Deterministic Finite Automata (DFA)

A Simple Computing Machine: A CPU with 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

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

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

- How hard is the problem?
- How complex is $\mathcal{L}$?
- How hard is it to test membership in $\mathcal{L}$?
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.

We focus on what can and can’t be solved on a particular kind of machine.
A simple computing machine.
- States.
- Transitions.
- No scratch paper.

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

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

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

![Diagram of a simple computing machine]

| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

Running the Machine
A Simple Computing Machine

Deterministic Finite Automata (DFA): 4 / 15

Running the Machine →
A Simple Computing Machine

### Running the Machine

<table>
<thead>
<tr>
<th>States</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$ (NO)</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$ (YES)</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_2$ (NO)</td>
</tr>
</tbody>
</table>

### Deterministic Finite Automata (DFA)

<table>
<thead>
<tr>
<th>Input</th>
<th>State Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0 \rightarrow q_0$ (NO)</td>
</tr>
<tr>
<td>1</td>
<td>$1 \rightarrow q_1$ (YES)</td>
</tr>
<tr>
<td>1</td>
<td>$1 \rightarrow q_2$ (NO)</td>
</tr>
<tr>
<td>1</td>
<td>$1 \rightarrow q_1$ (YES)</td>
</tr>
<tr>
<td>0</td>
<td>$0 \rightarrow q_0$ (NO)</td>
</tr>
<tr>
<td>0</td>
<td>$0 \rightarrow q_0$ (NO)</td>
</tr>
<tr>
<td>0</td>
<td>$0 \rightarrow q_0$ (NO)</td>
</tr>
<tr>
<td>1</td>
<td>$1 \rightarrow q_1$ (YES)</td>
</tr>
<tr>
<td>0</td>
<td>$0 \rightarrow q_0$ (NO)</td>
</tr>
<tr>
<td>1</td>
<td>$1 \rightarrow q_1$ (YES)</td>
</tr>
<tr>
<td>0</td>
<td>$0 \rightarrow q_0$ (NO)</td>
</tr>
</tbody>
</table>
A Simple Computing Machine

Running the Machine

<table>
<thead>
<tr>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1: q_0 \ 0 \ q_1$ ↔ In state $q_0$, if you read 0, transition to $q_1$</td>
</tr>
</tbody>
</table>
A Simple Computing Machine

Deterministic Finite Automata (DFA): 4 / 15

Running the Machine →
A Simple Computing Machine

States

- $q_0$
- $q_1$
- $q_2$

Transitions

1: $q_0$ 0 $q_1$
2: $q_0$ 1 $q_2$
3: $q_1$ 0 $q_1$

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

Running the Machine

[[Diagram of the machine]]
A Simple Computing Machine

Running the Machine

states

<table>
<thead>
<tr>
<th>states</th>
<th>transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$ 0 $q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$ 0 $q_1$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_1$ 1 $q_2$</td>
</tr>
</tbody>
</table>

In state $q_0$, if you read 0, transition to $q_1$.
A Simple Computing Machine

### States

- **q₀**: 0
- **q₁**: 1
- **q₂**: 0,1

### Transitions

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

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

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

---

transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$q_0$, 0, $q_1$</td>
</tr>
<tr>
<td>2</td>
<td>$q_0$, 1, $q_2$</td>
</tr>
<tr>
<td>3</td>
<td>$q_1$, 0, $q_1$</td>
</tr>
<tr>
<td>4</td>
<td>$q_1$, 1, $q_1$</td>
</tr>
<tr>
<td>5</td>
<td>$q_2$, 0, $q_2$</td>
</tr>
<tr>
<td>6</td>
<td>$q_2$, 1, $q_2$</td>
</tr>
</tbody>
</table>

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

---

Running the Machine

Creator: Malik Magdon-Ismail
A Simple Computing Machine

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

---

Creator: Malik Magdon-Ismail

Deterministic Finite Automata (DFA): 4 / 15

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.
3: When done processing every bit, output **YES** if the final resting state of the DFA is a **YES**-state; otherwise output **NO**.

---

**Transitions**

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

---

**States**

- $q_0$ *NO*
- $q_1$ *YES*
- $q_2$ *NO*
Running the Machine on an Input

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

\[
q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_0 \\
q_1 \xrightarrow{0} q_0 \\
q_2 \xrightarrow{0,1} q_1 \\
q_0 \xrightarrow{\triangleright} 010
\]

\[q_0 \xrightarrow{\triangleright} 010\]
Running the Machine on an Input

$\begin{array}{c}
0 \\
\uparrow
\end{array}$

$q_0|\downarrow 010$

$\begin{array}{c}
0 \\
\uparrow
\end{array}$

$q_0|\downarrow 010 \Rightarrow M q_1|0\downarrow 10$

($M$ is the name of our “Machine”)
Running the Machine on an Input

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

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

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

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

$q_0 | \overrightarrow{010} \Rightarrow q_0 | \overrightarrow{010} \Rightarrow q_1 | \overrightarrow{0110} \Rightarrow q_2 | \overrightarrow{0111}$

$M$ is the name of our “Machine”
Running the Machine on an Input

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

\[q_0 \xrightarrow{10} q_2 \xrightarrow{01} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0,1} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{01} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{01} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{01} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{01} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \]
Running the Machine on an Input

Pop Quiz. Give computation trace for ε, 010, 000. What strings does the machine ACCEPT and say (YES)?

Pop Quiz. Determine (YES) or (NO) if you can from partial traces. $q_0 \xrightarrow{\cdot} 0000$; $q_1 \xrightarrow{\cdot} 0000$; $q_2 \xrightarrow{\cdot} 000$.

(NO), REJECT
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 $M$

$$\mathcal{L}(M) = \{0, 00, 000, 0000, \ldots \} = \{0^n \mid n > 0\}.$$
Computing Problem Solved by a DFA

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 $M$?

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 question 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¢.
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¢.
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¢.

Diagram:

- 0¢
- 5¢
- 10¢
- 15¢
- 20¢

Arrows:
- → 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¢.
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¢.
Vending machine takes nickels and dimes and dispenses a soda when it has 25¢.

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

Input sequence: 10¢, 10¢, 5¢, 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¢,
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¢ ➔ 10¢ ➔ 20¢ ➔ 0¢ (+ soda) ➔ 10¢ ➔ 20¢ ➔ 5¢ (+ soda).
$\mathcal{L} = \{10\}$. 
\( \mathcal{L} = \{10\} \).
DFA for a Finite Language

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

- 0 means move to a rejecting ERROR state and stay there.
\[ \mathcal{L} = \{10\}. \]

- 0 means move to a rejecting ERROR state and stay there
- 1 is partial success.
DFAs for Infinite Languages

$L = \{10\}$.

- 0 means move to a rejecting ERROR state and stay there.
- 1 is partial success.
- Another 1 puts you into ERROR since you want 0;
\[ \mathcal{L} = \{10\}. \]

- 0 means move to a rejecting ERROR state and stay there.
- 1 is partial success.
- Another 1 puts you into ERROR since you want 0;
- 0 from \( q_1 \) and you are ready to accept \ldots unless \ldots
DFA for a Finite Language

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

- 0 means move to a rejecting ERROR state and stay there
- 1 is partial success.
- Another 1 puts you into ERROR since you want 0;
- 0 from \( q_1 \) and you are ready to accept \ldots unless \ldots
- More bits arrive, in which case move to ERROR.

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

\[ L_1 = *0* \quad \quad L_2 = *1 \quad \quad (\text{wildcard } * = \Sigma^*) \]
DFAs for Infinite Languages

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

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

$$L_1 = \ast 0 \ast$$

$$= \{ \text{strings with a 0} \}$$

$$= \{ 0, 00, 01, 10, 000, 001, 010, 011, 100, \ldots \}$$

$$L_2 = \ast 1$$

$$= \{ \text{strings ending in 1} \}$$

$$= \{ 1, 01, 11, 001, 011, 101, 111, \ldots \}$$
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

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

\[ M_1 \]

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

\[ M_2 \]
DFAs for Infinite Languages

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

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

(\text{wildcard } * = \Sigma^*)
DFAs for Infinite Languages

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

\[ M_1 \]

\[ L_2 = \ast 1 \] (wildcard \( \ast = \Sigma^* \))
\[ = \{ \text{strings ending in 1} \} \]
\[ = \{ 1, 01, 11, 001, 011, 101, 111, \ldots \} \]

\[ M_2 \]

\textbf{Complement.} Consider \( \overline{L_1} \): Must \textsc{accept} strings \( M_1 \text{ rejects} \).
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.

\[ M \]

\[ \text{← flip (YES) and (NO)-states.} \]
Two DFAs in One: Union and Intersection

\[ L_1 = \ast 0 \ast \]

\[ L_2 = \ast 1 \]  

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

The Joint-DFA has product states \( \{ q_0s_0, q_0s_1, q_1s_0, q_1s_1 \} \):
Two DFAs in One: Union and Intersection

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

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

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

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):
Two DFAs in One: Union and Intersection

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

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

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

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):

- \( q_0s_0 \): \( M_1 \) 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: Union and Intersection

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

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

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

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):

- \( q_0s_0 \): \( M_1 \) 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 \( s_0 \).
Two DFAs in One: Union and Intersection

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

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

(wildcard \(* = \Sigma^*)

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):

- \( q_0s_0 \): \( M_1 \) 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 \( s_0 \).
Two DFAs in One: Union and Intersection

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

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

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

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):

- \( q_0s_0 \): \( M_1 \) 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 \( s_1 \).
Two DFAs in One: Union and Intersection

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

$L_1 = *0*$

1

$q_0$

$q_1$

0, 1

$M_1$

$L_2 = *1$ (wildcard $* = \Sigma^*$)

1

$s_0$

$s_1$

0

1

$M_2$

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):

$q_0s_0$: $M_1$ 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 $s_1$. 
Two DFAs in One: Union and Intersection

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

![DFA M1 diagram]

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

![DFA M2 diagram]

The Joint-DFA has product states \( \{q_0s_0, q_0s_1, q_1s_0, q_1s_1\} \):

- \(q_0s_0\): \(M_1\) 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 \(\varepsilon, 0100, 11, 101\). 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?
3. If you want to solve \(\mathcal{L}_1 \cap \mathcal{L}_2\), what should the accept states of the joint-DFA be?
Concatenation and Kleene Star

\[ \mathcal{L}_1 = \{1\} \quad (M_1) \]
\[ \mathcal{L}_2 = \{0\} \quad (M_2) \]
Concatenation and Kleene Star

\[ \mathcal{L}_1 = \{1\} \quad (M_1) \]

\[ \mathcal{L}_2 = \{0\} \quad (M_2) \]

\[ \mathcal{L}_1 \cdot \mathcal{L}_2: \]
Concatenation and Kleene Star

\[ L_1 = \{1\} \quad (M_1) \]

\[ L_2 = \{0\} \quad (M_2) \]

\[ L_1 \cdot L_2: \]
Concatenation and Kleene Star

$L_1 = \{1\}$  
$(M_1)$

$L_2 = \{0\}$  
$(M_2)$

$L_1 \cdot L_2$:  

$L_1^*$:
Concatenation and Kleene Star

\[ \mathcal{L}_1 = \{1\} \quad (M_1) \]

\[ \mathcal{L}_2 = \{0\} \quad (M_2) \]

\[ \mathcal{L}_1 \cdot \mathcal{L}_2: \]

\[ \mathcal{L}_1^*: \]
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 \( \{0\}^* \cdot \{1\}^* = \{0^n 1^k \mid n \geq 0, k \geq 0\} \).
Pop Quiz. Give a DFA to solve \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

What about “equality,”

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

\[ q_0 = \text{state}(0^0) \]
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?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1)
\]
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?

\[
q_0 = \text{state}(0^0) \xrightarrow{0^0} \text{state}(0^1) \xrightarrow{0^1} \text{state}(0^2)
\]
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?

\[ q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \cdots \xrightarrow{M} \text{state}(0^{k-1}) \]
Is There Anything DF As 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?

\[
q_0 = \text{state}(0^0) \xrightarrow{M} \text{state}(0^1) \xrightarrow{M} \text{state}(0^2) \xrightarrow{M} \ldots \xrightarrow{M} \text{state}(0^{k-1}) \xrightarrow{M} \text{state}(0^k)
\]
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).
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 \( \{0^n1^n\} \).

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

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

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 \( \{0\}^* \cdot \{1\}^* = \{0^n1^k \mid n \geq 0, k \geq 0\} \).

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 \( 0^j1^i \notin \mathcal{L}_{0^n1^n} \).
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. 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
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,”

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

**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 \( 0^j1^i \notin L_{0^n1^n} \).

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

\[
q \mid 0^i \xrightarrow{1^i} q \quad \text{and} \quad q \mid 0^j \xrightarrow{1^i}.
\]
Is There Anything DF As Can’t Solve?

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

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 \( 0^j1^i \notin \mathcal{L}_{0^n1^n} \).

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

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

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

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

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

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

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

**Intuition:** The DFA has no “memory” to remember \( n \).
Our First Computing Machine

DFAs can be implemented using basic technology, so practical.

Powerful (regular languages), but also limited.

**Computing Model**

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

DFAs can be implemented using basic technology, so practical.

Powerful (regular languages), 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 (regular languages), but also limited.

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

**Analyze Model**
- **Capabilities**: what **can** be solved?
- **Limitations**: what **can’t** be solved?

Do we need a new model?
Our First Computing Machine

DFAs can be implemented using basic technology, so practical.

Powerful (regular languages), but also limited.

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

### Computing Model
Rules to:
- Construct machine;
- Solve problems.

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

Do we need a new model?
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;
- 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.
2: For each 1, pop the stack. At the end if the stack is empty, ACCEPT.

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