Turing Machines and Decidability

Purushothama B R

Department of Computer Science and Engineering
National Institute of Technology Goa ,INDIA

December 7, 2020

I want to Begin With... J K Rowling's Quote

It is impossible to live without failing at something, unless you live so cautiously that you might has well not have lived at all, in which case you have failed by default.

Turing Recognizable

Turing Recognizable

A language is Turing-recognizable if some Turing machine recognizes it.

Turing Machine

Turing Machine

- When we start a Turing machine on an input, three outcomes are possible.

Turing Machine

- When we start a Turing machine on an input, three outcomes are possible.
  - The machine may accept,

Turing Machine

- When we start a Turing machine on an input, three outcomes are possible.
  - The machine may accept,
  - Reject, or

Turing Machine

- When we start a Turing machine on an input, three outcomes are possible.
  - The machine may accept,
  - Reject, or
  - Loop.

Turing Machine

- When we start a Turing machine on an input, three outcomes are possible.
    - The machine may accept,
    - Reject, or
    - Loop.
- By loop we mean that the machine simply does not halt.

Turing Machine

- When we start a Turing machine on an input, three outcomes are possible.
  - The machine may accept,
  - Reject, or
  - Loop.
- By loop we mean that the machine simply does not halt.
- Looping may entail any simple or complex behavior that never leads to a halting state.

Contd..

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine **that is looping** from one that is **merely taking a long time** is **difficult.**

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine **that is looping** from one that is **merely taking a long time** is **difficult.**
- For this reason, we prefer Turing machines **that halt on all inputs**;

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine **that is looping** from one that is **merely taking a long time** is **difficult.**
- For this reason, we prefer Turing machines **that halt on all inputs**;
- Such machines never loop.

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine **that is looping** from one that is **merely taking a long time** is **difficult.**
- For this reason, we prefer Turing machines **that halt on all inputs**;
- Such machines never loop.
- These machines are called **deciders**

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine **that is looping** from one that is **merely taking a long time** is **difficult.**
- For this reason, we prefer Turing machines **that halt on all inputs**;
- Such machines never loop.
- These machines are called **deciders**
  - Since, they always make a decision to accept or reject.

Contd..

- A Turing machine **M** can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- Sometimes distinguishing a machine **that is looping** from one that is **merely taking a long time** is **difficult.**
- For this reason, we prefer Turing machines **that halt on all inputs**;
- Such machines never loop.
- These machines are called **deciders**
  - Since, they always make a decision to accept or reject.
- A **decider** that recognizes some language also is said to **decide** that language.

Decidable

Decidable

Turing Decidable

A language is **Turing-decidable** or simply **decidable** if some Turing machine decides it.

Decidable

Turing Decidable

A language is **Turing-decidable** or simply **decidable** if some Turing machine decides it.

- Every decidable language is Turing-recognizable.

Multi-tape Turing Machine

Theorem

Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof Sketch

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol $\#$ as a delimiter to separate the contents of the different tapes.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes,

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, **S** must keep track of the locations of the heads.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol $\#$ as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, **S** must keep track of the locations of the heads.
- It does so by writing a **tape symbol with a dot above it** to mark

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, **S** must keep track of the locations of the heads.
- It does so by writing a **tape symbol with a dot above it** to mark the place where the head on that tape would be.

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, **S** must keep track of the locations of the heads.
- It does so by writing a **tape symbol with a dot above it** to mark the place where the head on that tape would be.
- Think of these as "virtual" tapes and heads.

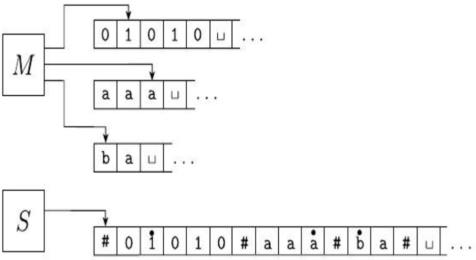
Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, **S** must keep track of the locations of the heads.
- It does so by writing a **tape symbol with a dot above it** to mark the place where the head on that tape would be.
- Think of these as "virtual" tapes and heads.
- As before, the "dotted" tape symbols are simply

Proof Sketch

- We show how to convert a **multitape TM M** to an equivalent **single tape TM S**.
- The key idea is to show how to simulate **M** with **S**.
- Say that **M** has **k** tapes.
- Then S simulates the effect of **k** tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, **S** must keep track of the locations of the heads.
- It does so by writing a **tape symbol with a dot above it** to mark the place where the head on that tape would be.
- Think of these as "virtual" tapes and heads.
- As before, the "dotted" tape symbols are simply new symbols that have been added to the tape alphabet.

Representing three tapes with Single tape



Proof Contd..

Proof Contd..

- S on input S $w = w_1 \dots w_n$:

Proof Contd..

- S on input S $w = w_1 \ldots w_n$:
  1. First S puts its tape into the format that represents all k tapes of M .

Proof Contd..

- S on input S $w = w_1 \ldots w_n$:
  1. First S puts its tape into the format that represents all k tapes of M .
  2. The formatted tape contains

$$\# \dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#.$$

Simulate a Move

Simulate a Move

- To simulate a single move,

Simulate a Move

- To simulate a single move,
  - In order to determine the symbols under the virtual heads,

Simulate a Move

- To simulate a single move,
  - In order to determine the symbols under the virtual heads,
  - S scans its tape from the first #, which marks the left-hand end,

Simulate a Move

- To simulate a single move,
  - In order to determine the symbols under the virtual heads,
  - S scans its tape from the first $\#$, which marks the left-hand end,
  - to the $(k+1)^{st}\#$, which marks the right-hand end,

Simulate a Move

- To simulate a single move,
    - In order to determine the symbols under the virtual heads,
    - S scans its tape from the first #, which marks the left-hand end,
    - to the $(k+1)^{st}\#$, which marks the right-hand end,
    - Then S makes a second pass

Simulate a Move

- To simulate a single move,
  - In order to determine the symbols under the virtual heads,
  - S scans its tape from the first #, which marks the left-hand end,
  - to the $(k+1)^{st}\#$, which marks the right-hand end,
  - Then S makes a second pass
    - To update the tapes according to the way that M's transition function dictates.

Contd..

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
- So S writes a blank symbol on this tape cell

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
- So S writes a blank symbol on this tape cell and shifts the tape contents,

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
- So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
- So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right.

Contd..

- If at any point S moves one of the virtual heads to the right onto a $\#$.
- This action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
- So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right.
- Then it continues the simulation as before.

Complexity

Theorem

The time taken by the one-tape TM N to simulate n moves of the $k - tape$ TM M is $O(n^2)$.

Complexity

Theorem

The time taken by the one-tape TM N to simulate n moves of the $k - tape$ TM M is $O(n^2)$.

Proof

Exercise

THE DEFINITION OF ALGORITHM

THE DEFINITION OF ALGORITHM

- Informally speaking, **an algorithm is a collection of simple instructions** for carrying out some task.

THE DEFINITION OF ALGORITHM

- Informally speaking, **an algorithm is a collection of simple instructions** for carrying out some task.
- Commonplace in everyday life,

THE DEFINITION OF ALGORITHM

- Informally speaking, **an algorithm is a collection of simple instructions** for carrying out some task.
- Commonplace in everyday life, algorithms sometimes are called **procedures** or **recipes.**

THE DEFINITION OF ALGORITHM

- Informally speaking, **an algorithm is a collection of simple instructions** for carrying out some task.
- Commonplace in everyday life, algorithms sometimes are called **procedures** or **recipes.**
- Algorithms also play an important role in mathematics.

THE DEFINITION OF ALGORITHM

- Informally speaking, **an algorithm is a collection of simple instructions** for carrying out some task.
- Commonplace in everyday life, algorithms sometimes are called **procedures** or **recipes.**
- Algorithms also play an important role in mathematics.
- Ancient mathematical literature contains descriptions of algorithms for a variety of tasks,

THE DEFINITION OF ALGORITHM

- Informally speaking, **an algorithm is a collection of simple instructions** for carrying out some task.
- Commonplace in everyday life, algorithms sometimes are called **procedures** or **recipes.**
- Algorithms also play an important role in mathematics.
- Ancient mathematical literature contains descriptions of algorithms for a variety of tasks,
  - Finding prime numbers and greatest common divisors.

Precise

Precise

- Even though algorithms have had a long history in mathematics,

Precise

- Even though algorithms have had a long history in mathematics,
- The notion of algorithm itself was not defined precisely until the twentieth century.

Precise

- Even though algorithms have had a long history in mathematics,
- The notion of algorithm itself was not defined precisely until the twentieth century.
- Before that, mathematicians had an intuitive notion of what algorithms were.

Precise

- Even though algorithms have had a long history in mathematics,
- The notion of algorithm itself was not defined precisely until the twentieth century.
- Before that, mathematicians had an intuitive notion of what algorithms were.
- And relied upon that notion when using and describing them.

Precise

- Even though algorithms have had a long history in mathematics,
- The notion of algorithm itself was not defined precisely until the twentieth century.
- Before that, mathematicians had an intuitive notion of what algorithms were.
- And relied upon that notion when using and describing them.
- But that intuitive notion was insufficient for gaining a deeper understanding of algorithms.

Story: HILBERT's Problems

Story: HILBERT's Problems

- In 1900, mathematician David Hilbert delivered

Story: HILBERT's Problems

- In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris.

Story: HILBERT's Problems

- In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris.
- In his lecture, he identified **23 mathematical problems**

Story: HILBERT's Problems

- In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris.
- In his lecture, he identified **23 mathematical problems**
- And **posed** them as a challenge for the coming century.

- In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris.
- In his lecture, he identified **23 mathematical problems**
- And **posed** them as a challenge for the coming century.
- **The tenth problem on his list concerned algorithms.**

Fundamentals: Polynomials

Fundamentals: Polynomials

- A polynomial is

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.
- A root of a polynomial is an

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.
- A root of a polynomial is an
  - Assignment of values to its variables so that the value of the polynomial is 0.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.
- A root of a polynomial is an
  - Assignment of values to its variables so that the value of the polynomial is 0.
  - The above polynomial has a root at $x = 5, y = 3, and z = 0$.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.
- A root of a polynomial is an
  - Assignment of values to its variables so that the value of the polynomial is 0.
  - The above polynomial has a root at $x = 5, y = 3, andz = 0$.
  - This root is an integral root.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.
- A root of a polynomial is an
  - Assignment of values to its variables so that the value of the polynomial is 0.
  - The above polynomial has a root at $x = 5, y = 3, and z = 0$.
  - This root is an integral root.
    - Since, all the variables are assigned integer values.

Fundamentals: Polynomials

- A polynomial is
  - a sum of terms,
  - where each term is a product of certain variables
  - and a constant called coefficient.
- Example $6xxxyzz = 6x^3yz^2$ is a term with co-efficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables x, y, z.
- Consider only coefficients that are integers.
- A root of a polynomial is an
  - Assignment of values to its variables so that the value of the polynomial is 0.
  - The above polynomial has a root at $x = 5, y = 3, andz = 0$.
  - This root is an integral root.
    - Since, all the variables are assigned integer values.
  - Some polynomials have an integral root and some do not.

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

- He did not use the term algorithm but rather

Hilbert's Tenth Problem

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

- He did not use the term algorithm but rather
- "**a process according to which it can be determined by a finite number of operations.**"

Hilbert's Tenth Problem

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

- He did not use the term algorithm but rather
- “**a process according to which it can be determined by a finite number of operations.”**
- Interestingly, in the way he phrased this problem,

Hilbert's Tenth Problem

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

- He did not use the term algorithm but rather
- "**a process according to which it can be determined by a finite number of operations.**"
- Interestingly, in the way he phrased this problem,
  - Hilbert explicitly asked that an algorithm be "devised."

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

- He did not use the term algorithm but rather
- “**a process according to which it can be determined by a finite number of operations.**”
- Interestingly, in the way he phrased this problem,
  - Hilbert explicitly asked that an algorithm be “devised.”
  - Thus he apparently assumed that such an algorithm **must exist.**

Hilbert's Tenth Problem

Hilbert's Tenth Problem

Devise an algorithm that tests whether a polynomial has an integral root.

- He did not use the term algorithm but rather
- "**a process according to which it can be determined by a finite number of operations.**"
- Interestingly, in the way he phrased this problem,
  - Hilbert explicitly asked that an algorithm be "devised."
  - Thus he apparently assumed that such an algorithm **must exist.**
  - Someone need only find it.

Unfortunately!!!

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.
- The intuitive concept may have been adequate for giving algorithms for certain tasks.

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.
- The intuitive concept may have been adequate for giving algorithms for certain tasks.
- It was useless for showing that no algorithm exists for a particular task.

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.
- The intuitive concept may have been adequate for giving algorithms for certain tasks.
- It was useless for showing that no algorithm exists for a particular task.
- Proving that an algorithm does not exist requires having a clear definition of algorithm.

Unfortunately!!!

- As we now know, **no algorithm exists for this task;**
- It is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.
- The intuitive concept may have been adequate for giving algorithms for certain tasks.
- It was useless for showing that no algorithm exists for a particular task.
- Proving that an algorithm does not exist requires having a clear definition of algorithm.
- Progress on the tenth problem had to wait for that definition.

Definition came!!!

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**
- Church used a notational system called the λ-calculus to define algorithms.

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**
- Church used a notational system called the λ-calculus to define algorithms.
- Turing did it with his **"machines."**

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**
- Church used a notational system called the λ-calculus to define algorithms.
- Turing did it with his **"machines."**
- These two definitions were shown to be equivalent.

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**
- Church used a notational system called the λ-calculus to define algorithms.
- Turing did it with his **"machines."**
- These two definitions were shown to be equivalent.
- This connection between the informal notion of algorithm

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**
- Church used a notational system called the λ-calculus to define algorithms.
- Turing did it with his **“machines.”**
- These two definitions were shown to be equivalent.
- This connection between the informal notion of algorithm and the

Definition came!!!

- The definition came in the **1936** papers of **Alonzo Church and Alan Turing.**
- Church used a notational system called the λ-calculus to define algorithms.
- Turing did it with his **"machines."**
- These two definitions were shown to be equivalent.
- This connection between the informal notion of algorithm and the precise definition has come to be called the **ChurchTuring thesis.**

Church-Turing Thesis

Intuitive notion of algorithms	equals	Turing machine algorithms

THANK YOU