Turing Machines

Arjun Chandrasekhar

History of Computer Science

History of Computer Science

- Euclidian algorithm for finding the gcd of two integers; first known 'algorithm' (300 BC).

History of Computer Science

- Euclidian algorithm for finding the gcd of two integers; first known 'algorithm' (300 BC).
- Hilbert's 10th problem: Given a Diophantine equation with any number of variables and integer coefficients: devise an algorithm to determine (in finite time) whether the equation has *integer* solutions (1900).

History of Computer Science

- Euclidian algorithm for finding the gcd of two integers; first known ‘algorithm’ (300 BC).
- Hilbert’s 10th problem: Given a Diophantine equation with any number of variables and integer coefficients: devise an algorithm to determine (in finite time) whether the equation has *integer* solutions (1900).
- **Alan Turing (Turing Machines) (1936)**

History of Computer Science

- Euclidian algorithm for finding the gcd of two integers; first known ‘algorithm’ (300 BC).
- Hilbert’s 10th problem: Given a Diophantine equation with any number of variables and integer coefficients: devise an algorithm to determine (in finite time) whether the equation has *integer* solutions (1900).
- **Alan Turing (Turing Machines) (1936)**
- Alonzo Church (Lambda Calculus) (1936)

History of Computer Science

- Euclidian algorithm for finding the gcd of two integers; first known 'algorithm' (300 BC).
- Hilbert's 10th problem: Given a Diophantine equation with any number of variables and integer coefficients: devise an algorithm to determine (in finite time) whether the equation has *integer* solutions (1900).
- **Alan Turing (Turing Machines) (1936)**
- Alonzo Church (Lambda Calculus) (1936)
- Church-Turing Thesis

Turing Machine

Turing Machine

A new model of computation

Turing Machine

A new model of computation

- Tape consisting of infinitely many squares

Turing Machine

A new model of computation

- Tape consisting of infinitely many squares
- Tape head that can move around the tape one square at a time

Turing Machine

A new model of computation

- Tape consisting of infinitely many squares
- Tape head that can move around the tape one square at a time
- Tape head can read from and write to the tape

Turing Machine

A new model of computation

- Tape consisting of infinitely many squares
- Tape head that can move around the tape one square at a time
- Tape head can read from and write to the tape
- The tape head a state that it can change based on what it reads

Turing Machine

A new model of computation

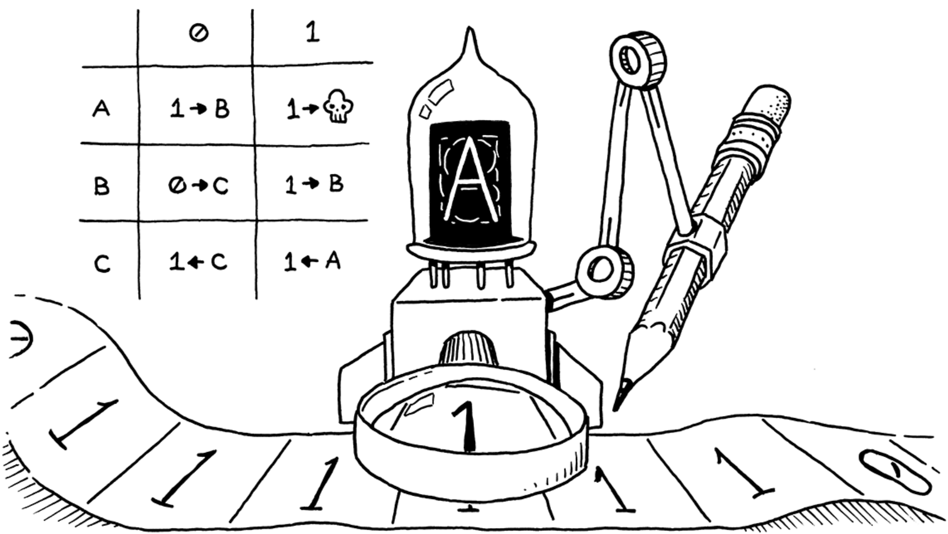
- Tape consisting of infinitely many squares
- Tape head that can move around the tape one square at a time
- Tape head can read from and write to the tape
- The tape head a state that it can change based on what it reads
- The machine accepts if the tape head enters the 'accept state'

Turing Machine

A new model of computation

- Tape consisting of infinitely many squares
- Tape head that can move around the tape one square at a time
- Tape head can read from and write to the tape
- The tape head a state that it can change based on what it reads
- The machine accepts if the tape head enters the ‘accept state’
- The machine rejects if the tape head enters the ‘reject state’

Turing Machine



Turing Machine Formal Description

Turing Machine Formal Description

Before you get stressed out by the next slide...

Turing Machine Formal Description

Before you get stressed out by the next slide...

- You do NOT need to memorize the formal definition of a TM

Turing Machine Formal Description

Before you get stressed out by the next slide...

- You do NOT need to memorize the formal definition of a TM
- I will NEVER ask you to give a formal description - informal descriptions will suffice on all assignments and exams

Turing Machine Formal Description

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple
$(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple
$(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$
3. Γ is the **tape alphabet**. We require that $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$
3. Γ is the **tape alphabet**. We require that $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the **transition function**

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$
3. Γ is the **tape alphabet**. We require that $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the **transition function**
5. $q_s \in Q$ is the **start state**

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$
3. Γ is the **tape alphabet**. We require that $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the **transition function**
5. $q_s \in Q$ is the **start state**
6. $q_A \in Q$ is the **accept state**

Turing Machine Formal Description

A **Turing Machine (TM)** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_s, q_A, q_R)$

1. Q is the set of **states**
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$
3. Γ is the **tape alphabet**. We require that $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the **transition function**
5. $q_s \in Q$ is the **start state**
6. $q_A \in Q$ is the **accept state**
7. $q_R \in Q$ is the **reject state**

Turing Machine Computation

Turing Machine Computation

A TM computes as follows:

Turing Machine Computation

A TM computes as follows:

1. Input w is placed on the leftmost squares on the tape (everything else is blank symbols $\sqcup$)

Turing Machine Computation

A TM computes as follows:

1. Input w is placed on the leftmost squares on the tape (everything else is blank symbols $\sqcup$)
2. At each step, the tape head reads the character on the current square

Turing Machine Computation

A TM computes as follows:

1. Input w is placed on the leftmost squares on the tape (everything else is blank symbols $\sqcup$)
2. At each step, the tape head reads the character on the current square
3. Based on the transition function, it changes to a new state, writes a new character, and moves left or right

Turing Machine Computation

A TM computes as follows:

1. Input w is placed on the leftmost squares on the tape (everything else is blank symbols $\sqcup$)
2. At each step, the tape head reads the character on the current square
3. Based on the transition function, it changes to a new state, writes a new character, and moves left or right
4. This continues until the machine enters the accept or reject state

Turing Machine Example

A machine to recognize $\Sigma^* 1 \Sigma^*$

Turing Machine Example

A machine to recognize $\Sigma^* 1 \Sigma^*$

1. Scan left to right

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$

1. Scan left to right
2. If we encounter a 1, immediately accept

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$

1. Scan left to right
2. If we encounter a 1, immediately accept
3. If we encounter a blank space (i.e. end of input), reject

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$ (formal definition)

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$ (formal definition)

1. $Q = \{q_0, q_A, q_R\}$

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$ (formal definition)

1. $Q = \{q_0, q_A, q_R\}$
2. $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$ (formal definition)

1. $Q = \{q_0, q_A, q_R\}$
2. $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$
3. $\delta(q_0, 0) = (q_0, 0, R)$
   $\delta(q_0, 1) = (q_A, 1, R)$
   $\delta(q_0, \sqcup) = (q_R, \sqcup, L)$

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$ (formal definition)

1. $Q = \{q_0, q_A, q_R\}$
2. $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$
3. $\delta(q_0, 0) = (q_0, 0, R)$
   $\delta(q_0, 1) = (q_A, 1, R)$
   $\delta(q_0, \sqcup) = (q_R, \sqcup, L)$
4. $q_s = q_0$, $q_A = q_A$, $q_R = q_R$

Turing Machine Example

A machine to recognize $\Sigma^*1\Sigma^*$ (formal definition)

1. $Q = \{q_0, q_A, q_R\}$
2. $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$
3. $\delta(q_0, 0) = (q_0, 0, R)$
   $\delta(q_0, 1) = (q_A, 1, R)$
   $\delta(q_0, \sqcup) = (q_R, \sqcup, L)$
4. $q_s = q_0$, $q_A = q_A$, $q_R = q_R$

What does this machine do on 010? 00?

Turing Machine Configuration

Turing Machine Configuration

- Over time, the TM changes three things: the tape head state, the tape contents, and the tape head location

Turing Machine Configuration

- Over time, the TM changes three things: the tape head state, the tape contents, and the tape head location
- A TM **configuration** is a formal way of describing the overall state of the machine

Turing Machine Configuration

Turing Machine Configuration

- Let $q \in Q$ be a state

Turing Machine Configuration

- Let $q \in Q$ be a state
- Let $u, v \in \Gamma^*$ be strings from the tape alphabet

Turing Machine Configuration

- Let $q \in Q$ be a state
- Let $u, v \in \Gamma^*$ be strings from the tape alphabet
- We write the configuration $C = uqv$ to denote:

Turing Machine Configuration

- Let $q \in Q$ be a state
- Let $u, v \in \Gamma^*$ be strings from the tape alphabet
- We write the configuration $C = uqv$ to denote:

  1. The current tape head state is q

Turing Machine Configuration

- Let $q \in Q$ be a state
- Let $u, v \in \Gamma^*$ be strings from the tape alphabet
- We write the configuration $C = uqv$ to denote:

    1. The current tape head state is q
    2. The tape contains uv

Turing Machine Configuration

- Let $q \in Q$ be a state
- Let $u, v \in \Gamma^*$ be strings from the tape alphabet
- We write the configuration $C = uqv$ to denote:

  1. The current tape head state is q
  2. The tape contains uv
  3. The tape head is on the first symbol of v

Turing Machine Configuration

What is the TM state in this configuration?

$$1011q_7 01111$$

Turing Machine Configuration

What is the TM state in this configuration?

$$1011q_7 01111$$

The TM is in state q_7

Turing Machine Configuration

What are the tape contents in this configuration?

$$1011q_701111$$

Turing Machine Configuration

What are the tape contents in this configuration?

$$1011q_7 01111$$

The tape contents are 101101111

Turing Machine Configuration

Where is the TM head in this configuration?

$$1011q_701111$$

Turing Machine Configuration

Where is the TM head in this configuration?

$$1011q_7 0 1111$$

The TM is on top of the second 0

TM Computation (Formal Definition)

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$
- Let $u, v \in \Gamma^*$

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$
- Let $u, v \in \Gamma^*$
- Let $q_i, q_j \in Q$

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$
- Let $u, v \in \Gamma^*$
- Let $q_i, q_j \in Q$
- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$
- Let $u, v \in \Gamma^*$
- Let $q_i, q_j \in Q$
- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$
  - "If the machine is reads b from state q_i, write c, transition to state q_j and move left"

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$
- Let $u, v \in \Gamma^*$
- Let $q_i, q_j \in Q$
- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$
  - "If the machine is reads b from state q_i, write c, transition to state q_j and move left"
- uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$

TM Computation (Formal Definition)

We say configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Let $a, b, c \in \Gamma$
- Let $u, v \in \Gamma^*$
- Let $q_i, q_j \in Q$
- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$
  - "If the machine is reads b from state q_i, write c, transition to state q_j and move left"
- uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$
  - "If the machine is reads b from state q_i, write c, transition to state q_j and move right"

Turing Machine Acceptance

Turing Machine Acceptance

- The **start configuration** of M on input w is the configuration $q_0 w$

Turing Machine Acceptance

- The **start configuration** of M on input w is the configuration $q_0 w$
- Any configuration that includes q_A is an **accepting configuration**

Turing Machine Acceptance

- The **start configuration** of M on input w is the configuration q_0w
- Any configuration that includes q_A is an **accepting configuration**
- A Turing Machine M accepts input w if there are a sequence of configurations $C_1, C_2, \ldots, C_n$ where

Turing Machine Acceptance

- The **start configuration** of M on input w is the configuration $q_0 w$
- Any configuration that includes q_A is an **accepting configuration**
- A Turing Machine M accepts input w if there are a sequence of configurations $C_1, C_2, \ldots, C_n$ where
  1. C_i is the start configuration

Turing Machine Acceptance

- The **start configuration** of M on input w is the configuration $q_0 w$
- Any configuration that includes q_A is an **accepting configuration**
- A Turing Machine M accepts input w if there are a sequence of configurations $C_1, C_2, \ldots, C_n$ where
    1. C_i is the start configuration
    2. Each C_i yields C_{i+1}

Turing Machine Acceptance

- The **start configuration** of M on input w is the configuration $q_0 w$
- Any configuration that includes q_A is an **accepting configuration**
- A Turing Machine M accepts input w if there are a sequence of configurations $C_1, C_2, \ldots, C_n$ where
  1. C_i is the start configuration
  2. Each C_i yields C_{i+1}
  3. C_n is an accepting configuration

Turing Machine Rejection

- The **start configuration** of M on input w is the configuration $q_0 w$
- Any configuration that includes q_R is a **rejecting configuration**
- A Turing Machine M rejects input w if there are a sequence of configurations $C_1, C_2, \ldots, C_n$ where
  1. C_i is the start configuration
  2. Each C_i yields C_{i+1}
  3. C_n is a rejecting configuration

Turing Machine Computation

Some notes

Turing Machine Computation

Some notes

- Unlike automata, a TM does *not* have to read characters one by one; it starts with the entire input on the tape, and it may move around freely

Turing Machine Computation

Some notes

- Unlike automata, a TM does *not* have to read characters one by one; it starts with the entire input on the tape, and it may move around freely
- If the machine is at the left end of the tape and it tries to move left, it stays in place

Turing Machine Example

What does the following TM do on the input ϵ?

State	0	1	$\sqcup$
q_0 (start)	$0 \rightarrow q_1$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{reject}}$
q_1	$0 \rightarrow q_1$	$1 \rightarrow q_1$	$\sqcup \rightarrow q_1$
q_2	$0 \rightarrow q_2$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{accept}}$

A. Accept

B. Reject

C. Loop

Turing Machine Example

What does the following TM do on the input ϵ?

State	0	1	$\sqcup$
q_0 (start)	$0 \rightarrow q_1$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{reject}}$
q_1	$0 \rightarrow q_1$	$1 \rightarrow q_1$	$\sqcup \rightarrow q_1$
q_2	$0 \rightarrow q_2$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{accept}}$

A. Accept

B. Reject ✓

C. Loop

Turing Machine Example

What does the following TM do on the input 000?

State	0	1	$\sqcup$
q_0 (start)	$0 \rightarrow q_1$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{reject}}$
q_1	$0 \rightarrow q_1$	$1 \rightarrow q_1$	$\sqcup \rightarrow q_1$
q_2	$0 \rightarrow q_2$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{accept}}$

A. Accept

B. Reject

C. Loop

Turing Machine Example

What does the following TM do on the input 000?

State	0	1	$\sqcup$
q_0 (start)	$0 \rightarrow q_1$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathsf{q}_{\mathsf{reject}}$
q_1	$0 \rightarrow q_1$	$1 \rightarrow q_1$	$\sqcup \rightarrow q_1$
q_2	$0 \rightarrow q_2$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathsf{q}_{\mathsf{accept}}$

A. Accept

B. Reject

C. Loop ✓

Turing Machine Example

What does the following TM do on the input 111?

State	0	1	$\sqcup$
q_0 (start)	$0 \to q_1$	$1 \to q_2$	$\sqcup \to \text{q}_{\text{reject}}$
q_1	$0 \to q_1$	$1 \to q_1$	$\sqcup \to q_1$
q_2	$0 \to q_2$	$1 \to q_2$	$\sqcup \to \text{q}_{\text{accept}}$

A. Accept

B. Reject

C. Loop

Turing Machine Example

What does the following TM do on the input 111?

State	0	1	$\sqcup$
q_0 (start)	$0 \rightarrow q_1$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q_{reject}}$
q_1	$0 \rightarrow q_1$	$1 \rightarrow q_1$	$\sqcup \rightarrow q_1$
q_2	$0 \rightarrow q_2$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q_{accept}}$

A. Accept ✓

B. Reject

C. Loop

Turing Machine Example

What does the following TM do on the input 101?

State	0	1	$\sqcup$
q_0 (start)	$0 \to q_1$	$1 \to q_2$	$\sqcup \to \mathrm{q_{reject}}$
q_1	$0 \to q_1$	$1 \to q_1$	$\sqcup \to q_1$
q_2	$0 \to q_2$	$1 \to q_2$	$\sqcup \to \mathrm{q_{accept}}$

A. Accept

B. Reject

C. Loop

Turing Machine Example

What does the following TM do on the input 101?

State	0	1	$\sqcup$
q_0 (start)	$0 \rightarrow q_1$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{reject}}$
q_1	$0 \rightarrow q_1$	$1 \rightarrow q_1$	$\sqcup \rightarrow q_1$
q_2	$0 \rightarrow q_2$	$1 \rightarrow q_2$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{accept}}$

A. Accept ✓

B. Reject

C. Loop

Turing Machine Halting

Turing Machine Halting

A Turing machine **halts** on input w if it accepts or rejects w

Turing Machine Halting

A Turing machine **halts** on input w if it accepts or rejects w

- The machine doesn't simply read each character once - it can go back and forth

Turing Machine Halting

A Turing machine **halts** on input *w* if it accepts or rejects *w*

- The machine doesn't simply read each character once - it can go back and forth
- THERE IS NO INHERENT GUARANTEE THAT A TURING MACHINE WILL HALT

Turing Machine Descriptions

Turing Machine Descriptions

There are three different ways we can describe a Turing machine

Turing Machine Descriptions

There are three different ways we can describe a Turing machine

1. **High level description**

Turing Machine Descriptions

There are three different ways we can describe a Turing machine

1. **High level description**
2. **Tape/implementation level description**

Turing Machine Descriptions

There are three different ways we can describe a Turing machine

1. **High level description**
2. **Tape/implementation level description**
3. **Formal description**

High Level Description

Clear, English description of the algorithm to solve the problem, but does not describe how to specifically implement it on a Turing machine

Tape Level Description

Clear description of how the Turing machine moves around and manipulates the tape, and when it accepts/rejects, but doesn't describe *every* individual state and transition

Formal Description

Describes every single state and every single transition

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

High level description:

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

High level description:

1. **Base Case:** if $w = \epsilon$ then accept

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

High level description:

1. **Base Case:** if $w = \epsilon$ then accept
2. **Recursive Case:** if $|w| = n \geq 1$, check if the first and last character match. If they do, erase them, and recurse on all the middle characters characters.

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Tape level description:

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Tape level description:

1. Read the left-most character, remember it (through the state), and erase it

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Tape level description:

1. Read the left-most character, remember it (through the state), and erase it
2. Scan until the end of the input (i.e. when you reach a blank) and check if the character at the end matches the character you read at the start

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Tape level description:

1. Read the left-most character, remember it (through the state), and erase it
2. Scan until the end of the input (i.e. when you reach a blank) and check if the character at the end matches the character you read at the start
   2.1 If so, erase it, go back to the start and repeat

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Tape level description:

1. Read the left-most character, remember it (through the state), and erase it
2. Scan until the end of the input (i.e. when you reach a blank) and check if the character at the end matches the character you read at the start
   2.1 If so, erase it, go back to the start and repeat
   2.2 If not, reject immediately

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Tape level description:

1. Read the left-most character, remember it (through the state), and erase it
2. Scan until the end of the input (i.e. when you reach a blank) and check if the character at the end matches the character you read at the start
   2.1 If so, erase it, go back to the start and repeat
   2.2 If not, reject immediately
3. Once the tape is blank, accept

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Formal description:

Turing Machine Example

Let's design a Turing machine to recognize the language of palindromes

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

Formal description:

	a	b	$\sqcup$
q_s (start)	$\sqcup \rightarrow q_a$	$\sqcup \rightarrow q_b$	$\sqcup \rightarrow \mathrm{q}_{\mathrm{accept}}$
q_a	$a \rightarrow q_a$	$b \rightarrow q_a$	$\sqcup \leftarrow q_a*$
q_b	$a \rightarrow q_b$	$b \rightarrow q_b$	$\sqcup \leftarrow q_b*$
q_a*	$\sqcup \leftarrow q_s*$	$\sqcup \leftarrow \mathrm{q}_{\mathrm{reject}}$	$\sqcup \leftarrow q_s*$
q_b*	$\sqcup \leftarrow \mathrm{q}_{\mathrm{reject}}$	$\sqcup \leftarrow q_s*$	$\sqcup \leftarrow q_s*$
q^s*	$a \leftarrow q_s*$	$b \leftarrow q_s*$	$\sqcup \rightarrow q_s$

The language of a TM

The language of a TM

Let M be a TM. We say the **language of** M, denoted $L(M)$, is the set of strings that are accepted by M

Turing-Decidable Languages

Turing-Decidable Languages

- We say M **decides** L if

Turing-Decidable Languages

- We say M **decides** L if
  1. If $w \in L$, M halts and accepts w

Turing-Decidable Languages

- We say M **decides** L if
  1. If $w \in L$, M halts and accepts w
  2. If $w \notin L$, M halts and rejects w

Turing-Decidable Languages

- We say M **decides** L if
  1. If $w \in L$, M halts and accepts w
  2. If $w \notin L$, M halts and rejects w
- Note that M should halt on **all** inputs

Turing-Decidable Languages

- We say M **decides** L if
  1. If $w \in L$, M halts and accepts w
  2. If $w \notin L$, M halts and rejects w
- Note that M should halt on **all** inputs
- We say L is **Turing-Decidable**, or simply **Decidable**

Turing-Recognizable Languages

Turing-Recognizable Languages

- We say M **recognizes** L if $L(M) = L$. That is,

Turing-Recognizable Languages

- We say M **recognizes** L if $L(M) = L$. That is,
  1. If $w \in L$, M halts and accepts w

Turing-Recognizable Languages

- We say M **recognizes** L if $L(M) = L$. That is,
  1. If $w \in L$, M halts and accepts w
  2. If $w \notin L$, M does not accept w. This could mean M halts and rejects, or M loops forever.

Turing-Recognizable Languages

- We say M **recognizes** L if $L(M) = L$. That is,
  1. If $w \in L$, M halts and accepts w
  2. If $w \notin L$, M does not accept w. This could mean M halts and rejects, or M loops forever.
- Note that M is only guaranteed to halt if the input is in the language.

Turing-Recognizable Languages

- We say M **recognizes** L if $L(M) = L$. That is,
  1. If $w \in L$, M halts and accepts w
  2. If $w \notin L$, M does not accept w. This could mean M halts and rejects, or M loops forever.
- Note that M is only guaranteed to halt if the input is in the language.
- We say L is **Turing-recognizable**, or simply **Recognizable**.

Turing-Recognizable Languages

Turing-Recognizable Languages

- For DFAs, NFAs, PDAs, etc. we have used the terms "decide" and "recognize" interchangeably

Turing-Recognizable Languages

- For DFAs, NFAs, PDAs, etc. we have used the terms “decide” and “recognize” interchangeably
  - This is because there is no risk of these machines looping forever

Turing-Recognizable Languages

- For DFAs, NFAs, PDAs, etc. we have used the terms “decide” and “recognize” interchangeably
  - This is because there is no risk of these machines looping forever
  - In my defense, I did not make create this convention, I just follow it

Turing-Recognizable Languages

- For DFAs, NFAs, PDAs, etc. we have used the terms "decide" and "recognize" interchangeably
  - This is because there is no risk of these machines looping forever
  - In my defense, I did not make create this convention, I just follow it
- With Turing-machines, we have to explicitly distinguish between deciding and recognizing a language.

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

- The tape level description could could take awhile to write out...

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

- The tape level description could could take awhile to write out...
- ...to say nothing of the formal description!

Turing

Let's
Turin

- e
  a
- .

Church-Turing Thesis

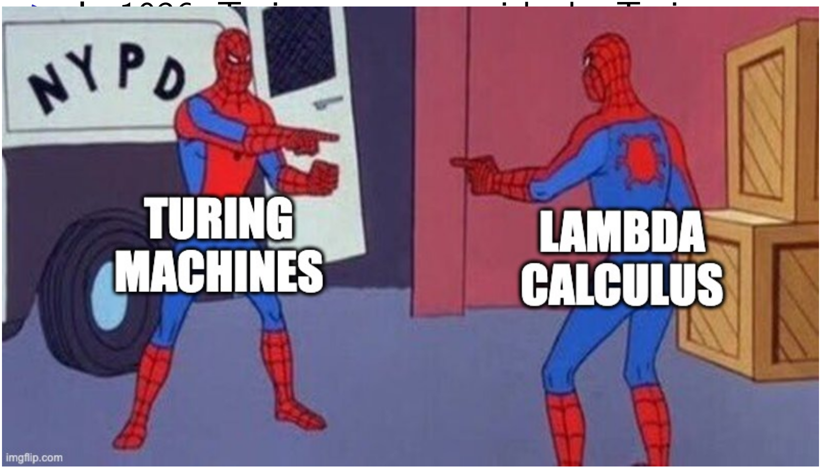
Church-Turing Thesis

- In 1936, Turing came up with the Turing machine while Alonzo Church simultaneously invented lambda calculus

Church-Turing Thesis

- In 1936, Turing came up with the Turing machine while Alonzo Church simultaneously invented lambda calculus
- Turing showed that these two formulations described the *same* class of functions.

Church-Turing Thesis



Church-Turing Thesis

- In 1936, Turing came up with the Turing machine while Alonzo Church simultaneously invented lambda calculus
- Turing showed that these two formulations described the *same* class of functions.
- Since then, several other models of computation have been proposed, and they all turned out to be equivalent to Turing machines.

Chu

ll

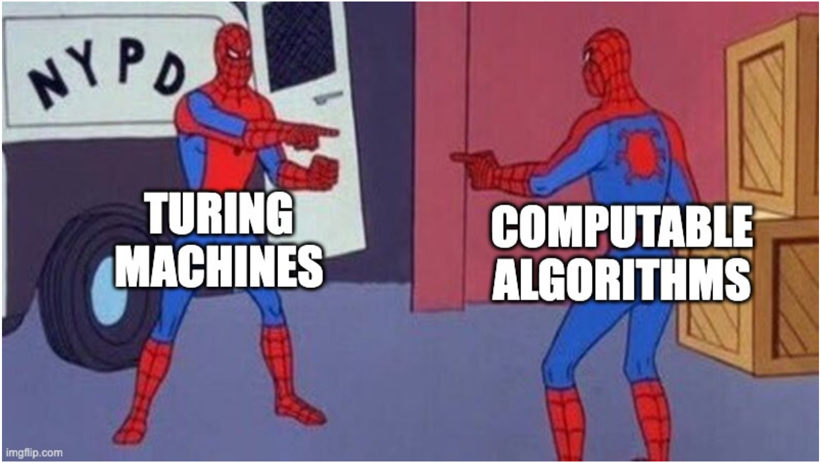
Church-Turing Thesis

- In 1936, Turing came up with the Turing machine while Alonzo Church simultaneously invented lambda calculus
- Turing showed that these two formulations described the *same* class of functions.
- Since then, several other models of computation have been proposed, and they all turned out to be equivalent to Turing machines.
    - We have yet to define a machine or programming language that is *more* powerful than a Turing machine

Church-Turing Thesis

The **Church-Turing Thesis** states that Turing machines (and all equivalent models) correspond our intuitive notion of what an "algorithm" is

Church-Turing Thesis



Church-Turing Thesis

The **Church-Turing Thesis** states that Turing machines (and all equivalent models) correspond our intuitive notion of what an "algorithm" is

- Any task that can be solved using a mechanical procedure can be solved using a Turing machine

Church-Turing Thesis

The **Church-Turing Thesis** states that Turing machines (and all equivalent models) correspond our intuitive notion of what an "algorithm" is

- Any task that can be solved using a mechanical procedure can be solved using a Turing machine
- This is not a theorem or even a mathematically precise statement - but we accept it as true because we have yet to find any satisfying counterexamples

Church-Turing Thesis

- To show that a language can be decided or recognized by a Turing machine, a high-level algorithmic description will suffice

Church-Turing Thesis

- To show that a language can be decided or recognized by a Turing machine, a high-level algorithmic description will suffice
- We can appeal to the Church-Turing thesis so say that whatever algorithm we describe can be implemented on a TM

Church-Turing Thesis

- To show that a language can be decided or recognized by a Turing machine, a high-level algorithmic description will suffice
- We can appeal to the Church-Turing thesis so say that whatever algorithm we describe can be implemented on a TM
- No more tedious formal descriptions

Church-Turing Thesis

- To show that a language can be decided or recognized by a Turing machine, a high-level algorithmic description will suffice
- We can appeal to the Church-Turing thesis so say that whatever algorithm we describe can be implemented on a TM
- No more tedious formal descriptions
- We still use tape-level descriptions to show that a new machine is equivalent to a TM

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

We can decide L with the following algorithm

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

We can decide L with the following algorithm

1. Loop through every pair of nodes u, v and check that they are connected by an edge

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

We can decide L with the following algorithm

1. Loop through every pair of nodes u, v and check that they are connected by an edge
2. If any two nodes are not connected, reject

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

We can decide L with the following algorithm

1. Loop through every pair of nodes u, v and check that they are connected by an edge
2. If any two nodes are not connected, reject
3. If we find that every pair of nodes is connected, accept

Turing Machines

Let's show that the following language is Turing-decidable

$$L = \{\langle G \rangle | G \text{ is a complete graph } \}$$

We can decide L with the following algorithm

1. Loop through every pair of nodes u, v and check that they are connected by an edge
2. If any two nodes are not connected, reject
3. If we find that every pair of nodes is connected, accept

By the Church-Turing thesis we can implement this algorithm on a TM. This algorithm will always halt. Thus L is Turing-decidable.

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

- This will demonstrate that Turing machines are strictly more powerful than any other machine we've covered

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

- This will demonstrate that Turing machines are strictly more powerful than any other machine we've covered
- For practice, let's give both a high-level description and a tape-level description

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

High level description:

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

High level description:

1. Make sure the a's, b's, and c's are in the right order

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

High level description:

1. Make sure the a's, b's, and c's are in the right order
2. Count the a's, b's, and c's. If they are equal, accept. Otherwise, reject.

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Tape level description

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Tape level description

1. Scan left to right and check that a's, b's, and c's are in the right order. If not, reject.

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Tape level description

1. Scan left to right and check that a's, b's, and c's are in the right order. If not, reject.
2. Find the left-most a and erase it. Scan right in search of a matching b and a matching c.

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Tape level description

1. Scan left to right and check that a's, b's, and c's are in the right order. If not, reject.
2. Find the left-most a and erase it. Scan right in search of a matching b and a matching c.
    2.1 If we find the matching b and c, erase them and repeat the process.

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Tape level description

1. Scan left to right and check that a's, b's, and c's are in the right order. If not, reject.
2. Find the left-most a and erase it. Scan right in search of a matching b and a matching c.
    2.1 If we find the matching b and c, erase them and repeat the process.
    2.2 If we don't find the matching b and c, reject

Turing Machine Example

Let's show that $\{a^n b^n c^n | n \geq 0\}$ is Turing-decidable

Tape level description

1. Scan left to right and check that a's, b's, and c's are in the right order. If not, reject.
2. Find the left-most a and erase it. Scan right in search of a matching b and a matching c.
    2.1 If we find the matching b and c, erase them and repeat the process.
    2.2 If we don't find the matching b and c, reject
3. If the tape becomes empty, accept.