Barber paradox.

Created by logician Bertrand Russell.
Village with just 1 barber, 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.

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 \(P(x)\).
\(P(x) = x \neq x.\)
There exists a \(y\) that satisfies statement 1 for \(P(\cdot)\).
Take \(x = y.\)
\(y \in y \iff y \notin y.\)
Oops!

Is this stuff actually useful?
Verify that my program is correct!
Check that the compiler works!
How about... Check that the compiler terminates on a certain input.

Implementing HALT.

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

- \(P\) - 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.

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

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

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


Halt and Turing.

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

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

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

\(\text{Turing}(P)\)

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 \(\text{HALT}\).
There is text that is the program \(\text{Turing}\).
Can run \(\text{Turing}\) on \(\text{Turing}\)!

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

- \(\text{Turing}(\text{Turing})\) halts
- \(\text{HALT}(\text{Turing}, \text{Turing}) = \text{halts}\)
- \(\text{Turing}(\text{Turing})\) loops forever
- \(\text{HALT}(\text{Turing}, \text{Turing}) \neq \text{halts}\)
- \(\text{Turing}(\text{Turing})\) halts.

Contradiction. Program \(\text{HALT}\) 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.

\[
P_1 \ H \ H \ L \ \ldots
P_2 \ L \ L \ H \ \ldots
P_3 \ L \ \ H \ \ H \ \ldots
\vdots
\vdots
\vdots
\]

\(\text{Halt}(P)\) - 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. Turing can be constructed from Halt. Halt does not exist!

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. 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.