CSE 135: Introduction to Theory of Computation
Deterministic Finite Automata

Sungjin Im
University of California, Merced
01-22-2015
Decision Problems

Decision Problems
Given input, decide “yes” or “no”

➤ **Examples:** Is \( x \) an even number? Is \( x \) prime? Is there a path from \( s \) to \( t \) in graph \( G \)?

➤ i.e., Compute a boolean function of input
Decision Problems

**Decision Problems**
Given input, decide "yes" or "no"

- **Examples:** Is $x$ an even number? Is $x$ prime? Is there a path from $s$ to $t$ in graph $G$?
- i.e., Compute a boolean function of input

**General Computational Problem**
In contrast, typically a problem requires computing some non-boolean function, or carrying out interactive/reactive computation in a distributed environment

- **Examples:** Find the factors of $x$. Find the balance in account number $x$. 
Decision Problems

Decision Problems
Given input, decide “yes” or “no”

▶ Examples: Is $x$ an even number? Is $x$ prime? Is there a path from $s$ to $t$ in graph $G$?
▶ i.e., Compute a boolean function of input

▶ Example of reduction to a decision problem
  ▶ Given a composite number $x$, find a factor greater than 1 and less than $x$.
  ▶ Does $x$ have a factor greater than $y$ and less than $z$?

▶ In this course, we will study decision problems because aspects of computability are captured by this special class of problems

General Computational Problem
In contrast, typically a problem requires computing some non-boolean function, or carrying out interactive/reactive computation in a distributed environment

▶ Examples: Find the factors of $x$. Find the balance in account number $x$. 
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
- As the program starts executing, some memory (a.k.a. state)
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
- As the program starts executing, some memory (a.k.a. state)
  - Includes the values of variables (and the “program counter”)

- State evolves throughout the computation
- Often, takes more memory for larger problem instances
- But some programs do not need larger state for larger instances!
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
- As the program starts executing, some memory (a.k.a. state)
  - Includes the values of variables (and the “program counter”)
  - State evolves throughout the computation
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
- As the program starts executing, some memory (a.k.a. state)
  - Includes the values of variables (and the “program counter”)
  - State evolves throughout the computation
  - Often, takes more memory for larger problem instances
What Does a Computation Look Like?

- Some code (a.k.a. control): the same for all instances
- The input (a.k.a. problem instance): encoded as a string over a finite alphabet
- As the program starts executing, some memory (a.k.a. state)
  - Includes the values of variables (and the “program counter”)
  - State evolves throughout the computation
  - Often, takes more memory for larger problem instances
- But some programs do not need larger state for larger instances!
Finite State Computation

- **Finite state**: A fixed upper bound on the size of the state, independent of the size of the input. They take boolean values (or values in a finite enumerated data type).

- Not enough memory to hold the entire input.

- "Streaming input": the automaton runs (i.e., changes state) on seeing each bit of input.
Finite State Computation

- **Finite state**: A fixed upper bound on the size of the state, independent of the size of the input. Take boolean values (or values in a finite enumerated data type).
- Not enough memory to hold the entire input.
Finite State Computation

- **Finite state**: A fixed upper bound on the size of the state, independent of the size of the input. They take boolean values (or values in a finite enumerated data type).
- Not enough memory to hold the entire input
  - “Streaming input”: automaton runs (i.e., changes state) on seeing each bit of input.
An Automatic Door

Front pad

Rear pad

door

Top view of Door
An Automatic Door

Top view of Door

- **Input**: A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...
An Automatic Door

Top view of Door

- **Input**: A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...

State diagram of controller
An Automatic Door

Top view of Door

- **Input**: A stream of events `<front>`, `<rear>`, `<both>`, `<neither>` ...

- Controller has a single bit of state.
Automaton

A finite automaton has:

- A finite set of states, with start/initial and accepting/final states;
- Transitions from one state to another on reading a symbol from the input.

Computation

Start at the initial state; in each step, read the next symbol of the input, take the transition (edge) labeled by that symbol to a new state.

Acceptance/Rejection: If after reading the input $w$, the machine is in a final state then $w$ is accepted; otherwise $w$ is rejected.

Transition Diagram of automaton
Finite Automata

**Automaton**

A finite automaton has: Finite set of states, with start/initial and accepting/final states;

```
Finite set of states,
with start/initial and accepting/final states;
```

Transition Diagram of automaton
Finite Automata

Automaton

A finite automaton has: Finite set of states, with start/initial and accepting/final states; Transitions from one state to another on reading a symbol from the input.

Transition Diagram of automaton
Finite Automata

Details

Automaton
A finite automaton has: Finite set of states, with start/initial and accepting/final states; Transitions from one state to another on reading a symbol from the input.

Computation
Start at the initial state; in each step, read the next symbol of the input, take the transition (edge) labeled by that symbol to a new state.

Transition Diagram of automaton
Finite Automata

Details

Automaton
A finite automaton has: Finite set of states, with start/initial and accepting/final states; Transitions from one state to another on reading a symbol from the input.

Computation
Start at the initial state; in each step, read the next symbol of the input, take the transition (edge) labeled by that symbol to a new state. Acceptance/Rejection: If after reading the input $w$, the machine is in a final state then $w$ is accepted; otherwise $w$ is rejected.

Transition Diagram of automaton
Example: Computation

- On input 1001, the computation is
  1. Start in state $q_0$. Read 1 and goto $q_1$.
  2. Read 0 and goto $q_1$.
  3. Read 0 and goto $q_1$.
  4. Read 1 and goto $q_0$. Since $q_0$ is not a final state, 1001 is rejected.

- On input 010, the computation is
  1. Start in state $q_0$. Read 0 and goto $q_0$.
  2. Read 1 and goto $q_1$.
  3. Read 0 and goto $q_1$.
  4. Since $q_1$ is a final state, 010 is accepted.
Example: Computation

On input 1001, the computation is
1. Start in state $q_0$. Read 1 and goto $q_1$.
2. Read 0 and goto $q_1$.
3. Read 0 and goto $q_1$.
4. Read 1 and goto $q_0$. Since $q_0$ is not a final state 1001 is rejected.

On input 010, the computation is
1. Start in state $q_0$. Read 0 and goto $q_0$.
2. Read 1 and goto $q_1$.
3. Read 0 and goto $q_1$. Since $q_1$ is a final state 010 is accepted.
Example I

Automaton accepts all strings of 0s and 1s
Example 1

Automaton accepts all strings of 0s and 1s
Example II

Automaton accepts strings ending in 1
Example II

Automaton accepts strings ending in 1
Example III

Automaton accepts strings having an odd number of 1s.
Example III

Automaton accepts strings having an odd number of 1s
Example IV

The automaton accepts strings having an odd number of 1s and odd number of 0s.
Automaton accepts strings having an odd number of 1s and odd number of 0s
Finite Automata in Practice

- grep: Unix command. grep “word to find” “file name”
- Thermostats
- Coke Machines
- Elevators
- Train Track Switches
- Security Properties
- Lexical Analyzers for Parsers
Definition
An alphabet is any finite, non-empty set of symbols. We will usually denote it by $\Sigma$.

Example
Examples of alphabets include $\{0, 1\}$ (binary alphabet); $\{a, b, \ldots, z\}$ (English alphabet); the set of all ASCII characters; $\{\text{moveforward}, \text{moveback}, \text{rotate90}\}$.
Strings

Definition
A string or word over alphabet $\Sigma$ is a (finite) sequence of symbols in $\Sigma$. Examples are ‘0101001’, ‘string’, ‘⟨moveback⟩⟨rotate90⟩’.
Strings

Definition
A string or word over alphabet $\Sigma$ is a (finite) sequence of symbols in $\Sigma$. Examples are ‘0101001’, ‘string’, ‘⟨moveback⟩⟨rotate90⟩’

$\epsilon$ is the empty string.
Strings

Definition
A string or word over alphabet $\Sigma$ is a (finite) sequence of symbols in $\Sigma$. Examples are ‘0101001’, ‘string’, ‘$\langle \text{moveback} \rangle \langle \text{rotate90} \rangle$’

- $\epsilon$ is the empty string.
- The length of string $u$ (denoted by $|u|$) is the number of symbols in $u$. Example, $|\epsilon| = 0$, $|011010| = 6$. 

Question: Is $\epsilon$ a prefix of ‘catnap’?
Strings

Definition
A string or word over alphabet $\Sigma$ is a (finite) sequence of symbols
in $\Sigma$. Examples are ‘0101001’, ‘string’, ‘⟨moveback⟩⟨rotate90⟩’

- $\epsilon$ is the empty string.
- The length of string $u$ (denoted by $|u|$) is the number of
  symbols in $u$. Example, $|\epsilon| = 0$, $|011010| = 6$.
- Concatenation: $uv$ is the string that has a copy of $u$ followed
  by a copy of $v$. Example, if $u = ‘cat’$ and $v = ‘nap’$ then
  $uv = ‘catnap’$. If $v = \epsilon$ the $uv = vu = u$. 
Strings

Definition
A string or word over alphabet $\Sigma$ is a (finite) sequence of symbols in $\Sigma$. Examples are ‘0101001’, ‘string’, ‘⟨moveback⟩⟨rotate90⟩’

- $\epsilon$ is the empty string.
- The length of string $u$ (denoted by $|u|$) is the number of symbols in $u$. Example, $|\epsilon| = 0$, $|011010| = 6$.
- Concatenation: $uv$ is the string that has a copy of $u$ followed by a copy of $v$. Example, if $u = ‘cat’$ and $v = ‘nap’$ then $uv = ‘catnap’$. If $v = \epsilon$ the $uv = vu = u$.
- $u$ is a prefix of $v$ if there is a string $w$ such that $v = uw$. Example ‘cat’ is a prefix of ‘catnap’. 
Definition
A string or word over alphabet $\Sigma$ is a (finite) sequence of symbols in $\Sigma$. Examples are ‘0101001’, ‘string’, ‘⟨moveback⟩⟨rotate90⟩’

- $\epsilon$ is the empty string.
- The length of string $u$ (denoted by $|u|$) is the number of symbols in $u$. Example, $|\epsilon| = 0$, $|011010| = 6$.
- Concatenation: $uv$ is the string that has a copy of $u$ followed by a copy of $v$. Example, if $u = 'cat'$ and $v = 'nap'$ then $uv = 'catnap'$ if $v = \epsilon$ the $uv = vu = u$.
- $u$ is a prefix of $v$ if there is a string $w$ such that $v = uw$. Example ‘cat’ is a prefix of ‘catnap’.
- Question: Is $\epsilon$ a prefix of ‘catnap’?
Languages

Definition

▶ For alphabet Σ, Σ∗ is the set of all strings over Σ. Σⁿ is the set of all strings of length n.

▶ A language over Σ is a set L ⊆ Σ∗. For example L = {1, 01, 11, 001} is a language over {0, 1}.

▶ A language L defines a decision problem: Inputs (strings) whose answer is 'yes' are exactly those belonging to L.
Languages

Definition

- For alphabet $\Sigma$, $\Sigma^*$ is the set of all strings over $\Sigma$. $\Sigma^n$ is the set of all strings of length $n$. 
Languages

Definition

- For alphabet $\Sigma$, $\Sigma^*$ is the set of all strings over $\Sigma$. $\Sigma^n$ is the set of all strings of length $n$.
- A language over $\Sigma$ is a set $L \subseteq \Sigma^*$. For example $L = \{1, 01, 11, 001\}$ is a language over $\{0, 1\}$. 
Languages

Definition

- For alphabet $\Sigma$, $\Sigma^*$ is the set of all strings over $\Sigma$. $\Sigma^n$ is the set of all strings of length $n$.
- A language over $\Sigma$ is a set $L \subseteq \Sigma^*$. For example $L = \{1, 01, 11, 001\}$ is a language over $\{0, 1\}$.
  - A language $L$ defines a decision problem:
Languages

Definition

- For alphabet $\Sigma$, $\Sigma^*$ is the set of all strings over $\Sigma$. $\Sigma^n$ is the set of all strings of length $n$.
- A language over $\Sigma$ is a set $L \subseteq \Sigma^*$. For example $L = \{1, 01, 11, 001\}$ is a language over $\{0, 1\}$.
  - A language $L$ defines a decision problem: Inputs (strings) whose answer is ‘yes’ are exactly those belonging to $L$.
Set Notation

We will often define languages using the set builder notation. Thus, $L = \{ w \in \Sigma^* \mid p(w) \}$ is the collection of all strings $w$ over $\Sigma$ that satisfy the property $p$. 
Set Notation

We will often define languages using the set builder notation. Thus, $L = \{ w \in \Sigma^* \mid p(w) \}$ is the collection of all strings $w$ over $\Sigma$ that satisfy the property $p$.

Example

- $L = \{ w \in \{0, 1\}^* \mid |w| \text{ is even} \}$ is
We will often define languages using the set builder notation. Thus, \( L = \{ w \in \Sigma^* \mid p(w) \} \) is the collection of all strings \( w \) over \( \Sigma \) that satisfy the property \( p \).

Example

\( L = \{ w \in \{0, 1\}^* \mid |w| \text{ is even} \} \) is the set of all even length strings over \( \{0, 1\} \).
Set Notation

We will often define languages using the set builder notation. Thus, \( L = \{ w \in \Sigma^* \mid p(w) \} \) is the collection of all strings \( w \) over \( \Sigma \) that satisfy the property \( p \).

Example

- \( L = \{ w \in \{0, 1\}^* \mid |w| \) is even\} is the set of all even length strings over \( \{0, 1\} \).
- \( L = \{ w \in \{0, 1\}^* \mid \) there is a \( u \) such that \( wu = 10001\} \) is
We will often define languages using the set builder notation. Thus, \( L = \{ w \in \Sigma^* \mid p(w) \} \) is the collection of all strings \( w \) over \( \Sigma \) that satisfy the property \( p \).

Example

- \( L = \{ w \in \{0, 1\}^* \mid |w| \text{ is even} \} \) is the set of all even length strings over \( \{0, 1\} \).
- \( L = \{ w \in \{0, 1\}^* \mid \text{there is a } u \text{ such that } wu = 10001 \} \) is the set of all prefixes of 10001.
To describe an automaton, we need to specify:

- What the alphabet is,
- What the states are,
- What the initial state is,
- What states are accepting/final, and
- What the transition from each state and input symbol is.

Thus, the above 5 things are part of the formal definition.
Finite Automata

Formal Definition

Definition

A finite automaton is $M = (Q, \Sigma, \delta, q_0, F)$, where

- $Q$ is the finite set of states
- $\Sigma$ is the finite alphabet
- $\delta : Q \times \Sigma \to Q$ “Next-state” transition function
- $q_0 \in Q$ initial state
- $F \subseteq Q$ final/accepting states
Deterministic Finite Automata

Formal Definition

Definition
A deterministic finite automaton (DFA) is \( M = (Q, \Sigma, \delta, q_0, F) \), where

- \( Q \) is the finite set of states
- \( \Sigma \) is the finite alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) “Next-state” transition function
- \( q_0 \in Q \) initial state
- \( F \subseteq Q \) final/accepting states

Given a state and a symbol, the next state is “determined”. 
Definition
For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function
\( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \) such that $\hat{\delta}(q, w)$ is $M$’s state after reading $w$ from state $q$. 

Definition
We say a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts string $w \in \Sigma^*$ iff $\hat{\delta}(q_0, w) \in F$. 

Computation
Computation

Definition
For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function
\( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \) such that \( \hat{\delta}(q, w) \) is $M$’s state after reading $w$ from state $q$. Formally,

\[
\hat{\delta}(q, w) = \begin{cases} 
q & \text{if } w = \epsilon \\
\delta(\hat{\delta}(q, u), a) & \text{if } w = ua
\end{cases}
\]
Definition
For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ such that $\hat{\delta}(q, w)$ is $M$’s state after reading $w$ from state $q$. Formally,

\[
\hat{\delta}(q, w) = \begin{cases} 
q & \text{if } w = \epsilon \\
\delta(\hat{\delta}(q, u), a) & \text{if } w = ua
\end{cases}
\]
Computation

Definition
For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ such that $\hat{\delta}(q, w)$ is $M$'s state after reading $w$ from state $q$. Formally,

$$\hat{\delta}(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ \delta(\hat{\delta}(q, u), a) & \text{if } w = ua \end{cases}$$
Computation

Definition
For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, let us define a function
$
\hat{\delta} : Q \times \Sigma^* \rightarrow Q \text{ such that } \hat{\delta}(q, w) \text{ is } M's \text{ state after reading } w \text{ from state } q.
$
Formally,

$$
\hat{\delta}(q, w) = \begin{cases} 
q & \text{if } w = \epsilon \\
\delta(\hat{\delta}(q, u), a) & \text{if } w = ua
\end{cases}
$$

Definition
We say a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts string $w \in \Sigma^*$ iff
$\hat{\delta}(q_0, w) \in F$. 
Acceptance/Recognition

**Definition**
The language accepted or recognized by a DFA $M$ over alphabet $\Sigma$ is $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. 

A language $L$ is said to be accepted/recognized by $M$ if $L = L(M)$. 

A language $L$ is regular if there is some DFA $M$ such that $L = L(M)$. 
Definition

The language accepted or recognized by a DFA $M$ over alphabet $\Sigma$ is $L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$. A language $L$ is said to be accepted/recognized by $M$ if $L = L(M)$. 
Acceptance/Recognition and Regular Languages

Definition
The language accepted or recognized by a DFA $M$ over alphabet $\Sigma$ is $L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$. A language $L$ is said to be accepted/recognized by $M$ if $L = L(M)$.

Definition
A language $L$ is regular if there is some DFA $M$ such that $L = L(M)$.
Formal Example of DFA

Example

Transition Diagram of DFA

Formally the automaton is $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ where

\[
\begin{align*}
\delta(q_0, 0) &= q_0 & \delta(q_0, 1) &= q_1 \\
\delta(q_1, 0) &= q_1 & \delta(q_1, 1) &= q_0
\end{align*}
\]

\[
\begin{align*}
\hat{\delta}(q_0, 010) &= \delta(\hat{\delta}(q_0, 01), 0) = \delta(\delta(\hat{\delta}(q_0, 0), 1), 0) \\
= \delta(\delta(q_0, 0), 10) &= \hat{\delta}(\delta(q_0, 0), 10) = \hat{\delta}(q_0, 10) \\
= \hat{\delta}(\delta(q_0, 1), 0) &= \hat{\delta}(q_1, 0) = \delta(q_1, 0) = q_1
\end{align*}
\]
Formal Example of DFA

Example

Transition Diagram of DFA

Formally the automaton is $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ where

- $\delta(q_0, 0) = q_0$
- $\delta(q_0, 1) = q_1$
- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_0$

$\hat{\delta}(q_0, 010) = \delta(\hat{\delta}(q_0, 01), 0) = \delta(\delta(\hat{\delta}(q_0, 0), 1), 0)$
$= \delta(\delta(q_0, 0), 0) = \hat{\delta}(q_0, 01) = \hat{\delta}(q_0, 10)$
$= \hat{\delta}(q_0, 10) = \delta(q_0, 0) = \delta(q_1, 0) = q_1$

Transition Table representation

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
</tbody>
</table>
A Simple Observation about DFAs

Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

But you can see that this is true by observing that $\hat{\delta}(q, u_1u_2u_3...u_k) = \delta(\delta(...(\delta(q, u_1), u_2), u_3), ...), u_{k-1}), u_k)$.
Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

Proof.

By induction! Let’s see . . .

But you can see that this is true by observing that

$\hat{\delta}(q, u_1u_2u_3...u_k) = \delta(\delta(...(\delta(q, u_1), u_2), u_3), ...), u_{k-1}, u_k)$
Domino Principle

- Line up $n$ dominoes numbered 0, 1, ... $n - 1$ such that if we knock one, the next one will fall.
- If $F_i$ denotes “$i$th domino falls”, we have $F_i \rightarrow F_{i+1}$
Domino Principle

- Line up $n$ dominoes numbered 0, 1, ..., $n-1$ such that if we knock one, the next one will fall.
- If $F_i$ denotes “$i$th domino falls”, we have $F_i \rightarrow F_{i+1}$.
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0$. 
Domino Principle

- Line up $n$ dominoes numbered 0, 1, \ldots, $n - 1$ such that if we knock one the next one will fall.
- If $F_i$ denotes "$i$th domino falls", we have $F_i \rightarrow F_{i+1}$
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0 \rightarrow F_1$
Domino Principle

- Line up $n$ dominoes numbered $0, 1, \ldots n-1$ such that if we knock one the next one will fall.
- If $F_i$ denotes “$i$th domino falls”, we have $F_i \rightarrow F_{i+1}$.
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0 \rightarrow F_1 \rightarrow F_2$. 
Domino Principle

- Line up $n$ dominoes numbered $0, 1, \ldots, n-1$ such that if we knock one the next one will fall.
- If $F_i$ denotes “$i$th domino falls”, we have $F_i \rightarrow F_{i+1}$.
- Thus, knocking the 0th domino will cause all the dominoes to fall because $F_0 \rightarrow F_1 \rightarrow F_2 \rightarrow \cdots \rightarrow F_{n-1}$.
Plato’s Infinite Domino Principle

Principle

Imagine one domino for each natural number 0, 1, 2, ..., arranged in an infinite row. Knocking the 0th domino will knock them all.

Plato
Plato’s Infinite Domino Principle

Principle
Imagine one domino for each natural number 0, 1, 2, ..., arranged in an infinite row. Knocking the 0th domino will knock them all.

“Proof”
Suppose they don’t all fall. Let $k > 0$ be the smallest numbered domino that remains standing. This means domino $k - 1$ fell. But then $k - 1$ will knock $k$ over. Therefore, $k$ must fall and remain standing, which is a contradiction.
Plato’s Infinite Domino Principle

Formally

Mathematically we can say

- $F_i$: $i$th domino falls
- Suppose for every natural number $i$, $F_i \rightarrow F_{i+1}$
- Suppose 0th domino is knocked over, i.e., $F_0$
- Then all dominoes will fall, i.e., $\forall i. F_i$. 
Dominoes and Mathematical Induction

Domino Principle
- Infinite sequence of dominoes

Induction Principle
- Infinite sequence of statements $S_0, S_1, \ldots$
Dominoes and Mathematical Induction

**Domino Principle**

- Infinite sequence of dominoes
- Knock the 0th domino

**Induction Principle**

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove $S_0$ is correct [Base Case]
Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes
- Knock the 0th domino
- Arrange dominoes such that knocking one will knock the next one

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove $S_0$ is correct [Base Case]
- For an arbitrary $i$, assuming $S_i$ to be correct establishes $S_{i+1}$ to be correct
Dominoes and Mathematical Induction

Domino Principle

- Infinite sequence of dominoes
- Knock the 0th domino
- Arrange dominoes such that knocking one will knock the next one

Induction Principle

- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove $S_0$ is correct [Base Case]
- For an arbitrary $i$, assuming $S_i$ to be correct [Induction Hypothesis] establishes $S_{i+1}$ to be correct [Induction Step]
Dominoes and Mathematical Induction

**Domino Principle**
- Infinite sequence of dominoes
- Knock the 0th domino
- Arrange dominoes such that knocking one will knock the next one
- Conclude all dominoes fall

**Induction Principle**
- Infinite sequence of statements $S_0, S_1, \ldots$
- Prove $S_0$ is correct [**Base Case**]
- For an arbitrary $i$, assuming $S_i$ to be correct [**Induction Hypothesis**] establishes $S_{i+1}$ to be correct [**Induction Step**]
- Conclude $\forall i. S_i$ is true
Proposition

For a DFA \( M = (Q, \Sigma, \delta, q_0, F) \), and any strings \( u, v \in \Sigma^* \) and state \( q \in Q \), \( \hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v) \).
Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

Proof.

We will prove this by induction.
Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

Proof.
We will prove this by induction.

- Let $S_i$ be "$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$"
Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

Proof.

We will prove this by induction.

- Let $S_i$ be “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$”
  - Observe that if $S_i$ is true for all $i$ then $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ for every $u$ and $v$
Example Inductive Proof

Base Case

Proof (contd).

To establish \( S_0 \), i.e., \( \hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v) \) when \( |v| = 0 \)
Example Inductive Proof

Base Case

Proof (contd).
To establish $S_0$, i.e., “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = 0$”

- If $|v| = 0$ then $v = \epsilon$
- Observe $u\epsilon = u$
- Thus, LHS = $\hat{\delta}(q, u\epsilon) = \hat{\delta}(q, u)$
- Observe by definition of $\hat{\delta}(\cdot, \cdot)$, for any $q'$, $\hat{\delta}(q', \epsilon) = q'$
- Thus, RHS = $\hat{\delta}(\hat{\delta}(q, u), \epsilon) = \hat{\delta}(q, u)$
Example Inductive Proof

Induction Step

Proof (contd).

Assume $S_i$, i.e., “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$”. Need to establish $S_{i+1}$.
Example Inductive Proof

Induction Step

Proof (contd).

Assume $S_i$, i.e., “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$”. Need to establish $S_{i+1}$.

- Consider $v$ such that $|v| = i + 1$. 

□
Example Inductive Proof
Induction Step

Proof (contd).
Assume $S_i$, i.e., “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$”. Need to establish $S_{i+1}$.

- Consider $v$ such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$
Example Inductive Proof

Induction Step

Proof (contd).

Assume $S_i$, i.e., “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$”. Need to establish $S_{i+1}$.

- Consider $v$ such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

\[
\hat{\delta}(q, uwa) = \delta(\hat{\delta}(q, uw), a) \quad \text{defn. of } \hat{\delta}
\]
Example Inductive Proof

Induction Step

Proof (contd).
Assume $S_i$, i.e., “$\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$ when $|v| = i$”. Need to establish $S_{i+1}$.

- Consider $v$ such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

\[
\hat{\delta}(q, uwa) = \delta(\hat{\delta}(q, uw), a) \quad \text{defn. of } \hat{\delta}
\]
\[
= \delta(\hat{\delta}(\hat{\delta}(q, u), w), a) \quad \text{ind. hyp.}
\]

□
Example Inductive Proof

Induction Step

Proof (contd).

Assume $S_i$, i.e., \( \hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v) \) when \( |v| = i \). Need to establish $S_{i+1}$.

- Consider $v$ such that $|v| = i + 1$. WLOG, $v = wa$, where $w \in \Sigma^*$ with $|w| = n$ and $a \in \Sigma$

\[
\hat{\delta}(q, uwa) = \delta(\hat{\delta}(q, uw), a) \quad \text{defn. of } \hat{\delta} \\
= \delta(\hat{\delta}(\hat{\delta}(q, u), w), a) \quad \text{ind. hyp.} \\
= \hat{\delta}(\hat{\delta}(q, u), wa) \quad \text{defn. of } \hat{\delta}
\]
Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

Proof.

“We will prove by induction on $|v|$” is a short-hand for
Proposition

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any strings $u, v \in \Sigma^*$ and state $q \in Q$, $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$.

Proof.

“We will prove by induction on $|v|$” is a short-hand for “We will prove the proposition by induction. Take $S_i$ to be statement of the proposition restricted to strings $v$ where $|v| = i$.”
Properties of $\hat{\delta}$

Corollary

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any string $v \in \Sigma^*$, $a \in \Sigma$ and state $q \in Q$, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$. 
Corollary

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any string $v \in \Sigma^*$, $a \in \Sigma$ and state $q \in Q$, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$.

Proof.

From previous proposition we have, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$ (taking $u = a$).
Properties of $\hat{\delta}$

Corollary

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and any string $v \in \Sigma^*$, $a \in \Sigma$ and state $q \in Q$, $\hat{\delta}(q, av) = \hat{\delta}(\delta(q, a), v)$.

Proof.

From previous proposition we have, $\hat{\delta}(q, av) = \hat{\delta}(\hat{\delta}(q, a), v)$ (taking $u = a$). Next,

$$\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a)$$

defn. of $\hat{\delta}$

$$= \delta(q, a)$$

as $\hat{\delta}(q, \epsilon) = q$
Language of $M_{\text{odd}}$

$P(L_{\text{odd}}) = \{w \in \{0, 1\}^* | w \text{ has an odd number of } 0\text{s and an odd number of } 1\text{s}\}$

Transition Diagram of $M_{\text{odd}}$
Language of $M_{\text{odd}}$

**Proposition**

$L(M_{\text{odd}}) = \{ w \in \{0, 1\}^* \mid w \text{ has an odd number of } 0\text{s and an odd number of } 1\text{s}\}$. 

Transition Diagram of $M_{\text{odd}}$
Proof about the language of $M_{\text{odd}}$

Proof.
We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff $w$ has an odd number of 0s and an odd number of 1s.
Proof about the language of $M_{\text{odd}}$

**Proof.**

We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff $w$ has an odd number of 0s and an odd number of 1s.

- **Base Case:** When $w = \epsilon$, $w$ has an even number of 0s and an even number of 1s and $\hat{\delta}(q_0, \epsilon) = q_0$ so the observation holds.
Proof about the language of $M_{odd}$

Proof.
We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff $w$ has an odd number of 0s and an odd number of 1s.

- **Base Case**: When $w = \epsilon$, $w$ has an even number of 0s and an even number of 1s and $\hat{\delta}(q_0, \epsilon) = q_0$ so the observation holds.

- **Induction Step $w = 0u$**: The parity of the number of 1s in $u$ and $w$ is the same, and the parity of the number of 0s is opposite. And $\hat{\delta}(q_0, w) = \hat{\delta}(\delta(q_0, 0), u) = \hat{\delta}(q_3, u)$
Proof about the language of $M_{\text{odd}}$

It fails!

Proof.
We will prove by induction on $|w|$ that $\hat{\delta}(q_0, w) \in F = \{q_2\}$ iff $w$ has an odd number of 0s and an odd number of 1s.

- **Base Case:** When $w = \epsilon$, $w$ has an even number of 0s and an even number of 1s and $\hat{\delta}(q_0, \epsilon) = q_0$ so the observation holds.

- **Induction Step $w = 0u$:** The parity of the number of 1s in $u$ and $w$ is the same, and the parity of the number of 0s is opposite. And $\hat{\delta}(q_0, w) = \hat{\delta}(\delta(q_0, 0), u) = \hat{\delta}(q_3, u)$

- Need to know what strings are accepted from $q_3$! Need to prove a stronger statement.

□
Corrected Proof

Proof.
We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff $w$ has odd number of 0s & odd number of 1s
(b) $\hat{\delta}(q_1, w) \in F$ iff
(c) $\hat{\delta}(q_2, w) \in F$ iff
(d) $\hat{\delta}(q_3, w) \in F$ iff

..→
Corrected Proof

Proof.
We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff $w$ has odd number of 0s & odd number of 1s
(b) $\hat{\delta}(q_1, w) \in F$ iff $w$ has odd number of 0s & even number of 1s
(c) $\hat{\delta}(q_2, w) \in F$ iff $w$ has even number of 0s & even number of 1s
(d) $\hat{\delta}(q_3, w) \in F$ iff $w$ has even number of 0s & odd number of 1s

\[ \rightarrow \]
Proof.
We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff $w$ has odd number of 0s & odd number of 1s
(b) $\hat{\delta}(q_1, w) \in F$ iff $w$ has odd number of 0s & even number of 1s
(c) $\hat{\delta}(q_2, w) \in F$ iff $w$ has even number of 0s & even number of 1s
(d) $\hat{\delta}(q_3, w) \in F$ iff
Proof.
We need to a stronger statement that asserts what strings are accepted from each state of the DFA. We will prove by induction on $|w|$ that

(a) $\hat{\delta}(q_0, w) \in F$ iff $w$ has odd number of 0s & odd number of 1s
(b) $\hat{\delta}(q_1, w) \in F$ iff $w$ has odd number of 0s & even number of 1s
(c) $\hat{\delta}(q_2, w) \in F$ iff $w$ has even number of 0s & even number of 1s
(d) $\hat{\delta}(q_3, w) \in F$ iff $w$ has even number of 0s & odd number of 1s

$\cdots$
Proof (contd).

Consider \( w \) such that \( |w| = 0 \). Then \( w = \epsilon \).
Corrected Proof

Base Case

Proof (contd).
Consider $w$ such that $|w| = 0$. Then $w = \varepsilon$.

- $w$ has even number of 0s and even number of 1s
Proof (contd).

Consider $w$ such that $|w| = 0$. Then $w = \epsilon$.

- $w$ has even number of 0s and even number of 1s
- For any $q \in Q$, $\hat{\delta}(q, w) = q$
Proof (contd).
Consider $w$ such that $|w| = 0$. Then $w = \epsilon$.

- $w$ has even number of 0s and even number of 1s
- For any $q \in Q$, $\hat{\delta}(q, w) = q$
- Thus, $\hat{\delta}(q, w) \in F$ iff $q = q_3$, and statements (a), (b), (c), and (d) hold in the base case.
Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. 

Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$. 

Case $q = q_0, a = 0$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff $u$ has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff $w$ has odd number of 0s and odd number of 1s.

Case $q = q_0, a = 1$: $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_1, u) \in F$ iff $u$ has odd number of 0s and even number of 1s (by ind. hyp. (b)) iff $w$ has odd number of 0s and odd number of 1s.
Corrected Proof
Induction Step: part (a)

Proof (contd).

Suppose (a), (b), (c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.
Corrected Proof
Induction Step: part (a)

Proof (contd).
Suppose (a), (b), (c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- **Case $q = q_0$, $a = 0$:** $\hat{\delta}(q_0, w) \in F$ iff
Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings \( w \) of length \( n \). Consider \( w = au \), where \( a \in \{0, 1\} \) and \( u \in \Sigma^* \) of length \( n \). Recall that \( \hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u) \).

- Case \( q = q_0, a = 0 \): \( \hat{\delta}(q_0, w) \in F \) iff \( \hat{\delta}(q_3, u) \in F \) iff
Corrected Proof

Induction Step: part (a)

Proof (contd).

Suppose (a), (b), (c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- **Case $q = q_0$, $a = 0$:** $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff $u$ has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff
Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\widehat{\delta}(q, au) = \widehat{\delta}(\delta(q, a), u)$.

- **Case $q = q_0$, $a = 0$:** $\delta(q_0, w) \in F$ iff $\delta(q_3, u) \in F$ iff $u$ has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff $w$ has odd number of 0s and odd number of 1s
Corrected Proof

Induction Step: part (a)

Proof (cont’d).

Suppose (a), (b), (c), and (d) hold for strings \( w \) of length \( n \).
Consider \( w = au \), where \( a \in \{0, 1\} \) and \( u \in \Sigma^* \) of length \( n \). Recall that \( \hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u) \).

▶ Case \( q = q_0, a = 0 \): \( \hat{\delta}(q_0, w) \in F \) iff \( \hat{\delta}(q_3, u) \in F \) iff \( u \) has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff \( w \) has odd number of 0s and odd number of 1s

▶ Case \( q = q_0, a = 1 \): \( \hat{\delta}(q_0, w) \in F \) iff
Corrected Proof
Induction Step: part (a)

Proof (contd).
Suppose (a),(b),(c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- **Case $q = q_0, a = 0$:** $\delta(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff $u$ has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff $w$ has odd number of 0s and odd number of 1s

- **Case $q = q_0, a = 1$:** $\delta(q_0, w) \in F$ iff $\hat{\delta}(q_1, u) \in F$ iff
Proof (contd).

Suppose (a),(b),(c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- **Case $q = q_0$, $a = 0$:** $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_3, u) \in F$ iff $u$ has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff $w$ has odd number of 0s and odd number of 1s

- **Case $q = q_0$, $a = 1$:** $\hat{\delta}(q_0, w) \in F$ iff $\hat{\delta}(q_1, u) \in F$ iff $u$ has odd number of 0s and even number of 1s (by ind. hyp. (b)) iff
Proof (contd).

Suppose (a), (b), (c), and (d) hold for strings $w$ of length $n$. Consider $w = au$, where $a \in \{0, 1\}$ and $u \in \Sigma^*$ of length $n$. Recall that $\hat{\delta}(q, au) = \hat{\delta}(\delta(q, a), u)$.

- **Case $q = q_0, a = 0$:** $\delta(q_0, w) \in F$ iff $\delta(q_3, u) \in F$ iff $u$ has even number of 0s and odd number of 1s (by ind. hyp. (d)) iff $w$ has odd number of 0s and odd number of 1s
- **Case $q = q_0, a = 1$:** $\delta(q_0, w) \in F$ iff $\delta(q_1, u) \in F$ iff $u$ has odd number of 0s and even number of 1s (by ind. hyp. (b)) iff $w$ has odd number of 0s and odd number of 1s
Corrected Proof

Proof (contd).

- **Case** $q = q_1$, $a = 0$: $\hat{\delta}(q_1, w) \in F$ iff $\hat{\delta}(q_2, u) \in F$ iff $u$ has even number of 0s and even number of 1s (by ind. hyp. (c)) iff $w$ has odd number of 0s and even number of 1s

And so on for the other cases of $q = q_1$ and $a = 1$, $q = q_2$ and $a = 0$, $q = q_2$ and $a = 1$, and finally $q = q_3$ and $a = 0$, and finally $q = q_3$ and $a = 1$. □
Proof (contd).

- Case $q = q_1$, $a = 0$: $\hat{\delta}(q_1, w) \in F$ iff $\hat{\delta}(q_2, u) \in F$ iff $u$ has even number of 0s and even number of 1s (by ind. hyp. (c)) iff $w$ has odd number of 0s and even number of 1s.
- And so on for the other cases of $q = q_1$ and $a = 1$, $q = q_2$ and $a = 0$, $q = q_2$ and $a = 1$, $q = q_3$ and $a = 0$, and finally $q = q_3$ and $a = 1$. □
Proving Correctness of a DFA

Proof Template
Given a DFA $M$ having $n$ states $\{q_0, q_1, \ldots, q_{n-1}\}$ with initial state $q_0$, and final states $F$, to prove that $L(M) = L$, we do the following.
Proving Correctness of a DFA

Proof Template

Given a DFA $M$ having $n$ states \{q_0, q_1, \ldots q_{n-1}\} with initial state $q_0$, and final states $F$, to prove that $L(M) = L$, we do the following.

1. Come up with languages $L_0, L_1, \ldots L_{n-1}$ such that $L_0 = L$
Proving Correctness of a DFA

Proof Template

Given a DFA $M$ having $n$ states \{q_0, q_1, \ldots q_{n-1}\} with initial state $q_0$, and final states $F$, to prove that $L(M) = L$, we do the following.

1. Come up with languages $L_0, L_1, \ldots L_{n-1}$ such that $L_0 = L$
2. Prove by induction on $|w|$, $\hat{\delta}(q_i, w) \in F$ if and only if $w \in L_i$
Problem
Given a language $L$, design a DFA $M$ that accepts $L$, i.e., $L(M) = L$. 
Typical Problem

Problem
Given a language $L$, design a DFA $M$ that accepts $L$, i.e., $L(M) = L$.
How does one go about it?
Methodology

- Imagine yourself in the place of the machine, reading symbols of the input, and trying to determine if it should be accepted.
- Remember at any point you have only seen a part of the input, and you don’t know when it ends.
Methodology

- Imagine yourself in the place of the machine, reading symbols of the input, and trying to determine if it should be accepted.
- Remember at any point you have only seen a part of the input, and you don’t know when it ends.
- Figure out what to keep in memory. It cannot be all the symbols seen so far: it must fit into a finite number of bits.
Strings containing 0

**Problem**
Design an automaton that accepts all strings over \( \{0, 1\} \) that contain at least one 0.

**Solution**
What do you need to remember?
Problem
Design an automaton that accepts all strings over \{0, 1\} that contain at least one 0.

Solution
What do you need to remember? Whether you have seen a 0 so far or not!

Automaton accepting strings with at least one 0.
Even length strings

**Problem**
Design an automaton that accepts all strings over \{0, 1\} that have an even length.

**Solution**
What do you need to remember?
Even length strings

Problem
Design an automaton that accepts all strings over \{0, 1\} that have an even length.

Solution
What do you need to remember? Whether you have seen an odd or an even number of symbols.

Automaton accepting strings of even length.
Pattern Recognition

Problem
Design an automaton that accepts all strings over \( \{0, 1\} \) that have 001 as a substring, where \( u \) is a substring of \( w \) if there are \( w_1 \) and \( w_2 \) such that \( w = w_1uw_2 \).

Solution
What do you need to remember?
Pattern Recognition

Problem
Design an automaton that accepts all strings over \{0, 1\} that have 001 as a substring, where \(u\) is a substring of \(w\) if there are \(w_1\) and \(w_2\) such that \(w = w_1uw_2\).

Solution
What do you need to remember? Whether you
- haven’t seen any symbols of the pattern
- have just seen 0
- have just seen 00
- have seen the entire pattern 001
Automaton accepting strings having 001 as substring.
grep Problem

Problem
Given text $T$ and string $s$, does $s$ appear in $T$?
grep Problem

Problem
Given text $T$ and string $s$, does $s$ appear in $T$?

Solution

$$\begin{align*}
&= s? \\
&\quad = s? \\
&\quad \quad = s? \\
&\quad \quad \quad = s? \\
&\quad \quad \quad \quad = s? \\
&\quad \quad \quad \quad \quad = s? \\
&\quad \quad \quad \quad \quad \quad = s? \\
&\quad \quad \quad \quad \quad \quad \quad = s? \\
&T_1 \quad T_2 \quad T_3 \quad \ldots \quad T_n \quad T_{n+1} \quad \ldots \quad T_t
\end{align*}$$

$$\text{Running time } = O(n \cdot t), \text{ where } |T| = t \text{ and } |s| = n.$$
grep Problem

Problem
Given text $T$ and string $s$, does $s$ appear in $T$?

Naïve Solution

\[
\begin{align*}
&=s? \\
&=s? \\
&=s? \\
&=s? \\
&=s? \\
&T_1 \quad T_2 \quad T_3 \quad \ldots \quad T_n \quad T_{n+1} \quad \ldots \quad T_t
\end{align*}
\]

Running time $= O(nt)$, where $|T| = t$ and $|s| = n$. 
Solution

- Build DFA $M$ for $L = \{w \mid$ there are $u, v$ s.t. $w = usv\}$
- Run $M$ on text $T$
grep Problem
Smarter Solution

Solution

- Build DFA $M$ for $L = \{ w \mid \text{there are } u, v \text{ s.t. } w = usv \}$
- Run $M$ on text $T$

Time = time to build $M + O(t)$!
 grep Problem
Smarter Solution

Solution

- Build DFA $M$ for $L = \{w \mid$ there are $u, v$ s.t. $w = usv\}$
- Run $M$ on text $T$

Time = time to build $M + O(t)!$

Questions

- Is $L$ regular no matter what $s$ is?
- If yes, can $M$ be built “efficiently”?
grep Problem
Smarter Solution

Solution
- Build DFA $M$ for $L = \{w \mid$ there are $u, v$ s.t. $w = usv\}$
- Run $M$ on text $T$

Time = time to build $M + O(t)$!

Questions
- Is $L$ regular no matter what $s$ is?
- If yes, can $M$ be built “efficiently”?

Knuth-Morris-Pratt (1977): Yes to both the above questions.
Knuth-Morris-Pratt (1977)

(a) $T$ = bacbababababaacbaba

(b) $T$ = bacbababababaacbaba

(c) $P_q = ababa$

$P_k = aba$

$s = ababac$  $q = 2$

$s' = s + 2$  $k = 3$
Problem
Design an automaton that accepts all strings \( w \) over \( \{0, 1\} \) such that \( w \) is the binary representation of a number that is a multiple of 5.

Solution
What must be remembered?

▶ If \( w \) is the number \( n \) then \( w_0 \) is \( 2^n \) and \( w_1 \) is \( 2^n + 1 \).

▶ \((a \cdot b + c) \mod 5 = (a \cdot (b \mod 5) + c) \mod 5\)

▶ e.g.
\(1011 = 11 \text{ (decimal)} \equiv 1 \mod 5\)
\(10110 = 22 \text{ (decimal)} \equiv 2 \mod 5\)
\(10111 = 23 \text{ (decimal)} \equiv 3 \mod 5\)
Multiples

Problem
Design an automaton that accepts all strings $w$ over $\{0, 1\}$ such that $w$ is the binary representation of a number that is a multiple of 5.

Solution
What must be remembered? The remainder when divided by 5.
Multiples

Problem
Design an automaton that accepts all strings $w$ over $\{0, 1\}$ such that $w$ is the binary representation of a number that is a multiple of 5.

Solution
What must be remembered? The remainder when divided by 5.
How do you compute remainders?

▶ If $w$ is the number $n$ then
$w_0$ is $2^n$ and $w_1$ is $2^n + 1$.

▶ $(a \cdot b + c) \mod 5 = (a \cdot (b \mod 5) + c) \mod 5$

▶ e.g.
$1011 = 11$ (decimal) $\equiv 1 \mod 5$
$10110 = 22$ (decimal) $\equiv 2 \mod 5$
$10111 = 23$ (decimal) $\equiv 3 \mod 5$
Multiples

Problem
Design an automaton that accepts all strings $w$ over $\{0, 1\}$ such that $w$ is the binary representation of a number that is a multiple of 5.

Solution
What must be remembered? The remainder when divided by 5. How do you compute remainders?

- If $w$ is the number $n$ then $w_0$ is $2n$ and $w_1$ is $2n + 1$.
- $(a \cdot b + c) \mod 5 = (a \cdot (b \mod 5) + c) \mod 5$
- e.g. $1011 = 11$ (decimal) $\equiv 1 \mod 5$
  $10110 = 22$ (decimal) $\equiv 2 \mod 5$
  $10111 = 23$ (decimal) $\equiv 3 \mod 5$
Automaton recognizing binary numbers that are multiples of 5.
Problem
Design an automaton for the language $L_k = \{w \mid k\text{th character from end of } w \text{ is } 1\}$

Solution
What do you need to remember?
Problem
Design an automaton for the language $L_k = \{ w | k\text{th character from end of } w \text{ is } 1 \}$

Solution
What do you need to remember? The last $k$ characters seen so far!
Formally, $M_k = (Q, \{0, 1\}, \delta, q_0, F)$

- **States** = $Q = \{ \langle w \rangle | w \in \{0, 1\}^* \text{ and } |w| \leq k \}$
- $\delta(\langle w \rangle, b) = \begin{cases} \langle wb \rangle & \text{if } |w| < k \\ \langle w_2w_3\ldots w_kb \rangle & \text{if } w = w_1w_2\ldots w_k \end{cases}$
- $q_0 = \langle \epsilon \rangle$
- $F = \{ \langle 1w_2w_3\ldots w_k \rangle | w_i \in \{0, 1\} \}$
Proposition

Any DFA recognizing $L_k$ has at least $2^k$ states.
Lower Bound on DFA size

Proposition

Any DFA recognizing $L_k$ has at least $2^k$ states.

Proof.
Let $M$, with initial state $q_0$, recognize $L_k$ and assume (for contradiction) that $M$ has $< 2^k$ states.
Lower Bound on DFA size

Proposition

Any DFA recognizing \( L_k \) has at least \( 2^k \) states.

Proof.

Let \( M \), with initial state \( q_0 \), recognize \( L_k \) and assume (for contradiction) that \( M \) has \( < 2^k \) states.

- Number of strings of length \( k = 2^k \)
- There must be two distinct string \( w_0 \) and \( w_1 \) of length \( k \) such that \( \hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1) \).
Proof (contd).

Let $i$ be the first position where $w_0$ and $w_1$ differ. Without loss of generality assume that $w_0$ has 0 in the $ith$ position and $w_1$ has 1.

$$w_0 = \ldots 0 \ldots$$
$$w_1 = \underbrace{\ldots 1 \ldots}_{i-1 \quad k-i}$$
Proof (contd).

Let \( i \) be the first position where \( w_0 \) and \( w_1 \) differ. Without loss of generality assume that \( w_0 \) has 0 in the \( ith \) position and \( w_1 \) has 1.

\[
\begin{align*}
  w_0 0^{i-1} &= \ldots 0 \ldots 0^{i-1} \\
  w_1 0^{i-1} &= \ldots 1 \ldots 0^{i-1}
\end{align*}
\]

Thus, \( M \) cannot accept both \( w_0 0^{i-1} \) and \( w_1 0^{i-1} \).
Proof (contd).

Let $i$ be the first position where $w_0$ and $w_1$ differ. Without loss of generality assume that $w_0$ has 0 in the $ith$ position and $w_1$ has 1.

\[
\begin{align*}
    w_00^{i-1} &= \ldots 0 \ldots 0^{i-1} \\
    w_10^{i-1} &= \underbrace{\ldots 1}_{i-1} \underbrace{\ldots 0^{i-1}}_{k-i}
\end{align*}
\]

$w_00^{i-1} \notin L_k$ and $w_10^{i-1} \in L_k$. Thus, $M$ cannot accept both $w_00^{i-1}$ and $w_10^{i-1}$. \[\rightarrow\]
Proof (contd).

So far, $w_0 0^{i-1} \notin L_k$, $w_1 0^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.
Proof (contd).

So far, $w_00^{i-1} \notin L_k$, $w_10^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

Thus, $M$ accepts or rejects both $w_00^{i-1}$ and $w_10^{i-1}$.

Contradiction! □
Proof (contd)

... Almost there

Proof (contd).

So far, $w_00^{i-1} \notin L_k$, $w_10^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

$$\hat{\delta}(q_0, w_00^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1})$$

by Proposition proved

\[\square\]
Proof (contd).

So far, \( w_00^{i-1} \not\in L_k \), \( w_10^{i-1} \in L_k \), and \( \hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1) \).

\[
\hat{\delta}(q_0, w_00^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) \quad \text{by Proposition proved}
\]

\[
= \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) \quad \text{by assump. on } w_0 \text{ and } w_1
\]

\[\square\]
Proof (contd).

So far, \( w_00^{i-1} \notin L_k \), \( w_10^{i-1} \in L_k \), and \( \hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1) \).

\[
\begin{align*}
\hat{\delta}(q_0, w_00^{i-1}) &= \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) \quad \text{by Proposition proved} \\
&= \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) \quad \text{by assump. on } w_0 \text{ and } w_1 \\
&= \hat{\delta}(q_0, w_10^{i-1}) \quad \text{by Proposition proved}
\end{align*}
\]

Thus, \( M \) accepts or rejects both \( w_00^{i-1} \) and \( w_10^{i-1} \).

\[ \square \]
Proof (contd).
So far, $w_00^{i-1} \not\in L_k$, $w_10^{i-1} \in L_k$, and $\hat{\delta}(q_0, w_0) = \hat{\delta}(q_0, w_1)$.

\[
\begin{align*}
\hat{\delta}(q_0, w_00^{i-1}) &= \hat{\delta}(\hat{\delta}(q_0, w_0), 0^{i-1}) & \text{by Proposition proved} \\
&= \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) & \text{by assump. on } w_0 \text{ and } w_1 \\
&= \hat{\delta}(q_0, w_10^{i-1}) & \text{by Proposition proved}
\end{align*}
\]

Thus, $M$ accepts or rejects both $w_00^{i-1}$ and $w_10^{i-1}$.
Contradiction! \qed