-to-right) starting from the initial state $q_0$. 

---

<table>
<thead>
<tr>
<th>transitions</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $q_0$ 0 $q_1$</td>
<td>$q_0$ NO</td>
</tr>
<tr>
<td>2: $q_0$ 1 $q_2$</td>
<td>$q_1$ YES</td>
</tr>
<tr>
<td>3: $q_1$ 0 $q_1$</td>
<td>$q_2$ NO</td>
</tr>
<tr>
<td>4: $q_1$ 1 $q_2$</td>
<td></td>
</tr>
<tr>
<td>5: $q_2$ 0 $q_2$</td>
<td></td>
</tr>
<tr>
<td>6: $q_2$ 1 $q_2$</td>
<td></td>
</tr>
</tbody>
</table>

In state $q_0$, if you read 0, transition to $q_1$. 

---

Running the Machine
A Simple Computing Machine

1: Process the input string (left-to-right) starting from the initial state $q_0$.
2: Process one bit at a time, each time transitioning from the current state to the next state according to the transition instructions.

---

<table>
<thead>
<tr>
<th>transitions</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $q_0$ 0 $q_1$</td>
<td>$q_0$ NO</td>
</tr>
<tr>
<td>2: $q_0$ 1 $q_2$</td>
<td>$q_1$ YES</td>
</tr>
<tr>
<td>3: $q_1$ 0 $q_1$</td>
<td>$q_2$ NO</td>
</tr>
<tr>
<td>4: $q_1$ 1 $q_2$</td>
<td></td>
</tr>
<tr>
<td>5: $q_2$ 0 $q_2$</td>
<td></td>
</tr>
<tr>
<td>6: $q_2$ 1 $q_2$</td>
<td></td>
</tr>
</tbody>
</table>

In state $q_0$, if you read 0, transition to $q_1$.  

---
A Simple Computing Machine

1: Process the input string (left-to-right) starting from the initial state $q_0$.
2: Process one bit at a time, each time transitioning from the current state to the next state according to the transition instructions.
3: When done processing every bit, output \textbf{YES} if the final resting state of the DFA is a \textbf{YES}-state; otherwise output \textbf{NO}.
Running the Machine on an Input

\[ \begin{array}{c|cc}
0 & 1 & 0 \\
\hline
\uparrow & & \\
\end{array} \]

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

Computing Problem Solved by a DFA →
Running the Machine on an Input


$q_0 \rightarrow 010$

$q_0 | \triangleright 010$

Creating a Deterministic Finite Automata (DFA): 5 / 14
Running the Machine on an Input

Computing Problem Solved by a DFA

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$

$q_0 \xrightarrow{010} q_0$
Running the Machine on an Input

0 1 0

0 1 0

$q_0\rightarrow 010$

$q_0\rightarrow 010 \implies M_q_1\rightarrow 10$

$q_0\rightarrow 010 \implies M_q_1\rightarrow 10$

($M$ is the name of our “Machine”)

Creator: Malik Magdon-Ismail

Deterministic Finite Automata (DFA): 5 / 14

Computing Problem Solved by a DFA
Running the Machine on an Input

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

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

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

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

$(M \text{ is the name of our "Machine"})$

$q_0 \xrightarrow{010} q_1 \xrightarrow{010} q_1 \xrightarrow{010}$

$q_0 \xrightarrow{010} M \xrightarrow{010} q_1 \xrightarrow{010}$
Running the Machine on an Input

$010 \uparrow$

$\downarrow$

$010 \uparrow$

$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$\hspace{1cm} $q_0 \xrightarrow{010}$

$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$\hspace{1cm} $q_0 \xrightarrow{010} \xrightarrow{M} q_1 \xrightarrow{01} q_2$

$q_0 \xrightarrow{010}$

$q_0 \xrightarrow{010} \xrightarrow{M} q_1 \xrightarrow{01} q_2$

$q_0 \xrightarrow{010} \xrightarrow{M} q_1 \xrightarrow{01} q_2 \xrightarrow{0}$

$q_0 \xrightarrow{010} \xrightarrow{M} q_1 \xrightarrow{01} q_2 \xrightarrow{0}$
Running the Machine on an Input

$q_0 \not\rightarrow 010$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$

$q_0 \not\rightarrow 010 \rightarrow q_1 \not\rightarrow 10 \rightarrow q_2 \not\rightarrow 01 \rightarrow q_0$
Running the Machine on an Input

$q_0 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$

$q_0 \rightarrow 010 \quad \Rightarrow \quad q_1 \rightarrow 010$
Running the Machine on an Input

\[ q_0 \xrightarrow{0} q_1 \]

\[ q_0 \xrightarrow{1} q_2 \]

\[ q_0 \xrightarrow{0} q_1 \]

\[ q_0 \xrightarrow{1} q_2 \]

\[ q_0 \xrightarrow{0} q_1 \]

\[ q_0 \xrightarrow{1} q_2 \]

\[ \text{NO}, \ REJECT \]
Running the Machine on an Input

Pop Quiz. Give computation trace for $\varepsilon$, 010, 000. What strings does the machine ACCEPT and say \textbf{YES}?
Running the Machine on an Input

Pop Quiz. Give computation trace for $\varepsilon$, 010, 000. What strings does the machine ACCEPT and say $\text{YES}$?

Pop Quiz. Determine $\text{YES}$ or $\text{NO}$ if you can from partial traces. $q_0 \rightarrow ? \rightarrow 0000$; $q_1 \rightarrow ? \rightarrow 0000$; $q_2 \rightarrow ? \rightarrow 0000$. 

Creator: Malik Magdon-Ismail
The computing problem solved by $M$ is the language $\mathcal{L}(M) = \{ w \mid M(w) = \text{YES} \}$. 
The computing problem solved by $M$ is the language $\mathcal{L}(M) = \{ w \mid M(w) = \text{YES} \}$.

$\mathcal{L}(M)$ is the automaton’s YES-set.
The computing problem solved by $M$ is the language $\mathcal{L}(M) = \{ w \mid M(w) = \text{YES} \}$.

$\mathcal{L}(M)$ is the automaton’s [YES]-set. For our automaton $M$

$$\mathcal{L}(M) = \{0, 00, 000, 0000, \ldots\} = \{0^n \mid n > 0\}.$$
The computing problem solved by $M$ is the language $\mathcal{L}(M) = \{w \mid M(w) = \text{YES}\}$.

$\mathcal{L}(M)$ is the automaton’s [YES]-set. For our automaton $M$

$$\mathcal{L}(M) = \{0, 00, 000, 0000, \ldots\} = \{0^n \mid n > 0\}.$$

1. For an automaton $M$, what is the computing problem $\mathcal{L}(M)$ solved by $M$?

**Practice.** Exercise 24.2 gives you lots of training in (1).
The computing problem solved by \( M \) is the language \( \mathcal{L}(M) = \{ w \mid M(w) = \text{YES} \} \).

\( \mathcal{L}(M) \) is the automaton’s \( \text{YES} \)-set. For our automaton \( M \)

\[
\mathcal{L}(M) = \{0, 00, 000, 0000, \ldots\} = \{0^n \mid n > 0\}.
\]

1. For an automaton \( M \), what is the computing problem \( \mathcal{L}(M) \) solved by \( M \)?

2. For a computing problem \( \mathcal{L} \), what automaton \( M \) solves \( \mathcal{L} \), i.e., \( \mathcal{L}(M) = \mathcal{L} \)?

Practice. Exercise 24.2 gives you lots of training in (1).
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

![Diagram of the vending machine with states labeled 0¢, 5¢, 10¢, 15¢, and 20¢. Arrows indicate transitions: 0¢ → 5¢, 5¢ → 10¢, 10¢ → 15¢, 15¢ → 20¢, 20¢ → 0¢. The transitions are labeled as 5¢ transition and 5¢ transition plus dispense soda.]
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

The vending machine has the following states:
- 0¢
- 5¢
- 10¢
- 15¢
- 20¢

- From 0¢, 5¢ transition is possible.
- From 5¢, 5¢ transition plus dispense soda is possible.
- From 10¢, 10¢ transition is possible.
- From 15¢, 5¢ transition is possible.
- From 20¢, 5¢ transition is possible.

The diagram illustrates the transitions between these states.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

![Diagram of the vending machine]

- 5¢ transition
- 5¢ transition plus dispense soda
- 10¢ transition
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

```
0¢  →  5¢ transition
10¢  →  5¢ transition plus dispense soda
20¢  →  10¢ transition
15¢  →  10¢ transition plus dispense soda

Creator: Malik Magdon-Ismail
```
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢,

0¢ → 10¢.
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢, 10¢,
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢, 10¢, 5¢,

\[ 0¢ \rightarrow 10¢ \rightarrow 20¢ \rightarrow 0¢ \text{ (} + \text{ soda) } \]
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢, 10¢, 5¢, 10¢,

\[ 0¢ \rightarrow 10¢ \rightarrow 20¢ \rightarrow 0¢ \ (\text{+ soda}) \rightarrow 10¢ \]
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢, 10¢, 5¢, 10¢, 10¢,

0¢  →  10¢  →  20¢  →  0¢ ( + soda )  →  10¢  →  20¢ .
The Vending Machine

Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

Input sequence: 10¢, 10¢, 5¢, 10¢, 10¢, 10¢.

$0¢ \rightarrow 10¢ \rightarrow 20¢ \rightarrow 0¢ \; (+ \; \text{soda}) \rightarrow 10¢ \rightarrow 20¢ \rightarrow 5¢ \; (+ \; \text{soda})$. 
\[ \mathcal{L} = \{10\}. \]
DFA for a Finite Language

$L = \{10\}.$
\[ \mathcal{L} = \{10\}. \]

- 0 means move to a rejecting error state.
$\mathcal{L} = \{10\}.$

- 0 means move to a rejecting error state.
DFA for a Finite Language

\[ L = \{10\} \].

- 0 means move to a rejecting error state.
- 1 is partial success.
DFA for a Finite Language

\[ \mathcal{L} = \{10\}. \]

- 0 means move to a rejecting error state.
- 1 is partial success.
- Another 1 puts you into error since you want 0;
DFA for a Finite Language

$\mathcal{L} = \{10\}$.

- 0 means move to a rejecting error state.
- 1 is partial success.
- Another 1 puts you into error since you want 0;
- 0 from $q_1$ and you are ready to accept ... unless ...
DFA for a Finite Language

\[ \mathcal{L} = \{10\}. \]

- 0 means move to a rejecting error state.
- 1 is partial success.
- Another 1 puts you into error since you want 0;
- 0 from \( q_1 \) and you are ready to accept ... unless ...
- More bits arrive, in which case move to the rejecting error state.
$L = \{10\}$.

- 0 means move to a rejecting error state.
- 1 is partial success.
- Another 1 puts you into error since you want 0;
- 0 from $q_1$ and you are ready to accept ...unless ...
- More bits arrive, in which case move to the rejecting error state.

**Practice.** Try random strings other than 01 and make sure our DFA rejects them.
DFAs for Infinite Languages

\[ \mathcal{L}_1 = *0* \quad \mathcal{L}_2 = *1 \quad \text{ (wildcard } * = \Sigma^*) \]
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ = \{ \text{strings ending in 1} \} \]

(wildcard \( \ast = \Sigma^* \))
DFAs for Infinite Languages

\[ \mathcal{L}_1 = *0* \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = *1 \]
\[ (\text{wildcard } * = \Sigma^*) \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a } 0 \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ = \{ \text{strings ending in } 1 \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

(wildcard \( \ast = \Sigma^* \))
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

\[ M_1 \]
DFAs for Infinite Languages

$$\mathcal{L}_1 = \ast 0 \ast$$
$$= \{ \text{strings with a 0} \}$$
$$= \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \}$$

$$\mathcal{L}_2 = \ast 1 \ast$$
$$= \{ \text{strings ending in 1} \}$$
$$= \{ 1, 01, 11, 001, 011, 101, 111, \ldots \}$$

(wildcard $$\ast = \Sigma^*$$)
DFAs for Infinite Languages

\[ \mathcal{L}_1 = *0* \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = *1 \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ (\text{wildcard} \ast = \Sigma^*) \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]
DFAs for Infinite Languages

\[ L_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ L_2 = \ast 1 \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

(wildcard \( \ast = \Sigma^* \))
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ (\text{wildcard } \ast = \Sigma^*) \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]
DFAs for Infinite Languages

$L_1 = \ast 0 \ast$
$= \{\text{strings with a 0}\}$
$= \{0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots\}$

$L_2 = \ast 1$
$= \{\text{strings ending in 1}\}$
$= \{1, 01, 11, 001, 011, 101, 111, \ldots\}$

(wildcard $\ast = \Sigma^*$)

$M_1$

$M_2$
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \quad \text{(wildcard } \ast = \Sigma^*) \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]
DFAs for Infinite Languages

\[ L_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ L_2 = \ast 1 \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

Complement. Consider \( \overline{L_1} \):

\[ M_1 \]

\[ M_2 \]
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast \{ \ast \} \]
\[ = \{ \text{strings with a } 0 \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ M_1 \]

\[ \mathcal{L}_2 = \ast \{ \ast \} \]
\[ = \{ \text{strings ending in } 1 \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

\[ M_2 \]

**Complement.** Consider \( \overline{\mathcal{L}_1} \): Must ACCEPT strings \( M_1 \) REJECTS.
DFAs for Infinite Languages

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

Complement. Consider \( \overline{\mathcal{L}_1} \): Must ACCEPT strings \( M_1 \) REJECTS.

\[ \mathcal{L}_1 = \ast 0 \ast \]
\[ = \{ \text{strings with a 0} \} \]
\[ = \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \} \]

\[ \mathcal{L}_2 = \ast 1 \]
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

\[ M_1 \]

\[ M_2 \]

\[ \leftarrow \text{flip [YES] and [NO]-states.} \]
Two DFAs in One

\[ L_1 = \ast 0 \ast \]

\[ M_1 \]

1

0

q_0 \rightarrow 0

q_1

\[ L_2 = \ast 1 \]

\[ M_2 \]

0

1

s_0 \rightarrow 1

s_1

(wildcard \( \ast = \Sigma^* \))
Two DFAs in One

\[ \mathcal{L}_1 = \*0\* \]

\[ \mathcal{L}_2 = \*1 \]

(wildcard \( \* = \Sigma^* \))

\[ M_1 \]

\[ M_2 \]

The Joint-DFA:
Two DFAs in One

\[ \mathcal{L}_1 = \ast 0 \ast \]

The Automaton \( M_1 \):

\[ M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \]

\[ \mathcal{L}_2 = \ast 1 \]

The Automaton \( M_2 \):

\[ \text{(wildcard } \ast = \Sigma^*) \]

The Joint-DFA:

\[ q_0 s_0: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \]
Two DFAs in One

\( \mathcal{L}_1 = *0* \)

\( \mathcal{L}_2 = *1 \)  

(wildcard \( * = \Sigma^* \))

The Joint-DFA:

\( q_0s_0: M_1 \) is in state \( q_0 \) and \( M_2 \) is in state \( s_0 \).

\( q_0s_1: M_1 \) is in state \( q_0 \) and \( M_2 \) is in state \( s_1 \).
Two DFAs in One

\[ \mathcal{L}_1 = *0* \]

\[ \mathcal{L}_2 = *1 \]

\((\text{wildcard } \ast = \Sigma^*)\)

\(M_1\)

The Joint-DFA:

\(q_0s_0: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0.\)

\(q_0s_1: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_1.\)

\(q_1s_0: M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_0.\)
Two DFAs in One

\[ L_1 = *0* \]

\[ L_2 = *1 \]  \hspace{1cm} \text{(wildcard } * = \Sigma^*)

\[ M_1 \]

\[ M_2 \]

The Joint-DFA:

\[ q_0s_0: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \]

\[ q_0s_1: M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_1. \]

\[ q_1s_0: M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_0. \]
Two DFAs in One

$\mathcal{L}_1 = *0*$

$\mathcal{L}_2 = *1$

(wildcard $* = \Sigma^*$)

$M_1$

The Joint-DFA:

$q_0s_0$: $M_1$ is in state $q_0$ and $M_2$ is in state $s_0$.

$q_0s_1$: $M_1$ is in state $q_0$ and $M_2$ is in state $s_1$.

$q_1s_0$: $M_1$ is in state $q_1$ and $M_2$ is in state $s_0$.

$q_1s_1$: $M_1$ is in state $q_1$ and $M_2$ is in state $s_1$. 

Creator: Malik Magdon-Ismail
Two DFAs in One

\[ \mathcal{L}_1 = \ast 0 \ast \]

\[ \mathcal{L}_2 = \ast 1 \]

(wildcard \( \ast = \Sigma^* \))

The Joint-DFA:

\[ q_0 s_0 : M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \]

\[ q_0 s_1 : M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_1. \]

\[ q_1 s_0 : M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_0. \]

\[ q_1 s_1 : M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_1. \]
Two DFAs in One

\[ L_1 = \ast 0 \ast \]

\[ L_2 = \ast 1 \]

\( \ast = \Sigma^* \) (wildcard)

\begin{align*}
M_1 & : \\
q_0 & \rightarrow 0 \rightarrow q_1 \\
0 & \rightarrow q_0 \rightarrow 1 \rightarrow q_1
\end{align*}

\begin{align*}
M_2 & : \\
s_0 & \rightarrow 1 \rightarrow s_1 \\
0 & \rightarrow s_0 \rightarrow 1
\end{align*}

The Joint-DFA:

\begin{align*}
q_0s_0 & : M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \\
q_0s_1 & : M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_1. \\
q_1s_0 & : M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_0. \\
q_1s_1 & : M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_1.
\end{align*}

Pop Quiz.

Run the joint and individual DFAs for 0100. What are the final states of each DFA?
Two DFAs in One

\[ \mathcal{L}_1 = *0* \]
\[ \mathcal{L}_2 = *1 \]

(wildcard \( * = \Sigma^* \))

The Joint-DFA:

\[ q_0s_0: \text{ } M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_0. \]
\[ q_0s_1: \text{ } M_1 \text{ is in state } q_0 \text{ and } M_2 \text{ is in state } s_1. \]
\[ q_1s_0: \text{ } M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_0. \]
\[ q_1s_1: \text{ } M_1 \text{ is in state } q_1 \text{ and } M_2 \text{ is in state } s_1. \]

Pop Quiz.

1. Run the joint and individual DFAs for 0100. What are the final states of each DFA?

2. If you want to solve \( \mathcal{L}_1 \cup \mathcal{L}_2 \), what should the accept states of the joint-DFA be?
Two DFAs in One

\[ L_1 = *0* \]

\[ L_2 = *1 \]  

(wildcard * = \( \Sigma^* \))

\[ M_1 \]

\[ M_2 \]

The Joint-DFA:

\( q_0s_0 \): \( M_1 \) is in state \( q_0 \) and \( M_2 \) is in state \( s_0 \).

\( q_0s_1 \): \( M_1 \) is in state \( q_0 \) and \( M_2 \) is in state \( s_1 \).

\( q_1s_0 \): \( M_1 \) is in state \( q_1 \) and \( M_2 \) is in state \( s_0 \).

\( q_1s_1 \): \( M_1 \) is in state \( q_1 \) and \( M_2 \) is in state \( s_1 \).

Pop Quiz.

1. Run the joint and individual DFAs for 0100. What are the final states of each DFA?
2. If you want to solve \( L_1 \cup L_2 \), what should the accept states of the joint-DFA be?
3. If you want to solve \( L_1 \cap L_2 \), what should the accept states of the joint-DFA be?
The Power of DFAs: What can they Solve?
The Power of DFAs: What can they Solve?

- Finite languages.
  (building blocks of regular expressions)
The Power of DFAs: What can they Solve?

- Finite languages.
  (building blocks of regular expressions)

- Complement, intersection, union.
  (operations to form complex regular expressions)
The Power of DFAs: What can they Solve?

- Finite languages.
  (building blocks of regular expressions)

- Complement, intersection, union.
  (operations to form complex regular expressions)

- Concatenation and Kleene-star (little more complicated, see text).
  (operations to form complex regular expressions)

That’s what we need for regular expressions.
The Power of DFAs: What can they Solve?

- Finite languages.
  (building blocks of regular expressions)

- Complement, intersection, union.
  (operations to form complex regular expressions)

- Concatenation and Kleene-star (little more complicated, see text).
  (operations to form complex regular expressions)

That’s what we need for regular expressions.

DFAs solve languages (computing problems) expressed as regular expressions.

(That is why the languages solved by DFAs are called regular languages.)
Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).
Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n \cdot 1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
\mathcal{L}_{0^n1^n} = \{0^n \cdot 1^n \mid n \geq 0\}.
\]
**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n \cdot 1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[ \mathcal{L}_{0^n1^n} = \{0^n \cdot 1^n \mid n \geq 0\} \]

**Theorem.** There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)
Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
\mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)

**Proof.** Contradiction.
Is There Anything DFAs Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
\mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

Creator: Malik Magdon-Ismail
Is There Anything DFAs Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[ L_{0^n1^n} = \{0^n1^n \mid n \geq 0\}. \]

**Theorem.** There is no DFA that solves \( L_{0^n1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?
Pop Quiz. Give a DFA to solve $\{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\}$.

What about “equality,”

$$\mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.$$

Theorem. There is no DFA that solves $\mathcal{L}_{0^n1^n}$

Proof. Contradiction. Suppose a DFA $M$ with $k$ states solves $\{0^n1^n\}$. What happens to this DFA when you keep feeding it 0’s?

$q_0 = \text{state}(0^0)$
Is There Anything DFAs Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n 1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
L_{0^n 1^n} = \{0^n 1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( L_{0^n 1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n 1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1)
\]
Is There Anything DF As Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n \cdot 1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
L_{0^n 1^n} = \{0^n \cdot 1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( L_{0^n 1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n 1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2)
\]
Pop Quiz. Give a DFA to solve $\{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\}$.

What about “equality,”

$$L_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.$$

Theorem. There is no DFA that solves $L_{0^n1^n}$

Proof. Contradiction. Suppose a DFA $M$ with $k$ states solves $\{0^n1^n\}$.
What happens to this DFA when you keep feeding it 0’s?

$q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \ldots \xrightarrow{M} \text{state}(0^{k-1})$
Is There Anything DFAs Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
L_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( L_{0^n1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\]
Is There Anything DF As Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
\mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\]

After \( k \) 0’s, \( k + 1 \) states visited.
Is There Anything DFAs Can’t Solve?

Pop Quiz. Give a DFA to solve $\{0\}^* \cdot \{1\}^* = \{0^n 1^k | n \geq 0, k \geq 0\}$.

What about “equality,”

$$\mathcal{L}_{0^n 1^n} = \{0^n 1^n | n \geq 0\}.$$

**Theorem.** There is no DFA that solves $\mathcal{L}_{0^n 1^n}$

**Proof.** Contradiction. Suppose a DFA $M$ with $k$ states solves $\{0^n 1^n\}$.

What happens to this DFA when you keep feeding it 0’s?

$$q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)$$

After $k$ 0’s, $k + 1$ states visited. There must be a repetition (pigeonhole).
Is There Anything DFAs Can’t Solve?

**Pop Quiz.** Give a DFA to solve $\{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\}$.

What about “equality,”

$$L_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.$$  

**Theorem.** There is no DFA that solves $L_{0^n1^n}$

**Proof.** Contradiction. Suppose a DFA $M$ with $k$ states solves $\{0^n1^n\}$.

What happens to this DFA when you keep feeding it 0’s?

$$q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)$$

After $k$ 0’s, $k + 1$ states visited. There must be a repetition (pigeonhole).

$$\text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k.$$
Is There Anything DFAs Can’t Solve?

Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k | n \geq 0, k \geq 0\} \).

What about “equality,”

\[ \mathcal{L}_{0^n1^n} = \{0^n1^n | n \geq 0\} \]

Theorem. There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)

Proof. Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\( q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k) \)

After \( k \) 0’s, \( k + 1 \) states visited. There must be a repetition (pigeonhole).

\( \text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k. \)

Consider the two input strings \( 0^i1^i \in \mathcal{L}_{0^n1^n} \) and \( 0^i1^i \notin \mathcal{L}_{0^n1^n} \).
Pop Quiz. Give a DFA to solve $\{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\}$.

What about “equality,”

$$\mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.$$

**Theorem.** There is no DFA that solves $\mathcal{L}_{0^n1^n}$

**Proof.** Contradiction. Suppose a DFA $M$ with $k$ states solves $\{0^n1^n\}$.

What happens to this DFA when you keep feeding it 0’s?

$$q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)$$

After $k$ 0’s, $k + 1$ states visited. There must be a repetition (pigeonhole).

$$\text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k.$$ 

Consider the two input strings $0^i1^i \in \mathcal{L}_{0^n1^n}$ and $0^j1^i \not\in \mathcal{L}_{0^n1^n}$.

After $M$ has processed the 0s in both strings, it is in state $q$, and the traces of the two computations are

$q \mid 0^i \triangleright 1^i$ \quad and \quad $q \mid 0^j \triangleright 1^i$. 

$\square$
Is There Anything DFAs Can’t Solve?

**Pop Quiz.** Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n 1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
L_{0^n 1^n} = \{0^n 1^n \mid n \geq 0\}.
\]

**Theorem.** There is no DFA that solves \( L_{0^n 1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n 1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\]

After \( k \) 0’s, \( k + 1 \) states visited. There must be a repetition (pigeonhole).

\[
\text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k.
\]

Consider the two input strings \( 0^i 1^i \in L_{0^n 1^n} \) and \( 0^j 1^i \not\in L_{0^n 1^n} \).

After \( M \) has processed the 0s in both strings, it is in state \( q \), and the traces of the two computations are

\[
q \xrightarrow{1^i} \quad \text{and} \quad q \xrightarrow{1^i}.
\]

Same number of 1’s remain, from state \( q \).
Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[
L_{0^n1^n} = \{0^n1^n \mid n \geq 0\}.
\]

Theorem. There is no DFA that solves \( L_{0^n1^n} \).

Proof. Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\]

After \( k \) 0’s, \( k + 1 \) states visited. There must be a repetition (pigeonhole).

\[
\text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k.
\]

Consider the two input strings \( 0^i1^i \in L_{0^n1^n} \) and \( 0^j1^i \not\in L_{0^n1^n} \).

After \( M \) has processed the 0s in both strings, it is in state \( q \), and the traces of the two computations are

\[
q \mid \square \rightarrow 1^i \quad \text{and} \quad q \mid \square \rightarrow 1^i.
\]

Same number of 1’s remain, from state \( q \). Either both rejected or both accepted. \textbf{FISHY!}
Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

\[ \mathcal{L}_{0^n1^n} = \{0^n1^n \mid n \geq 0\} \]

**Theorem.** There is no DFA that solves \( \mathcal{L}_{0^n1^n} \)

**Proof.** Contradiction. Suppose a DFA \( M \) with \( k \) states solves \( \{0^n1^n\} \).

What happens to this DFA when you keep feeding it 0’s?

\[
\begin{align*}
q_0 &= \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\end{align*}
\]

After \( k \) 0’s, \( k + 1 \) states visited. There must be a repetition (pigeonhole).

\[
\text{state}(0^i) = \text{state}(0^j) = q, \quad i < j \leq k.
\]

Consider the two input strings \( 0^i1^i \in \mathcal{L}_{0^n1^n} \) and \( 0^j1^i \notin \mathcal{L}_{0^n1^n} \).

After \( M \) has processed the 0s in both strings, it is in state \( q \), and the traces of the two computations are

\[
q \mid \square \overrightarrow{1^i} \quad \text{and} \quad q \mid \square \overrightarrow{1^i}.
\]

Same number of 1’s remain, from state \( q \). Either both rejected or both accepted. **FISHY!**

**Intuition:** The DFA has no “memory” to remember \( n \).
DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.

Computing Model

Rules to:
1. Construct machine;
2. Solve problems.
Our First Computing Machine

DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.

<table>
<thead>
<tr>
<th>Computing Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules to:</td>
</tr>
<tr>
<td>1. Construct machine;</td>
</tr>
<tr>
<td>2. Solve problems.</td>
</tr>
</tbody>
</table>

| Analyze Model |
DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.
Our First Computing Machine

DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.

**Computing Model**
Rules to:
1. Construct machine;
2. Solve problems.

**Analyze Model**
1. *Capabilities*: what *can* be solved?
2. *Limitations*: what *can’t* be solved?
Our First Computing Machine

DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.

Computing Model
Rules to:
1. Construct machine;
2. Solve problems.

Analyze Model
1. Capabilities: what can be solved?
2. Limitations: what can’t be solved?

Do we need a new model?
DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.

**Computing Model**
Rules to:
1. Construct machine;
2. Solve problems.

**Analyze Model**
1. *Capabilities*: what can be solved?
2. *Limitations*: what can’t be solved?

Do we need a new model?

DFAs fail at so simple a problem as equality.
Our First Computing Machine

DFAs can be implemented using basic technology, so practical.

Powerful, but also limited.

**Computing Model**

Rules to:
1. Construct machine;
2. Solve problems.

**Analyze Model**

1. *Capabilities*: what *can* be solved?
2. *Limitations*: what *can’t* be solved?

DFAs fail at so simple a problem as equality.
- That’s not acceptable.
- We need a more powerful machine.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet (pop)
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet (pop);
- push something new onto the top of the stack.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet (pop)
- push something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a pushdown automaton (PDA)
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet (pop);
- push something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a pushdown automaton (PDA)

How does the stack help to solve \( \{0^n1^n \mid n \geq 0\} \)?
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet (pop)
- push something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a pushdown automaton (PDA)

How does the stack help to solve \( \{0^n1^n \mid n \geq 0\} \)?

1. When you read in each 0, write it to the stack.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:
- see the top sheet;
- remove the top sheet (pop)
- push something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a pushdown automaton (PDA)

How does the stack help to solve \( \{0^n1^n \mid n \geq 0\} \)?

1: When you read in each 0, write it to the stack.
2: For each 1, pop the stack. At the end if the stack is empty, **ACCEPT**.
Adding Memory

DFAs have no scratch paper. It’s hard to compute entirely in your head.

Stack Memory. Think of a file-clerk with a stack of papers. The clerk’s capabilities:

- see the top sheet;
- remove the top sheet (pop)
- push something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a pushdown automaton (PDA)

How does the stack help to solve \( \{0^n1^n \mid n \geq 0\} \)?

1: When you read in each 0, write it to the stack.
2: For each 1, pop the stack. At the end if the stack is empty, ACCEPT.

The memory allows the automaton to “remember” \( n \).