Reducibility

Purushothama B R

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

December 7, 2020

I want to Begin With... Michael Jordan's Quote

I've missed more than 9000 shots in my career. I've lost almost 300 games. 26 times, I've been trusted to take the game winning shot and missed. I've failed over and over and over again in my life. And that is why I succeed.

Here We Are..

Here We Are..

- We established the Turing machine as our model of a general purpose computer.

Here We Are..

- We established the Turing machine as our model of a general purpose computer.
- We have seen several examples of problems that are solvable on a Turing machine

Here We Are..

- We established the Turing machine as our model of a general purpose computer.
- We have seen several examples of problems that are solvable on a Turing machine
- Gave one example of a problem, A_{TM} ,

Here We Are..

- We established the Turing machine as our model of a general purpose computer.
- We have seen several examples of problems that are solvable on a Turing machine
- Gave one example of a problem, A_{TM} ,that is **computationally unsolvable.**

Here We Are..

- We established the Turing machine as our model of a general purpose computer.
- We have seen several examples of problems that are solvable on a Turing machine
- Gave one example of a problem, A_{TM} ,that is **computationally unsolvable.**
- Let us examine several additional unsolvable problems.

Here We Are..

- We established the Turing machine as our model of a general purpose computer.
- We have seen several examples of problems that are solvable on a Turing machine
- Gave one example of a problem, A_{TM} ,that is **computationally unsolvable.**
- Let us examine several additional unsolvable problems.
- I will introduce a **primary method for proving that problems are computationally unsolvable.**

Here We Are..

- We established the Turing machine as our model of a general purpose computer.
- We have seen several examples of problems that are solvable on a Turing machine
- Gave one example of a problem, A_{TM} ,that is **computationally unsolvable.**
- Let us examine several additional unsolvable problems.
- I will introduce a **primary method for proving that problems are computationally unsolvable.**
- It is called **reducibility.**

Reduction

Reduction

Reduction

Reduction

- A **reduction** is

Reduction

- A **reduction** is
  - A way of **converting one problem to another problem**

Reduction

Reduction

- A **reduction** is
  - A way of **converting one problem to another problem** in such a way that

Reduction

Reduction

- A **reduction** is
  - A way of **converting one problem to another problem** in such a way that
  - A **solution to the second problem**

Reduction

- A **reduction** is
  - A way of **converting one problem to another problem** in such a way that
  - A **solution to the second problem**can be used

Reduction

Reduction

- A **reduction** is
  - A way of **converting one problem to another problem** in such a way that
  - A **solution to the second problem**can be used **to solve the first problem.**

Reduction

- A **reduction** is
  - A way of **converting one problem to another problem** in such a way that
  - A **solution to the second problem**can be used **to solve the first problem.**

- Such reducibilities come up often in everyday life.

Example

Example

- Suppose that you want to **find your way** around a new city.

Example

- Suppose that you want to **find your way** around a new city.
- You know that doing so would be easy,

Example

- Suppose that you want to **find your way** around a new city.
- You know that doing so would be easy, if you had a **map.**

Example

- Suppose that you want to **find your way** around a new city.
- You know that doing so would be easy, if you had a **map.**
- Thus, you can **reduce** the problem of finding your way around the city

Example

- Suppose that you want to **find your way** around a new city.
- You know that doing so would be easy, if you had a **map.**
- Thus, you can **reduce** the problem of finding your way around the city to the problem of obtaining a map of the city.

Example-II: Kidding

Example-II: Kidding

- Suppose that you want to find a solution to assignment problem.

Example-II: Kidding

- Suppose that you want to find a solution to assignment problem.
- You know that doing so would be easy

Example-II: Kidding

- Suppose that you want to find a solution to assignment problem.
- You know that doing so would be easy if you had a search engine for searching solution.

Example-II: Kidding

- Suppose that you want to find a solution to assignment problem.
- You know that doing so would be easy if you had a search engine for searching solution.
- Thus, you can **reduce** the problem of finding solution to assignment problem

Example-II: Kidding

- Suppose that you want to find a solution to assignment problem.
- You know that doing so would be easy if you had a search engine for searching solution.
- Thus, you can **reduce** the problem of finding solution to assignment problem to the problem of searching in google.

Reducibility

- Reducibility always involves two problems,

Reducibility

- Reducibility always involves two problems, which I call A and B.

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B,**

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can **use a solution to B to solve A.**

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can **use a solution to B to solve A.**
- So in our city example,

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can **use a solution to B to solve A.**
- So in our city example,
  - A is the problem of finding your way around the city.

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can **use a solution to B to solve A.**
- So in our city example,
  - A is the problem of finding your way around the city.
  - B is the problem of obtaining a map.

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can **use a solution to B to solve A.**
- So in our city example,
  - A is the problem of finding your way around the city.
  - B is the problem of obtaining a map.
- Note: reducibility says nothing about solving A or B alone.

Reducibility

- Reducibility always involves two problems, which I call A and B.
- **If A reduces to B**, we can **use a solution to B to solve A.**
- So in our city example,
  - A is the problem of finding your way around the city.
  - B is the problem of obtaining a map.
- Note: reducibility says nothing about solving A or B alone.
- But it says only about the **solvability of A** in the **presence of a solution to B.**

Other Examples of Reducibility

Other Examples of Reducibility

- The problem of traveling from Boston to Paris

Other Examples of Reducibility

- The problem of traveling from Boston to Paris
  - Reduces to the problem of **a plane ticket** between the two cities.

Other Examples of Reducibility

- The problem of traveling from Boston to Paris
  - Reduces to the problem of **a plane ticket** between the two cities.
  - That problem in turn **reduces to** the problem of **earning the money** for the ticket.

Other Examples of Reducibility

- The problem of traveling from Boston to Paris
  - Reduces to the problem of **a plane ticket** between the two cities.
  - That problem in turn **reduces to** the problem of **earning the money** for the ticket.
  - And that problem **reduces to** the problem of **finding a job.**

Reducibility in Mathematical Problems

- Reducibility also occurs in **mathematical problems.**

Reducibility in Mathematical Problems

- Reducibility also occurs in **mathematical problems.**
- The problem of measuring the area of a rectangle reduces

Reducibility in Mathematical Problems

- Reducibility also occurs in **mathematical problems.**
- The problem of measuring the area of a rectangle reduces
  - To the problem of measuring its length and width.

Reducibility in Mathematical Problems

- Reducibility also occurs in **mathematical problems.**
- The problem of measuring the area of a rectangle reduces
  - To the problem of measuring its length and width.
- The problem of solving a system of linear equations

Reducibility in Mathematical Problems

- Reducibility also occurs in **mathematical problems.**
- The problem of measuring the area of a rectangle reduces
  - To the problem of measuring its length and width.
- The problem of solving a system of linear equations
  - Reduces to the problem of inverting a matrix.

Role of Reducibility

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and **B is decidable,**

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and **B is decidable,**
    - **A also is decidable.**

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and **B is decidable,**
    - **A also is decidable.**
- Equivalently, **if A is undecidable** and

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and **B is decidable,**
    - **A also is decidable.**
- Equivalently, **if A is undecidable** and **reducible to B,**

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and **B is decidable,**
    - **A also is decidable.**
- Equivalently, **if A is undecidable** and **reducible to B,**
  - **B is undecidable.**

Role of Reducibility

- Reducibility plays an important role in **classifying problems by decidability.**
- And in **complexity theory** as well.
- When A is reducible to B,
  - Solving A **cannot be harder** than solving B.
  - Because a solution to B gives a solution to A.
- In terms of computability theory,
  - If **A is reducible to B** and **B is decidable,**
    - **A also is decidable.**
- Equivalently, **if A is undecidable** and **reducible to B,**
  - **B is undecidable.**
- This last version is key to proving that various problems are undecidable.

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$
  - The problem of determining whether a Turing machine halts

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$
  - The problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$
  - The problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.
- This problem is widely known as the **Halting Problem**.

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$
  - The problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.
- This problem is widely known as the **Halting Problem**.
- We use the **undecidability of** A_{TM}

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$
  - The problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.
- This problem is widely known as the **Halting Problem**.
- We use the **undecidability of** A_{TM} to prove the undecidability of $HALT_{TM}$

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY

- We have already established the undecidability of A_{TM}.
  - It is the problem of determining whether a Turing machine accepts a given input.
- Lets consider a related problem $HALT_{TM}$
  - The problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.
- This problem is widely known as the **Halting Problem**.
- We use the **undecidability of** A_{TM} to prove the undecidability of $HALT_{TM}$
  - By **reducing** A_{TM} **to** $HALT_{TM}$.

Halting Problem

Halting Problem

- Let,
  $HALT_{TM} = \{< M, w > | M \text{ is a TM and M halts on input } w\}$.

Halting Problem

- Let,

  $HALT_{TM} = \{< M, w > | M \text{ is a TM and M halts on input } w\}$.

Theorem 5.1

$HALT_{TM}$ is undecidable.

Proof Idea

Proof Idea

- This proof is by contradiction.

Proof Idea

- This proof is by contradiction.
- We assume that $HALT_{TM}$ is decidable

Proof Idea

- This proof is by contradiction.
- We assume that $HALT_{TM}$ is decidable
- And use that assumption to show that A_{TM} is decidable

Proof Idea

- This proof is by contradiction.
- We assume that $HALT_{TM}$ is decidable
- And use that assumption to show that A_{TM} is decidable
- Contradicting the assumption that A_{TM} is undecidable (**We know it!!!**) .

Proof Idea

- This proof is by contradiction.
- We assume that $HALT_{TM}$ is decidable
- And use that assumption to show that A_{TM} is decidable
- Contradicting the assumption that A_{TM} is undecidable (**We know it!!!**) .
- The key idea is to show that A_{TM} is reducible to $HALT_{TM}$.

Proof

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S**

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
  - Pretend that you are S.

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
  - Pretend that you are S.
  - Your task is to decide A_{TM} .

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
  - Pretend that you are S.
  - Your task is to decide A_{TM} .
  - You are given an input of the form $< M, w >$.

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
  - Pretend that you are S.
  - Your task is to decide A_{TM} .
  - You are given an input of the form $< M, w >$.
  - You must output accept

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
    - Pretend that you are S.
    - Your task is to decide A_{TM} .
    - You are given an input of the form $< M, w >$.
    - You must output accept if M accepts w.

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
  - Pretend that you are S.
  - Your task is to decide A_{TM} .
  - You are given an input of the form $< M, w >$.
  - You must output accept if M accepts w.
  - And you must output reject,

Proof

- Lets assume that we have a **TM R** that decides $HALT_{TM}$
- Then, we use R to construct **S** (TM that decides A_{TM}) .
- To get a feel for the way to construct S,
  - Pretend that you are S.
  - Your task is to decide A_{TM} .
  - You are given an input of the form $< M, w >$.
  - You must output accept if M accepts w.
  - And you must output reject, if M loops or rejects on w.

Proof

Proof

- Try simulating M on w.

Proof

- Try simulating M on w.
- If it accepts or rejects, do the same.

Proof

- Try simulating M on w.
- If it accepts or rejects, do the same.
- **But you may not be able to determine whether M is looping.**

Proof

- Try simulating M on w.
- If it accepts or rejects, do the same.
- **But you may not be able to determine whether M is looping.**
  - In that case your **simulation will not terminate.**

Proof

- Try simulating M on w.
- If it accepts or rejects, do the same.
- **But you may not be able to determine whether M is looping.**
  - In that case your **simulation will not terminate.**
- Thats bad because you are a decider and

Proof

- Try simulating M on w.
- If it accepts or rejects, do the same.
- **But you may not be able to determine whether M is looping.**
    - In that case your **simulation will not terminate.**
- Thats bad because you are a decider and thus never permitted to loop.

Proof

- Try simulating M on w.
- If it accepts or rejects, do the same.
- **But you may not be able to determine whether M is looping.**
  - In that case your **simulation will not terminate.**
- Thats bad because you are a decider and thus never permitted to loop.
- So this idea by itself does not work.

What to Do?

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on
  - You can do the simulation without any danger of looping.

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on
  - You can do the simulation without any danger of looping.
- Thus, if TM R exists, we can decide A_{TM}

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on
  - You can do the simulation without any danger of looping.
- Thus, if TM R exists, we can decide A_{TM}
- But we know that A_{TM} is undecidable.

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on
  - You can do the simulation without any danger of looping.
- Thus, if TM R exists, we can decide A_{TM}
- But we know that A_{TM} is undecidable.
- By virtue of this contradiction,

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on
  - You can do the simulation without any danger of looping.
- Thus, if TM R exists, we can decide A_{TM}
- But we know that A_{TM} is undecidable.
- By virtue of this contradiction,
  - We can conclude that R does not exist.

What to Do?

- Instead, use the assumption that you have TM **R** that decides $HALT_{TM}$.
- With R, you can test whether M halts on w.
- If R indicates that M doesnt halt on w,
  - reject because $< M, w >$ isn't in A_{TM} .
- However, if R indicates that M does halt on
  - You can do the simulation without any danger of looping.
- Thus, if TM R exists, we can decide A_{TM}
- But we know that A_{TM} is undecidable.
- By virtue of this contradiction,
  - We can conclude that R does not exist.
- Therefore, $HALT_{TM}$ is undecidable.

Wrap up the Proof details.

Wrap up the Proof details.

- Lets assume for the purpose of obtaining a contradiction that R decides $HALT_{TM}$.

Wrap up the Proof details.

- Lets assume for the purpose of obtaining a contradiction that R decides $HALT_{TM}$.
- We construct TM S to decide A_{TM}. S operates as below.

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

- Lets assume for the purpose of obtaining a contradiction that R decides $HALT_{TM}$.
- We construct TM S to decide A_{TM}. S operates as below.

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

- Clearly, if R decides $HALT_{TM}$, then S decides A_{TM}

Wrap up the Proof details.

- Lets assume for the purpose of obtaining a contradiction that R decides $HALT_{TM}$.
- We construct TM S to decide A_{TM}. S operates as below.

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

- Clearly, if R decides $HALT_{TM}$, then S decides A_{TM}
- Because A_{TM} is undecidable,

Wrap up the Proof details.

- Lets assume for the purpose of obtaining a contradiction that R decides $HALT_{TM}$.
- We construct TM S to decide A_{TM}. S operates as below.

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

- Clearly, if R decides $HALT_{TM}$, then S decides A_{TM}
- Because A_{TM} is undecidable, $HALT_{TM}$ also must be undecidable.

THANK YOU