Barber paradox.

Created by logician Bertrand Russell.
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven.
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven. Barber announces:
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven. Barber announces:

“I shave all and only those men who do not shave themselves.”
Barber paradox.

Created by logician Bertrand Russell.
Village with just 1 barber (a man), all men clean-shaven. Barber announces:
“I shave all and only those men who do not shave themselves.”

Who shaves the barber?
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven. Barber announces:
“I shave all and only those men who do not shave themselves.”

Who shaves the barber?

Case 1: It’s the barber.
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven.
Barber announces:
“I shave all and only those men who do not shave themselves.”

Who shaves the barber?

Case 1: It’s the barber.
Case 2: Somebody else.
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven. Barber announces:
“I shave all and only those men who do not shave themselves.”

Who shaves the barber?

Case 1: It’s the barber.
Case 2: Somebody else.

Cannot answer that question in either case!
Barber paradox.

Created by logician Bertrand Russell.

Village with just 1 barber (a man), all men clean-shaven.
Barber announces:
“I shave all and only those men who do not shave themselves.”

Who shaves the barber?

Case 1: It’s the barber.
Case 2: Somebody else.

Cannot answer that question in either case! Paradox!!!
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[ \exists y \ \forall x \ (x \in y \iff P(x)) \tag{1} \]
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[ \exists y \ \forall x \ (x \in y \iff P(x)) \]  \hspace{1cm} (1)

\( y \) is the set of elements that satisfies the proposition \( P(x) \).
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[ \exists y \ \forall x \ (x \in y \iff P(x)) \]  \hspace{1cm} (1)

\( y \) is the set of elements that satisfies the proposition \( P(x) \).

\( P(x) = x \notin x. \)
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[
\exists y \ \forall x \ (x \in y \iff \ P(x))
\]  \hspace{1cm} (1)

\(y\) is the set of elements that satisfies the proposition \(P(x)\).

\(P(x) = x \notin x\).

There exists a \(y\) that satisfies statement 1 for \(P(\cdot)\).
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x (x \in y \iff P(x)) \tag{1} \]

\( y \) is the set of elements that satisfies the proposition \( P(x) \).

\( P(x) = x \notin x. \)

There exists a \( y \) that satisfies statement 1 for \( P(\cdot) \).

Take \( x = y. \)
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[ \exists y \quad \forall x \quad (x \in y \iff P(x)) \quad (1) \]

\( y \) is the set of elements that satisfies the proposition \( P(x) \).

\( P(x) = x \notin x \).

There exists a \( y \) that satisfies statement 1 for \( P(\cdot) \).

Take \( x = y \).

\[ y \in y \iff y \notin y. \]
Russell’s Paradox: Assuming Existence of Set of All Sets

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x \ (x \in y \iff P(x)) \quad (1) \]

\( y \) is the set of elements that satisfies the proposition \( P(x) \).

\( P(x) = x \not\in x. \)

There exists a \( y \) that satisfies statement 1 for \( P(\cdot) \).

Take \( x = y \).

\[ y \in y \iff y \not\in y. \]

Contradiction!
Is this stuff actually useful?
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!

(output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

HALT

\[(P, I)\]

\(P\) - program

\(I\) - input.

Determines if \(P(I)\) (run on \(I\)) halts or loops forever.

Notice:

Need a computer...with the notion of a stored program!!!!

(not an adding machine!

not a person and an adding machine.)

Program is a text string.

Text string can be an input to a program.

Program can be an input to a program.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
(output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$
Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$
   $P$ - program
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
(output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\[ HALT(P, I) \]
- \( P \) - program
- \( I \) - input.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\[ \text{HALT}(P, I) \]

- \( P \) - program
- \( I \) - input.

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!
Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$
   $P$ - program
   $I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.

Notice:
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$

$P$ - program

$I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.

Notice:

Need a computer
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\( \text{HALT}(P, I) \)

\( P \) - program
\( I \) - input.

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$

$P$ - program
$I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine!)
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
  (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$
  $P$ - program
  $I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)
Is this stuff actually useful?

Problem 1: Verify that my program is correct!
Problem 2: Check that the compiler works correctly!
(output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\(\text{HALT}(P, I)\)
- \(P\) - program
- \(I\) - input.

Determines if \(P(I)\) (\(P\) run on \(I\)) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)
Is this stuff actually useful?

Problem 1: Verify that my program is correct!
Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\[ \text{HALT}(P, I) \]
   \( P \) - program
   \( I \) - input.

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!
Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)
How about.. Check that the compiler terminates on a certain input.

$$HALT(P, I)$$
   $$P$$ - program
   $$I$$ - input.

Determines if $$P(I)$$ ($$P$$ run on $$I$$) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Text string can be an input to a program.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!
Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\[ \text{HALT}(P, I) \]
   \[ P \text{ - program} \]
   \[ I \text{ - input.} \]

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Text string can be an input to a program.
Program can be an input to a program.
Is this stuff actually useful?

Problem 1: Verify that my program is correct!

Problem 2: Check that the compiler works correctly!
   (output program is equivalent to its input program)

How about.. Check that the compiler terminates on a certain input.

\[ \text{HALT}(P, I) \]
\[ P - \text{program} \]
\[ I - \text{input.} \]

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Text string can be an input to a program.
Program can be an input to a program.
Implementing HALT.

HALT (P, I)

- P - program
- I - input.

Determines if P(I) (P run on I) halts or loops forever.

Run P on I and check!

How long do you wait?
Implementing HALT.

$HALT(P, I)$

$P$ - program

$I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.
Implementing HALT.

\[ \text{HALT}(P, I) \]
\[
P \text{ - program} \\
I \text{ - input.}
\]

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

Run \( P \) on \( I \) and check!
Implementing HALT.

$HALT(P, I)$

$P$ - program

$I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.

Run $P$ on $I$ and check!

How long do you wait?
Halt does not exist.
Halt does not exist.

\[
HALT(P, I)
\]

- \(P\) - program
- \(I\) - input.

Determines if \(P(I)\) (\(P\) run on \(I\)) halts or loops forever.
Halt does not exist.

\( \text{HALT}(P, I) \)
- \( P \) - program
- \( I \) - input.

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

**Theorem:** There is no program HALT.
Halt does not exist.

\[ \text{HALT}(P, I) \]
\[ P \text{ - program} \]
\[ I \text{ - input.} \]

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

**Theorem:** There is no program \( \text{HALT} \).

**Proof Idea:** Proof by contradiction, use self-reference.
Halt and Turing.

Proof:

Assume there is a program \( \text{HALT}(\cdot, \cdot) \). Turing(P):

1. If \( \text{HALT}(P, P) = \text{halts} \), then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program \( \text{HALT} \).

There is text that "is" the program \( \text{Turing} \).

Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts

\[
\Rightarrow \text{HALT}(\text{Turing}, \text{Turing}) = \text{halts}
\]

\[
\Rightarrow \text{Turing}(\text{Turing}) \text{ loops forever.}
\]

Case 2: Turing(Turing) loops forever

\[
\Rightarrow \text{HALT}(\text{Turing}, \text{Turing}) \neq \text{halts}
\]

\[
\Rightarrow \text{Turing}(\text{Turing}) \text{ halts.}
\]

Contradiction.

Program HALT does not exist!
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot,\cdot)$. 
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot, \cdot)$.

Turing($P$)
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot,\cdot)$.

1. **Turing(P)**
2. If $HALT(P,P) =$“halts”, then go into an infinite loop.
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot,\cdot)$.

Turing($P$)

1. If $HALT(P,P)$ = “halts”, then go into an infinite loop.
2. Otherwise, halt immediately.
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot, \cdot)$.

$\text{Turing}(P)$
1. If $HALT(P,P) =$“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$. 

Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot,\cdot)$.

**Turing(P)**

1. If $HALT(P,P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.

There is text that “is” the program $HALT$.
Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing($P$)
1. If $HALT(P, P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.
There is text that “is” the program $HALT$.
There is text that is the program Turing.
Halt and Turing.

**Proof:** Assume there is a program \( HALT(\cdot,\cdot) \).

\textbf{Turing}(P) \\
1. If \( HALT(P,P) = \text{"halts"} \), then go into an infinite loop. \\
2. Otherwise, halt immediately.

Assumption: there is a program \( HALT \). \\
There is text that “is” the program \( HALT \). \\
There is text that is the program \( Turing \). \\
Can run \( Turing \) on \( Turing \)!
Proof: Assume there is a program $HALT(·,·)$. 

$Turing(P)$
1. If $HALT(P,P) =$“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.
There is text that “is” the program $HALT$.
There is text that is the program $Turing$.
Can run $Turing$ on $Turing$!

Does $Turing(Turing)$ halt?
Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing($P$)
1. If $HALT(P,P) =$"halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that “is” the program HALT.
There is text that is the program Turing.
Can run Turing on Turing!

Does $Turing(Turing)$ halt?

Case 1: Turing(Turing) halts
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot, \cdot)$.

$Turing(P)$

1. If $HALT(P, P) =$ "halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.

There is text that “is” the program $HALT$.

There is text that is the program $Turing$.

Can run $Turing$ on $Turing$!

Does $Turing(Turing)$ halt?

Case 1: $Turing(Turing)$ halts

$\implies$ then $HALT(Turing, Turing) = \text{halts}$
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot,\cdot)$.

$Turing(P)$
1. If $HALT(P,P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.
There is text that “is” the program $HALT$.
There is text that is the program $Turing$.
Can run $Turing$ on $Turing$!

Does $Turing(Turing)$ halt?

Case 1: $Turing(Turing)$ halts
   $\implies$ then $HALT(Turing, Turing) = \text{halts}$
   $\implies$ $Turing(Turing)$ loops forever.
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot, \cdot)$.

**Turing(P)**
1. If $HALT(P, P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that “is” the program HALT.
There is text that is the program Turing.
Can run Turing on Turing!

Does $Turing(Turing)$ halt?

Case 1: Turing(Turing) halts
   $\implies$ then $HALT(Turing, Turing) = \text{halts}$
   $\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
Halt and Turing.

**Proof:** Assume there is a program *HALT*(·, ·).

*Turing*(P)
1. If *HALT*(P, P) = “halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program *HALT*. There is text that “is” the program *HALT*. There is text that is the program *Turing*. Can run *Turing* on *Turing*!

Does *Turing*(Turing) halt?

Case 1: *Turing*(Turing) halts

⇒ then *HALT*(Turing, Turing) = halts

⇒ *Turing*(Turing) loops forever.

Case 2: *Turing*(Turing) loops forever

⇒ then *HALT*(Turing, Turing) ≠ halts

Program HALT does not exist!
**Halt and Turing.**

**Proof:** Assume there is a program $HALT(\cdot, \cdot)$.

**Turing**($P$)
1. If $HALT(P, P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.  
There is text that “is” the program HALT.  
There is text that is the program Turing.  
Can run Turing on Turing!

Does $\text{Turing}(\text{Turing})$ halt?

Case 1: $\text{Turing}(\text{Turing})$ halts  
$
\implies \text{then } HALT(\text{Turing}, \text{Turing}) = \text{halts}  
\implies \text{Turing}(\text{Turing})$ loops forever.

Case 2: $\text{Turing}(\text{Turing})$ loops forever  
$
\implies \text{then } HALT(\text{Turing}, \text{Turing}) \neq \text{halts}  
\implies \text{Turing}(\text{Turing})$ halts.

Contradiction.  
Program HALT does not exist!
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot,\cdot)$.

Turing($P$)
1. If $HALT(P,P) =$“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT. There is text that “is” the program HALT. There is text that is the program Turing. Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
   \[\Rightarrow \text{ then } HALT(Turing, Turing) = \text{halts}\]
   \[\Rightarrow \text{Turing(Turing) loops forever.}\]

Case 2: Turing(Turing) loops forever
   \[\Rightarrow \text{ then } HALT(Turing, Turing) \neq \text{halts}\]
   \[\Rightarrow \text{Turing(Turing) halts.}\]

Contradiction.
Halt and Turing.

**Proof:** Assume there is a program $HALT(·, ·)$.

**Turing(P)**
1. If $HALT(P, P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.
There is text that “is” the program $HALT$.
There is text that is the program $Turing$.
Can run $Turing$ on $Turing$!

Does $Turing(Turing)$ halt?

Case 1: $Turing(Turing)$ halts
   $\implies$ then $HALT(Turing, Turing) = \text{halts}$
   $\implies$ $Turing(Turing)$ loops forever.

Case 2: $Turing(Turing)$ loops forever
   $\implies$ then $HALT(Turing, Turing) \neq \text{halts}$
   $\implies$ $Turing(Turing)$ halts.

Contradiction. Program $HALT$ does not exist!
Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If $HALT(P,P) =$ “halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT. There is text that “is” the program HALT. There is text that is the program Turing. Can run Turing on Turing!

Does $Turing(Turing)$ halt?

Case 1: Turing(Turing) halts
   $\implies$ then $HALT(Turing, Turing) = \text{halts}$
   $\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
   $\implies$ then $HALT(Turing, Turing) \neq \text{halts}$
   $\implies$ Turing(Turing) halts.

Contradiction. Program HALT does not exist!
Another view of proof: diagonalization.

Any program is a fixed length string.
Another view of proof: diagonalization.

Any program is a fixed length string. Fixed length strings are enumerable.
Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.
Another view of proof: diagonalization.

Any program is a fixed length string. Fixed length strings are enumerable. Program halts or not any input, which is a string.

\begin{array}{|c|c|c|c|}
\hline
 & P_1 & P_2 & P_3 & \ldots \\
\hline
P_1 & H & H & L & \ldots \\
P_2 & L & L & H & \ldots \\
P_3 & L & H & H & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots \\
\hline
\end{array}

Halt(P, P) - diagonal. Turing - is not Halt. and is different from every P_i on the diagonal. Turing is not on list. ⇒ Turing is not a program. But Turing can be constructed as a program if the program Halt exists. Halt does not exist!
Another view of proof: diagonalization.

Any program is a fixed length string. 
Fixed length strings are enumerable. 
Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
</tr>
</tbody>
</table>

Halt($P,P$) - diagonal.
Another view of proof: diagonalization.

Any program is a fixed length string.  
Fixed length strings are enumerable.  
Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>...</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Halt(\( P, P \)) - diagonal.
Turing - is not Halt.
Another view of proof: diagonalization.

Any program is a fixed length string.  
Fixed length strings are enumerable.  
Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>...</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Halt($P,P$) - diagonal.  
Turing - is not Halt.  
and is different from every $P_i$ on the diagonal.
Another view of proof: diagonalization.

Any program is a fixed length string. Fixed length strings are enumerable. Program halts or not any input, which is a string.

\[
\begin{array}{c|cccc}
 & P_1 & P_2 & P_3 & \ldots \\
\hline
P_1 & H & H & L & \ldots \\
P_2 & L & L & H & \ldots \\
P_3 & L & H & H & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots \\
\end{array}
\]

Halt(P,P) - diagonal. Turing - is not Halt. and is different from every \( P_i \) on the diagonal. Turing is not on list.
Another view of proof: diagonalization.

Any program is a fixed length string. Fixed length strings are enumerable. Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>...</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Halt($P, P$) - diagonal.
Turing - is not Halt.
Turing is not on list. $\implies$ Turing is not a program.
Another view of proof: diagonalization.

Any program is a fixed length string. 
Fixed length strings are enumerable. 
Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>...</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Halt($P,P$) - diagonal. 
Turing - is not Halt. 
and is different from every $P_i$ on the diagonal. 
Turing is not on list. $\implies$ Turing is not a program. 
But Turing can be constructed as a program if the program Halt exists.
Another view of proof: diagonalization.

Any program is a fixed length string.  
Fixed length strings are enumerable.  
Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
</tr>
</tbody>
</table>

Halt($P,P$) - diagonal.  
Turing - is not Halt.  
and is different from every $P_i$ on the diagonal.  
Turing is not on list.  $\implies$ Turing is not a program.  
But Turing can be constructed as a program if the program Halt exists.  
Halt does not exist!
Another view of proof: diagonalization.

Any program is a fixed length string. Fixed length strings are enumerable. Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>...</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

Halt($P, P$) - diagonal.
Turing - is not Halt.
and is different from every $P_i$ on the diagonal.
Turing is not on list. $\Rightarrow$ Turing is not a program.
But Turing can be constructed as a program if the program Halt exists.
Halt does not exist!
Turing machine.
A Turing machine.
– an (infinite) tape with characters
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ...
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!
Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!

Now that’s a computer! (not far from today’s computers)
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is...
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... a programming language!!! Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... a programming language!!! Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

   Any formal system either is inconsistent or incomplete.
   Inconsistent: A false sentence can be proven.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.
Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: “built” computers out of arithmetic.
Church proved an equivalent theorem. (Previously.)

Used \( \lambda \) calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

   Any formal system either is inconsistent or incomplete.
   Inconsistent: A false sentence can be proven.
   Incomplete: There is no proof for some sentence in the system.

Along the way: “built” computers out of arithmetic.
   Showed that every mathematical statement corresponds to an
Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: “built” computers out of arithmetic.
Showed that every mathematical statement corresponds to an ....natural number!!!!
Summary: computability.

Computer Programs are interesting objects.
Mathematical objects.
Formal Systems.

Computer Programs cannot completely "understand" computer programs.
Example: no computer program can tell if any other computer program HALTS.
Proof Idea: Diagonalization.
Program: Turing (or DIAGONAL) takes $P$.
Assume there is HALT.
DIAGONAL flips answer.
Loops if $P$ halts, halts if $P$ loops.
What does Turing do on Turing? Doesn't loop or HALT.
HALT does not exist!
More on this topic in CS 172.

Computation is a lens for other action in the world.
Summary: computability.

Computer Programs are interesting objects. Mathematical objects. Formal Systems.

Computer Programs cannot completely “understand” computer programs.
Summary: computability.

Computer Programs are interesting objects.
Mathematical objects.
Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Example: no computer program can tell if any other computer program HALTS.
Summary: computability.

Computer Programs are interesting objects.
Mathematical objects.
Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Example: no computer program can tell if any other computer program HALTS.

Proof Idea: Diagonalization.
Program: Turing (or DIAGONAL) takes $P$.
Assume there is HALT.
DIAGONAL flips answer.
Loops if $P$ halts, halts if $P$ loops.
What does Turing do on turing? Doesn’t loop or HALT.
HALT does not exist!
Summary: computability.

Computer Programs are interesting objects.
  Mathematical objects.
  Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Example: no computer program can tell if any other computer program HALTS.

Proof Idea: Diagonalization.
  Program: Turing (or DIAGONAL) takes $P$.
  Assume there is HALT.
  DIAGONAL flips answer.
  Loops if P halts, halts if P loops.
  What does Turing do on turing? Doesn’t loop or HALT.
  HALT does not exist!

More on this topic in CS 172.
Summary: computability.

Computer Programs are interesting objects.
   Mathematical objects.
   Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Example: no computer program can tell if any other computer program HALTS.

Proof Idea: Diagonalization.
   Program: Turing (or DIAGONAL) takes $P$.
   Assume there is HALT.
      DIAGONAL flips answer.
      Loops if $P$ halts, halts if $P$ loops.
   What does Turing do on Turing? Doesn’t loop or HALT.
      HALT does not exist!

More on this topic in CS 172.

Computation is a lens for other action in the world.