Theory of Computation
Turing Reducibility

Arjun Chandrasekhar

Programs can Create other Programs

Programs can Create other Programs

- Can we write a java program that creates another java program?

Programs can Create other Programs

- Can we write a java program that creates another java program?
- Can we decide what the new program should do based on what command line argument the original program received?

Programs can Create other Programs

- Can we write a java program that creates another java program?
- Can we decide what the new program should do based on what command line argument the original program received?
- After creating a new program, can we analyze that program?

Programs can create other programs

- Consider the program makeProgram.java

Programs can create other programs

- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input

Programs can create other programs

- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input
  2. makeProgram.java creates a java source code file called oneString.java

Programs can create other programs

- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input
  2. makeProgram.java creates a java source code file called oneString.java
     2.1 oneString.java takes an input string s

Programs can create other programs

- Consider the program makeProgram.java
    1. makeProgram.java takes a string w as input
    2. makeProgram.java creates a java source code file called oneString.java
        2.1 oneString.java takes an input string s
        2.2 print ACCEPT if $s = w$
        print REJECT if $s \neq w$
        Note that w is a hard-coded constant

Programs can analyze new programs

Programs can analyze new programs

- Let's say we have a program called even.java

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java
  1. makeProgram.java takes a string *w* as input
  2. Creates a java source code file called oneString.java

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input
  2. Creates a java source code file called oneString.java
     2.1 oneString.java takes an input string s

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input
  2. Creates a java source code file called oneString.java
     2.1 oneString.java takes an input string s
     2.2 print ACCEPT if $s = w$
     print REJECT if $s \neq w$
     Note that w is a hard-coded constant

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input
  2. Creates a java source code file called oneString.java
     2.1 oneString.java takes an input string s
     2.2 print ACCEPT if $s = w$
     print REJECT if $s \neq w$
     Note that w is a hard-coded constant
  3. run even.java on oneString.java

Programs can analyze new programs

- Let's say we have a program called even.java
  - even.java checks if another source code file has an even number of characters
- Consider the program makeProgram.java
  1. makeProgram.java takes a string w as input
  2. Creates a java source code file called oneString.java
     2.1 oneString.java takes an input string s
     2.2 print ACCEPT if $s = w$
     print REJECT if $s \neq w$
     Note that w is a hard-coded constant
  3. run even.java on oneString.java

I've put the code on the course website (in python)

Reducibility

Reducibility

- "If I can solve problem B, then I can solve problem A."

Reducibility

- “If I can solve problem B, then I can solve problem A.”
- “So solving problem B is at least as hard as solving problem A”

Reducibility

- If I can obtain a job, I can earn some money

Reducibility

- If I can obtain a job, I can earn some money
  - The problem of earning money can be reduced to the problem of obtaining a job

Reducibility

- If I can obtain a job, I can earn some money
- If I can get to Los Angeles, I can obtain a job

Reducibility

- If I can obtain a job, I can earn some money
- If I can get to Los Angeles, I can obtain a job
    - The problem of obtaining a job can be reduced to the problem of getting to Los Angeles

Reducibility

- If I can obtain a job, I can earn some money
- If I can get to Los Angeles, I can obtain a job
- If I can find a map to Los Angeles, I can get to Los Angeles

Reducibility

- If I can obtain a job, I can earn some money
- If I can get to Los Angeles, I can obtain a job
- If I can find a map to Los Angeles, I can get to Los Angeles
  - The problem of getting to Los Angeles can be reduced to the problem of finding a map to Los Angeles

Reducibility

- If I can obtain a job, I can earn some money
- If I can get to Los Angeles, I can obtain a job
- If I can find a map to Los Angeles, I can get to Los Angeles
- **The problem of earning money can be reduced to the problem of finding a map to Los Angeles**

Reducibility

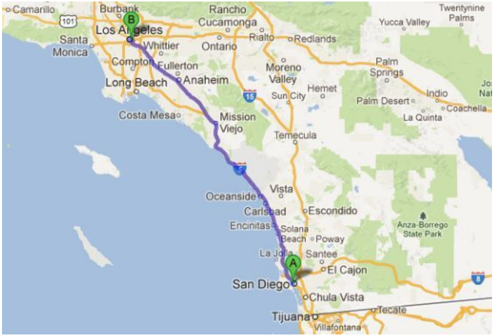
- The problem of earning money can be reduced to the problem of finding a map to Los Angeles

Reducibility

- The problem of earning money can be reduced to the problem of finding a map to Los Angeles
- Suppose I find a map to Los Angeles

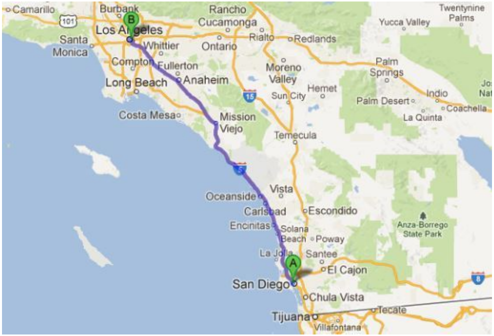
Reducibility

- The problem of earning money can be reduced to the problem of finding a map to Los Angeles
- Suppose I find a map to Los Angeles



Reducibility

- The problem of earning money can be reduced to the problem of finding a map to Los Angeles
- Suppose I find a map to Los Angeles



Can I start earning money?

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead
    - **The problem of getting to Hogsmead can be reduced to the problem of obtaining the Marauder's Map**

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead
  - **The problem of getting to Hogsmead can be reduced to the problem of obtaining the Marauder's Map**

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead
  - **The problem of getting to Hogsmead can be reduced to the problem of obtaining the Marauder's Map**

When we make this statement, we are not claiming that it is possible to actually obtain the Marauder's map. We are just considering the hypothetical scenario in which we could obtain it.

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead
- Now let's say I convince you that getting to Hogsmead is impossible (because it's a fictional place)

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead
- Now let's say I convince you that getting to Hogsmead is impossible (because it's a fictional place)
- **Then you would conclude that obtaining the Marauder's Map is also impossible**

Reducibility and Impossibility

- If I can obtain the Marauder's Map, I can get to Hogsmead
- Now let's say I convince you that getting to Hogsmead is impossible (because it's a fictional place)
- **Then you would conclude that obtaining the Marauder's Map is also impossible**
  - Otherwise we would be able to do something we know is impossible

Turing Reducibilty

Turing Reducibilty

- Let A and B be formal languages

Turing Reducibilty

- Let A and B be formal languages
- Suppose that we prove that if there were a machine M_B to decide B, then we could construct a machine M_A to decide A

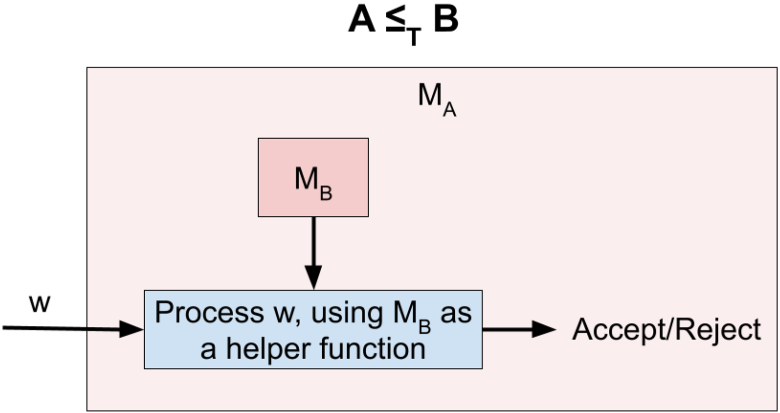
Turing Reducibilty

- Let A and B be formal languages
- Suppose that we prove that if there were a machine M_B to decide B, then we could construct a machine M_A to decide A
- Then we say A is **Turing reducible** (or simply **reducible**) to B

Turing Reducibilty

- Let A and B be formal languages
- Suppose that we prove that if there were a machine M_B to decide B, then we could construct a machine M_A to decide A
- Then we say A is **Turing reducible** (or simply **reducible**) to B
  - We use the notation $A \leq_T B$

Turing Reducibility



If we can decide B, we can decide A

$\mathrm{EVEN} \leq_T \mathrm{ODD}$

Let's prove that $\mathrm{EVEN} \leq_T \mathrm{ODD}$

$$\mathrm{EVEN} = \{w | w \in \mathbb{N}, w \text{ is even}\}$$
$$\mathrm{ODD} = \{w | w \in \mathbb{N}, w \text{ is odd}\}$$

Suppose we have a machine M_{ODD} that decides ODD. We need to construct a machine M_{EVEN} that decides EVEN

$\mathrm{EVEN} \leq_T \mathrm{ODD}$

Let's prove that $\mathrm{EVEN} \leq_T \mathrm{ODD}$

$$\mathrm{EVEN} = \{w | w \in \mathbb{N}, w \text{ is even}\}$$
$$\mathrm{ODD} = \{w | w \in \mathbb{N}, w \text{ is odd}\}$$

Suppose we have a machine M_{ODD} that decides ODD. We need to construct a machine M_{EVEN} that decides EVEN

- Note that n is even $\Leftrightarrow$ n is not odd

EVEN $\leq_T$ ODD

Let's prove that EVEN $\leq_T$ ODD

$$\text{EVEN} = \{w | w \in \mathbb{N}, w \text{ is even}\}$$
$$\text{ODD} = \{w | w \in \mathbb{N}, w \text{ is odd}\}$$

Suppose we have a machine M_{ODD} that decides ODD. We need to construct a machine M_{EVEN} that decides EVEN

1. M_{EVEN} takes an integer $n \in \mathbb{N}$ as input

$\mathrm{EVEN} \leq_T \mathrm{ODD}$

Let's prove that $\mathrm{EVEN} \leq_T \mathrm{ODD}$

$$\mathrm{EVEN} = \{w | w \in \mathbb{N}, w \text{ is even}\}$$
$$\mathrm{ODD} = \{w | w \in \mathbb{N}, w \text{ is odd}\}$$

Suppose we have a machine M_{ODD} that decides ODD. We need to construct a machine M_{EVEN} that decides EVEN

1. M_{EVEN} takes an integer $n \in \mathbb{N}$ as input
2. M_{EVEN} runs M_{ODD} on n

$\mathrm{EVEN} \leq_T \mathrm{ODD}$

Let's prove that $\mathrm{EVEN} \leq_T \mathrm{ODD}$

$$\mathrm{EVEN} = \{w | w \in \mathbb{N}, w \text{ is even}\}$$
$$\mathrm{ODD} = \{w | w \in \mathbb{N}, w \text{ is odd}\}$$

Suppose we have a machine M_{ODD} that decides ODD. We need to construct a machine M_{EVEN} that decides EVEN

1. M_{EVEN} takes an integer $n \in \mathbb{N}$ as input
2. M_{EVEN} runs M_{ODD} on n
    2.1 If M_{ODD} accepts n, then M_{EVEN} rejects n

$\mathrm{EVEN} \leq_T \mathrm{ODD}$

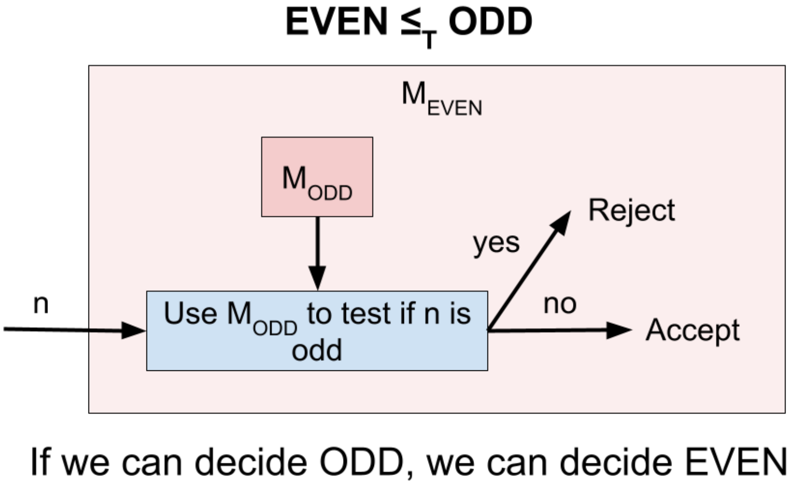
Let's prove that $\mathrm{EVEN} \leq_T \mathrm{ODD}$

$$\mathrm{EVEN} = \{w | w \in \mathbb{N}, w \text{ is even}\}$$
$$\mathrm{ODD} = \{w | w \in \mathbb{N}, w \text{ is odd}\}$$

Suppose we have a machine M_{ODD} that decides ODD. We need to construct a machine M_{EVEN} that decides EVEN

1. M_{EVEN} takes an integer $n \in \mathbb{N}$ as input
2. M_{EVEN} runs M_{ODD} on n
   - 2.1 If M_{ODD} accepts n, then M_{EVEN} rejects n
   - 2.2 If M_{ODD} rejects n then M_{EVEN} accepts n

EVEN $\leq_T$ ODD



$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D \rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

Prove that $\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D \rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

Prove that $\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

- Assume we have a machine M_{EQ} which decides $\mathrm{EQ}_{\mathrm{DFA}}$

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D \rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$
$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$

Prove that $\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

- Assume we have a machine M_{EQ} which decides $\mathrm{EQ}_{\mathrm{DFA}}$
- Show how we can construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D \rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$

$\mathrm{ALL_{DFA}} \leq_T \mathrm{EQ_{DFA}}$

Consider the following two languages

$$\mathrm{ALL_{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ_{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs}, L(D_1) = L(D_2)\}$$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ_{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL_{DFA}}$
- Let D_1 be a DFA

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$
- Let D_1 be a DFA
- Let D_2 be a DFA that recognizes Σ^*

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$
$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs}, L(D_1) = L(D_2)\}$$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$
- Let D_1 be a DFA
- Let D_2 be a DFA that recognizes Σ^*
- $L(D_1) = \Sigma^* \Leftrightarrow L(D_1) = L(D_2)$

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D \rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$

$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$

1. M_A takes $\langle D \rangle$ as input

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$
$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs}, L(D_1) = L(D_2)\}$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$

1. M_A takes $\langle D\rangle$ as input
2. M_A creates a DFA D_2 which recognizes Σ^*

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$
$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$

1. M_A takes $\langle D\rangle$ as input
2. M_A creates a DFA D_2 which recognizes Σ^*
3. M_A runs M_{EQ} on $\langle D, D_2\rangle$

$\text{ALL}_{\text{DFA}} \leq_T \text{EQ}_{\text{DFA}}$

Consider the following two languages

$\text{ALL}_{\text{DFA}} = \{\langle D \rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$
$\text{EQ}_{\text{DFA}} = \{\langle D_1, D_2 \rangle | D_1, D_2 \text{ are DFAs}, L(D_1) = L(D_2)\}$

- Assume we have a machine M_{EQ} that decides EQ_{DFA}.
- We need to construct a machine M_A that decides ALL_{DFA}

1. M_A takes $\langle D \rangle$ as input
2. M_A creates a DFA D_2 which recognizes Σ^*
3. M_A runs M_{EQ} on $\langle D, D_2 \rangle$
    3.1 If M_{EQ} accepts $\langle D, D_2 \rangle$ then M_A accepts $\langle D \rangle$

$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

Consider the following two languages

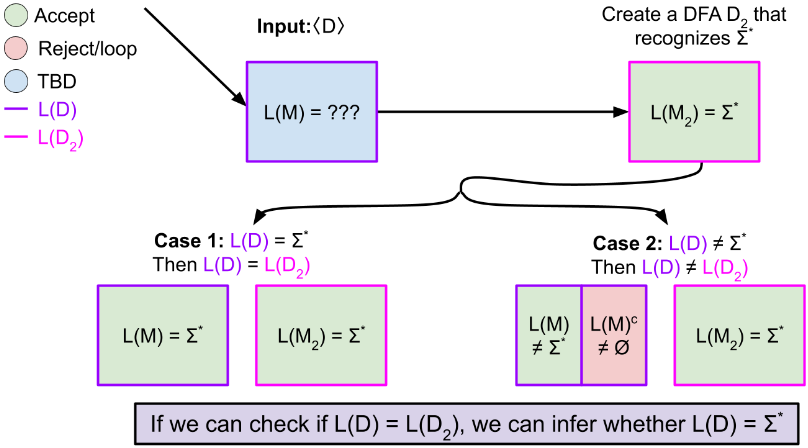
$$\mathrm{ALL}_{\mathrm{DFA}} = \{\langle D\rangle | D \text{ is a DFA}, L(D) = \Sigma^*\}$$

$$\mathrm{EQ}_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle | D_1, D_2 \text{ are DFAs, } L(D_1) = L(D_2)\}$$

- Assume we have a machine M_{EQ} that decides $\mathrm{EQ}_{\mathrm{DFA}}$.
- We need to construct a machine M_A that decides $\mathrm{ALL}_{\mathrm{DFA}}$

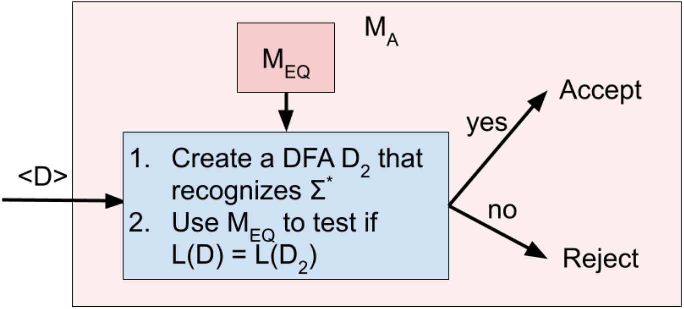
1. M_A takes $\langle D\rangle$ as input
2. M_A creates a DFA D_2 which recognizes Σ^*
3. M_A runs M_{EQ} on $\langle D, D_2\rangle$
   - 3.1 If M_{EQ} accepts $\langle D, D_2\rangle$ then M_A accepts $\langle D\rangle$
   - 3.2 If M_{EQ} rejects $\langle D, D_2\rangle$ then M_A rejects $\langle D\rangle$

$\text{ALL}_{\text{DFA}} \leq_T \text{EQ}_{\text{DFA}}$



$\mathrm{ALL}_{\mathrm{DFA}} \leq_T \mathrm{EQ}_{\mathrm{DFA}}$

$\mathrm{ALL}_{\mathrm{DFA}}$ = {<D> | D is a DFA, L(D) = Σ^*}
$\mathrm{EQ}_{\mathrm{DFA}}$ = {<D_1, D_2> | D_1, D_2 are DFAs, L(D_1) = L(D_2)}



If we can decide $\mathrm{EQ}_{\mathrm{DFA}}$, we can decide $\mathrm{ALL}_{\mathrm{DFA}}$

The Language $\mathrm{A_{TM}}$

Consider the following language

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

The Language $\mathrm{A_{TM}}$

Consider the following language

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's prove that $\mathrm{A_{TM}}$ is RE

The Language $\mathrm{A_{TM}}$

Consider the following language

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's prove that $\mathrm{A_{TM}}$ is RE
- We receive a machine M and an input w

The Language $\mathrm{A_{TM}}$

Consider the following language

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's prove that $\mathrm{A_{TM}}$ is RE
- We receive a machine M and an input w
- We want to accept $\langle M, w\rangle$ if M accepts w

The Language $\mathrm{A_{TM}}$

Consider the following language

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's prove that $\mathrm{A_{TM}}$ is RE
- We receive a machine M and an input w
- We want to accept $\langle M, w\rangle$ if M accepts w
- We want to loop on or (ideally) reject $\langle M, w\rangle$ if M rejects or loops on w

$\mathrm{A_{TM}}$ is RE

Let's prove that the following language is RE

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

$\mathrm{A_{TM}}$ is RE

Let's prove that the following language is RE

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

We'll design a machine M_A that recognizes $\mathrm{A_{TM}}$

$\mathrm{A_{TM}}$ is RE

Let's prove that the following language is RE

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

We'll design a machine M_A that recognizes $\mathrm{A_{TM}}$

1. M_A receives $\langle M, w\rangle$ as input

$\mathrm{A_{TM}}$ is RE

Let's prove that the following language is RE

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

We'll design a machine M_A that recognizes $\mathrm{A_{TM}}$

1. M_A receives $\langle M, w\rangle$ as input
2. M_A simulates M on w

A_{TM} is RE

Let's prove that the following language is RE

$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

We'll design a machine M_A that recognizes A_{TM}

1. M_A receives $\langle M, w\rangle$ as input
2. M_A simulates M on w
   2.1 If M ever accepts w, M_A accepts $\langle M, w\rangle$

A_{TM} is RE

Let's prove that the following language is RE

$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

We'll design a machine M_A that recognizes A_{TM}

1. M_A receives $\langle M, w\rangle$ as input
2. M_A simulates M on w
   2.1 If M ever accepts w, M_A accepts $\langle M, w\rangle$
   2.2 If M rejects or loops on w, M_A will not accept $\langle M, w\rangle$

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$\mathrm{A}_{\mathrm{TM}} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides A_{TM}

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides $\mathrm{A_{TM}}$

- This is just a hypothetical

$A_{TM} \leq_T$ HALT

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides A_{TM}

- This is just a hypothetical
- We are not saying M_H exists - we proved that it doesn't.

$A_{TM} \leq_T$ HALT

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides A_{TM}

- This is just a hypothetical
- We are not saying M_H exists - we proved that it doesn't.
- We are saying we could design M_A if we could use M_H as a subroutine

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides $\mathrm{A_{TM}}$

- In this hypothetical, M_H takes $\langle M, w \rangle$ and tells us if M halts on w

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides $\mathrm{A_{TM}}$

- In this hypothetical, M_H takes $\langle M, w \rangle$ and tells us if M halts on w
- We will design a machine M_A

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides $\mathrm{A_{TM}}$

- In this hypothetical, M_H takes $\langle M, w \rangle$ and tells us if M halts on w
- We will design a machine M_A
  - M_A takes $\langle M, w \rangle$

$A_{TM} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides A_{TM}

- In this hypothetical, M_H takes $\langle M, w \rangle$ and tells us if M halts on w
- We will design a machine M_A
  - M_A takes $\langle M, w \rangle$
  - We want it to tell us whether M accepts w

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

Let's prove that if we had a machine M_H that could decide HALT, we could construct a machine M_A that decides $\mathrm{A_{TM}}$

- In this hypothetical, M_H takes $\langle M, w \rangle$ and tells us if M halts on w
- We will design a machine M_A
  - M_A takes $\langle M, w \rangle$
  - We want it to tell us whether M accepts w

Let's design M_A, which will take advantage of M_H

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

To decide $\mathrm{A_{TM}}$ we do the following:

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

To decide $\mathrm{A_{TM}}$ we do the following:

1. Receive $\langle M, w\rangle$ as input

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w \rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w \rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

To decide $\mathrm{A_{TM}}$ we do the following:

1. Receive $\langle M, w \rangle$ as input
2. Check if M will halt on w. If it not, we reject

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

To decide $\mathrm{A_{TM}}$ we do the following:

1. Receive $\langle M, w\rangle$ as input
2. Check if M will halt on w. If it not, we reject
3. If M is guaranteed to halt, we run it on w

$A_{TM} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$A_{TM} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $A_{TM} \leq_T \mathrm{HALT}$

To decide A_{TM} we do the following:

1. Receive $\langle M, w\rangle$ as input
2. Check if M will halt on w. If it not, we reject
3. If M is guaranteed to halt, we run it on w
4. Accept $\langle M, w\rangle$ if and only if M accepts w

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:
  1. M_A takes $\langle M, w\rangle$ as input

$\mathrm{A}_{\mathrm{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A}_{\mathrm{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide A_{TM} as follows:
    1. M_A takes $\langle M, w\rangle$ as input
    2. M_A runs M_H on $\langle M, w\rangle$

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:
  1. M_A takes $\langle M, w\rangle$ as input
  2. M_A runs M_H on $\langle M, w\rangle$
     - "check whether it's safe to run M on w"

$A_{TM} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$A_{TM} = \{\langle M, w\rangle | w \in L(M)\}$$
Theorem: $A_{TM} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide A_{TM} as follows:
  1. M_A takes $\langle M, w\rangle$ as input
  2. M_A runs M_H on $\langle M, w\rangle$
  3. If M_H rejects $\langle M, w\rangle$ then M_A rejects $\langle M, w\rangle$

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:
  1. M_A takes $\langle M, w\rangle$ as input
  2. M_A runs M_H on $\langle M, w\rangle$
  3. If M_H rejects $\langle M, w\rangle$ then M_A rejects $\langle M, w\rangle$
     - "M loops on w so it clearly doesn't accept w

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:
  1. M_A takes $\langle M, w\rangle$ as input
  2. M_A runs M_H on $\langle M, w\rangle$
  3. If M_H rejects $\langle M, w\rangle$ then M_A rejects $\langle M, w\rangle$
  4. Otherwise M_A runs M on w until it halts

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:
  1. M_A takes $\langle M, w\rangle$ as input
  2. M_A runs M_H on $\langle M, w\rangle$
  3. If M_H rejects $\langle M, w\rangle$ then M_A rejects $\langle M, w\rangle$
  4. Otherwise M_A runs M on w until it halts
     - M_H told us that M was guaranteed to halt

$\mathrm{A}_{\mathrm{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w \rangle | M \text{ halts on } w\}$$
$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w \rangle | w \in L(M)\}$$

Theorem: $\mathrm{A}_{\mathrm{TM}} \leq_T \mathrm{HALT}$

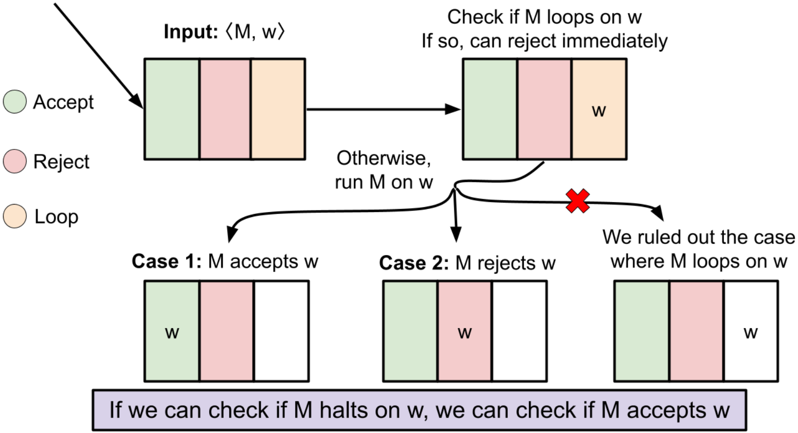
- Let M_H decide HALT
- M_A will decide A_{TM} as follows:
  1. M_A takes $\langle M, w \rangle$ as input
  2. M_A runs M_H on $\langle M, w \rangle$
  3. If M_H rejects $\langle M, w \rangle$ then M_A rejects $\langle M, w \rangle$
  4. Otherwise M_A runs M on w until it halts
     4.1 If M accepts w then M_A accepts $\langle M, w \rangle$

$\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
Theorem: $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$

- Let M_H decide HALT
- M_A will decide $\mathrm{A_{TM}}$ as follows:
  1. M_A takes $\langle M, w\rangle$ as input
  2. M_A runs M_H on $\langle M, w\rangle$
  3. If M_H rejects $\langle M, w\rangle$ then M_A rejects $\langle M, w\rangle$
  4. Otherwise M_A runs M on w until it halts
     - 4.1 If M accepts w then M_A accepts $\langle M, w\rangle$
     - 4.2 If M rejects w then M_A rejects $\langle M, w\rangle$

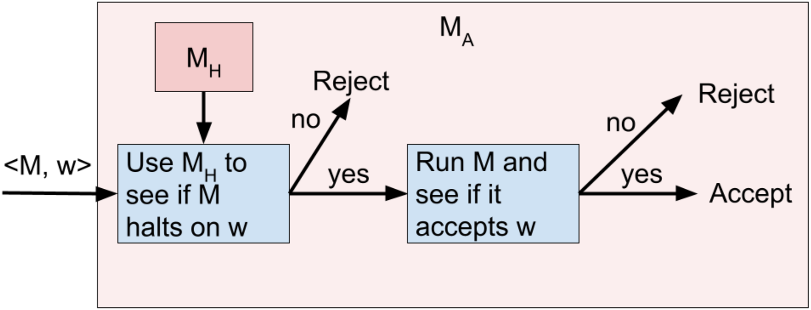
$A_{TM} \leq_T$ HALT



$A_{TM} \leq_T \text{HALT}$

$A_{TM} = \{\langle M, w\rangle \mid M \text{ accepts } w\}$

$\text{HALT} = \{\langle M, w\rangle \mid M \text{ halts on } w\}$



If we can decide HALT, we can decide A_{TM}

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's suppose have a machine M_A which decides $\mathrm{A_{TM}}$

$\text{HALT} \leq_T \text{A}_{\text{TM}}$

Let's prove that HALT is reducible to A_{TM}

$$\text{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\text{A}_{\text{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's suppose have a machine M_A which decides A_{TM}
- Let's design a machine M_H that could decide HALT if it could use M_A as a subroutine

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's suppose have a machine M_A which decides A_{TM}
- Let's design a machine M_H that could decide HALT <u>if it could use M_A as a subroutine</u>
  - We are <u>not</u> claiming HALT is decidable in general

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let's suppose have a machine M_A which decides A_{TM}
- Let's design a machine M_H that could decide HALT if it could use M_A as a subroutine
  - We are not claiming HALT is decidable in general
  - We are only showing it is decidable under this hypothetical scenario

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let M_A decide $\mathrm{A_{TM}}$

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let M_A decide $\mathrm{A_{TM}}$
- We'll design M_H, which receives $\langle M, w\rangle$ and wants to decide if M halts on w

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let M_A decide $\mathrm{A_{TM}}$
- We'll design M_H, which receives $\langle M, w\rangle$ and wants to decide if M halts on w
- We will create a new machine P at runtime

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let M_A decide A_{TM}
- We'll design M_H, which receives $\langle M, w\rangle$ and wants to decide if M halts on w
- We will create a new machine P at runtime
- P will accept w if and only if M halts on w

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

$$\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$$
$$\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$

- Let M_A decide A_{TM}
- We'll design M_H, which receives $\langle M, w\rangle$ and wants to decide if M halts on w
- We will create a new machine P at runtime
- P will accept w if and only if M halts on w
- If we can determine whether P accepts w, we can determine whether M halts on w

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

Suppose M_A decides A_{TM}. We'll design a machine M_H that decides HALT

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

Suppose M_A decides A_{TM}. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

Suppose M_A decides $\mathrm{A_{TM}}$. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

Suppose M_A decides A_{TM}. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
    2.1 P takes input s

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

Suppose M_A decides $\mathrm{A_{TM}}$. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
    - 2.1 P takes input s
    - 2.2 P simulates M on s
      Here, M is hard-coded

HALT $\leq_T$ $\mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

Suppose M_A decides $\mathrm{A_{TM}}$. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
   2.1 P takes input s
   2.2 P simulates M on s
   Here, M is hard-coded
   2.3 If M ever halts, P accepts s (even if M rejected)
   If M loops forever then so will P

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Let's prove that HALT is reducible to A_{TM}

Suppose M_A decides A_{TM}. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
   - 2.1 P takes input s
   - 2.2 P simulates M on s
     Here, M is hard-coded
   - 2.3 If M ever halts, P accepts s (even if M rejected)
     If M loops forever then so will P

When does P accept a string w?

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

Suppose M_A decides $\mathrm{A_{TM}}$. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
   2.1 P takes input s
   2.2 P simulates M on s
   Here, M is hard-coded
   2.3 If M ever halts, P accepts s (even if M rejected)
   If M loops forever then so will P

When does P accept a string w?

P accepts $w \Leftrightarrow M$ halts on w

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

Suppose M_A decides $\mathrm{A_{TM}}$. We'll design a machine M_H that decides HALT

1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
   2.1 P takes input s
   2.2 P simulates M on s
   Here, M is hard-coded
   2.3 If M ever halts, P accepts s (even if M rejected)
   If M loops forever then so will P
3. Run M_A on input $\langle P, w \rangle$

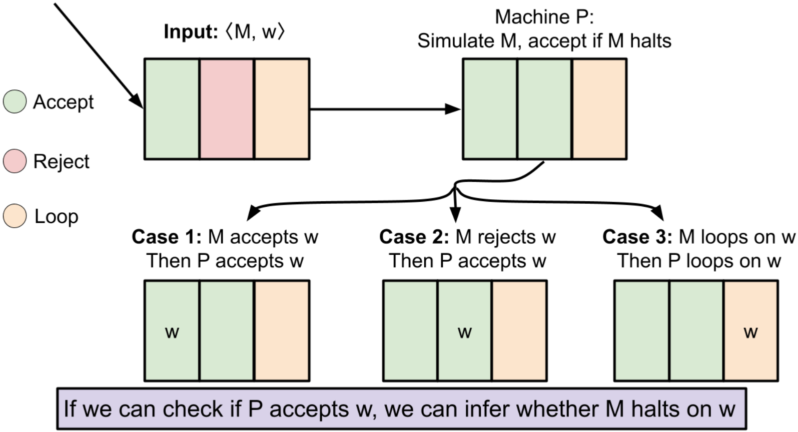
$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Let's prove that HALT is reducible to $\mathrm{A_{TM}}$

Suppose M_A decides $\mathrm{A_{TM}}$. We'll design a machine M_H that decides HALT

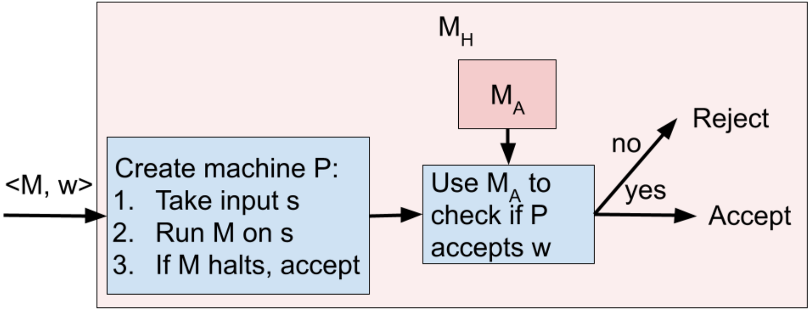
1. M_H receives $\langle M, w \rangle$ as input
2. Construct a machine P:
   2.1 P takes input s
   2.2 P simulates M on s
   Here, M is hard-coded
   2.3 If M ever halts, P accepts s (even if M rejected)
   If M loops forever then so will P
3. Run M_A on input $\langle P, w \rangle$
   3.1 If M_A accepts $\langle P, w \rangle$, M_H accepts $\langle M, w \rangle$
   If M_A rejects $\langle P, w \rangle$, M_H rejects $\langle M, w \rangle$

HALT $\leq_T$ A_{TM}



HALT $\leq_T$ A_{TM}

A_{TM} = {<M, w> | M accepts w}
HALT = {<M, w> | M halts on w}



If we can decide A_{TM}, we can decide HALT

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Corollary: $\mathrm{A_{TM}}$ is undecidable

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Corollary: A_{TM} is undecidable

Why?

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Corollary: A_{TM} is undecidable

Why?

- AFSOC A_{TM} is decidable

$\mathrm{HALT} \leq_T \mathrm{A}_{\mathrm{TM}}$

Corollary: A_{TM} is undecidable

Why?

- AFSOC A_{TM} is decidable
- Then HALT is decidable

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Corollary: $\mathrm{A_{TM}}$ is undecidable

Why?

- AFSOC $\mathrm{A_{TM}}$ is decidable
- Then HALT is decidable
  - But we know HALT is undecidable - this is a contradiction!

$\mathrm{HALT} \leq_T \mathrm{A_{TM}}$

Corollary: $\mathrm{A_{TM}}$ is undecidable

Why?

- AFSOC $\mathrm{A_{TM}}$ is decidable
- Then HALT is decidable
  - But we know HALT is undecidable - this is a contradiction!
- We conclude that $\mathrm{A_{TM}}$ couldn't have been decidable

Undecidability Proofs

We want to show that language B is undecidable

Undecidability Proofs

We want to show that language B is undecidable

Technique: Reduce from a known undecidable language A

Undecidability Proofs

We want to show that language B is undecidable

Technique: Reduce from a known undecidable language A

1. AFSOC B is decidable

Undecidability Proofs

We want to show that language B is undecidable

Technique: Reduce from a known undecidable language A

1. AFSOC B is decidable
2. Show that $A \leq_T B$
   "If we can decide B we can also decide A"

Undecidability Proofs

We want to show that language B is undecidable

Technique: Reduce from a known undecidable language A

1. AFSOC B is decidable
2. Show that $A \leq_T B$
   "If we can decide B we can also decide A"
3. But A is known to be undecidable

Undecidability Proofs

We want to show that language B is undecidable

Technique: Reduce from a known undecidable language A

1. AFSOC B is decidable
2. Show that $A \leq_T B$
   "If we can decide B we can also decide A"
3. But A is known to be undecidable
   - This is a contradiction!

Undecidability Proofs

We want to show that language B is undecidable

Technique: Reduce from a known undecidable language A

1. AFSOC B is decidable
2. Show that $A \leq_T B$
   "If we can decide B we can also decide A"
3. But A is known to be undecidable
   - This is a contradiction!
4. We conclude that B was never decidable in the first place