The Halting Problem

No, we can't build a machine to solve all of our problems.

Background

Some History

1900 - David Hilbert poses his "23 questions" including the decision problem

1928 - David Hilbert poses *Entscheidungsproblem* for Mathematics

1930 - Kurt Gödel proves Mathematics is incomplete

1936 - Alonzo Church proves *Entscheidungsproblem* unsolvable (lambda calc)

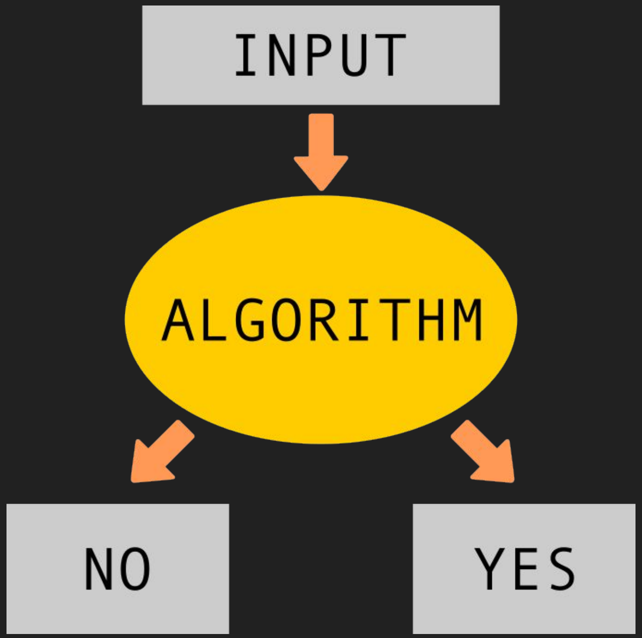
1937 - Alan Turing proves *Entscheidungsproblem* unsolvable (Turing machines)

1948 - Stephen Kleene explores the idea that the problem is about termination

1952 - Martin Davis coins the term "halting problem"

The Decision Problem

The Decision Problem



A Decision Problem is any computational problem that can be phrased as a yes/no question where the answer is based on a set of initial values or inputs.

Examples:

- Is *x* prime?
- Given *x*, *y* does *x* divide *y* evenly?
- Given a username *u* and password *p*, does *p* authenticate *u*?

So what is decidable?

And what is undecidable?

Some Possibilities

Zeroth order logic

First order logic

Higher order logic

Chess

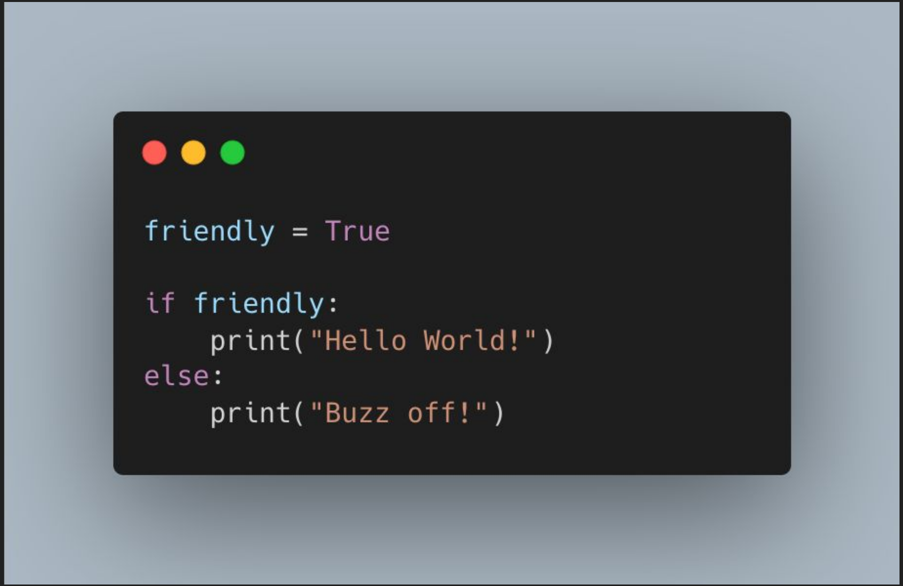
Minesweeper

Social interaction

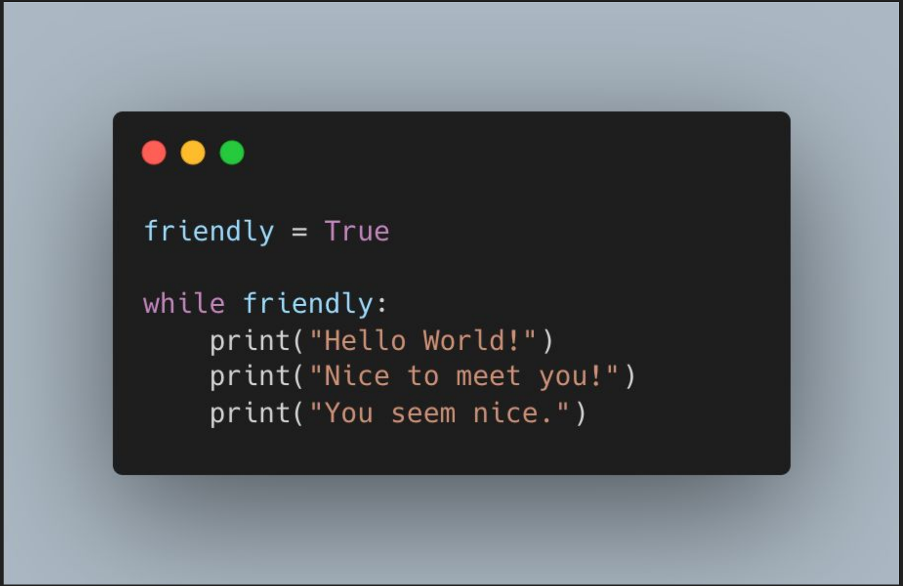
Matrix multiplication decomposition

Regex

Halting



```
friendly = True

if friendly:
    print("Hello World!")
else:
    print("Buzz off!")
```

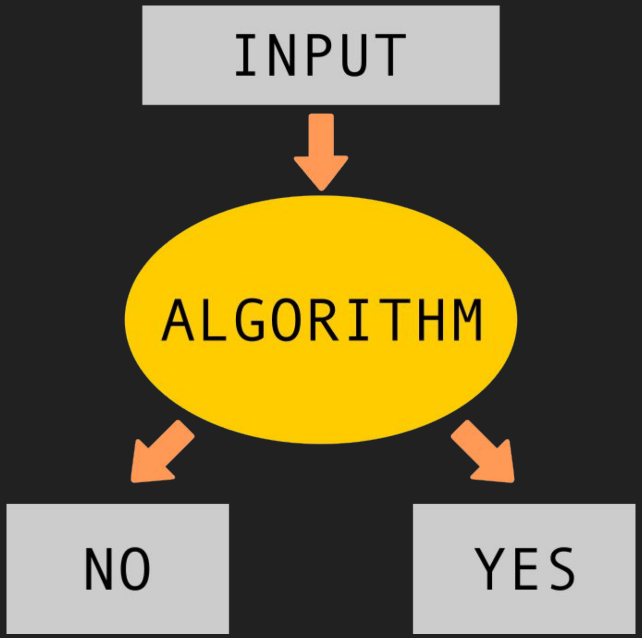


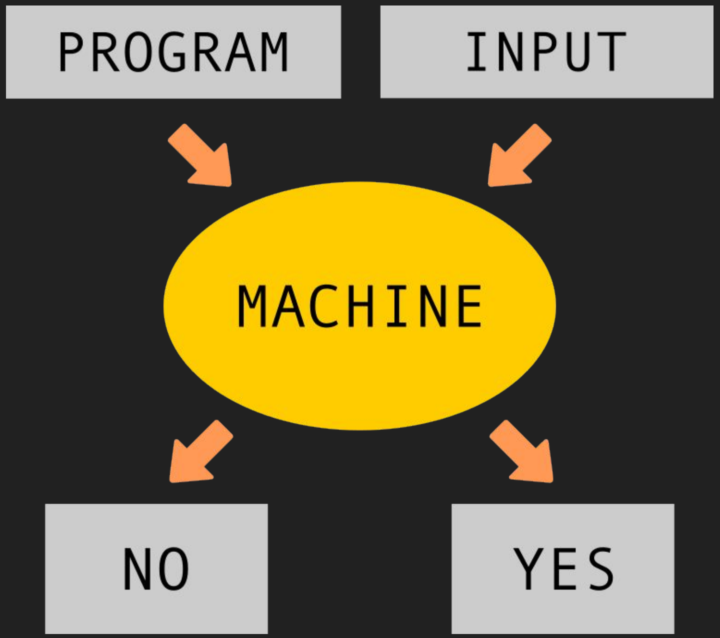
```
friendly = True

while friendly:
    print("Hello World!")
    print("Nice to meet you!")
    print("You seem nice.")
```

This program **halts**

This program **does not halt**

The Halting Problem



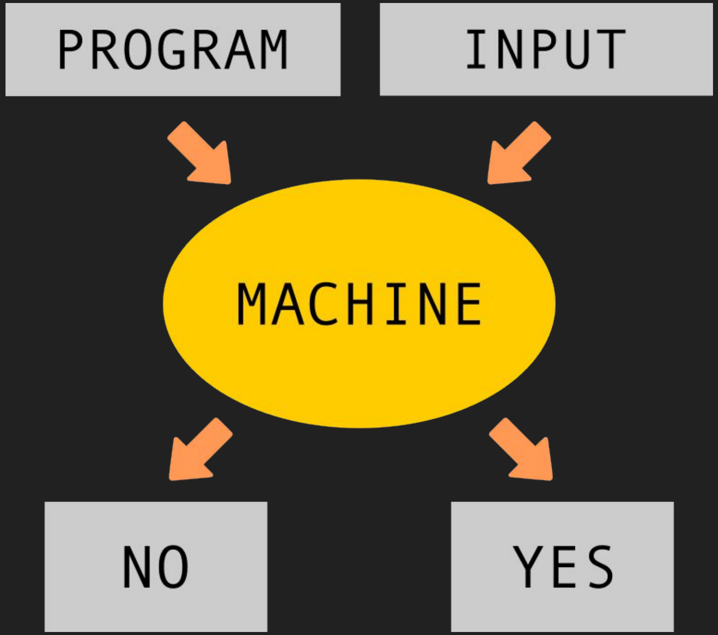


The Halting Problem

A specific decision problem

If a program **`p`** is given input **`i`** will the program halt or will it loop forever?

Essentially, this is an infinite loop checker for specific inputs.

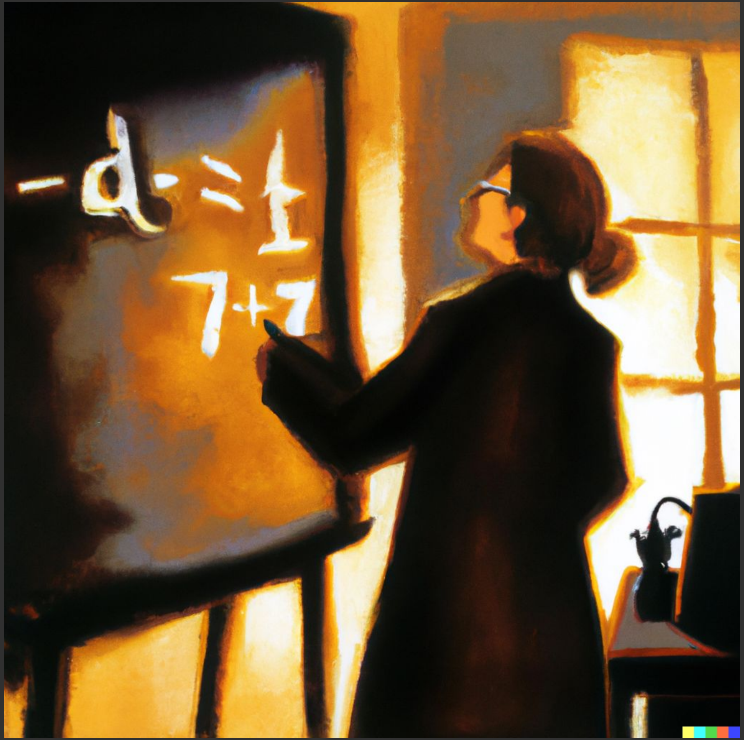


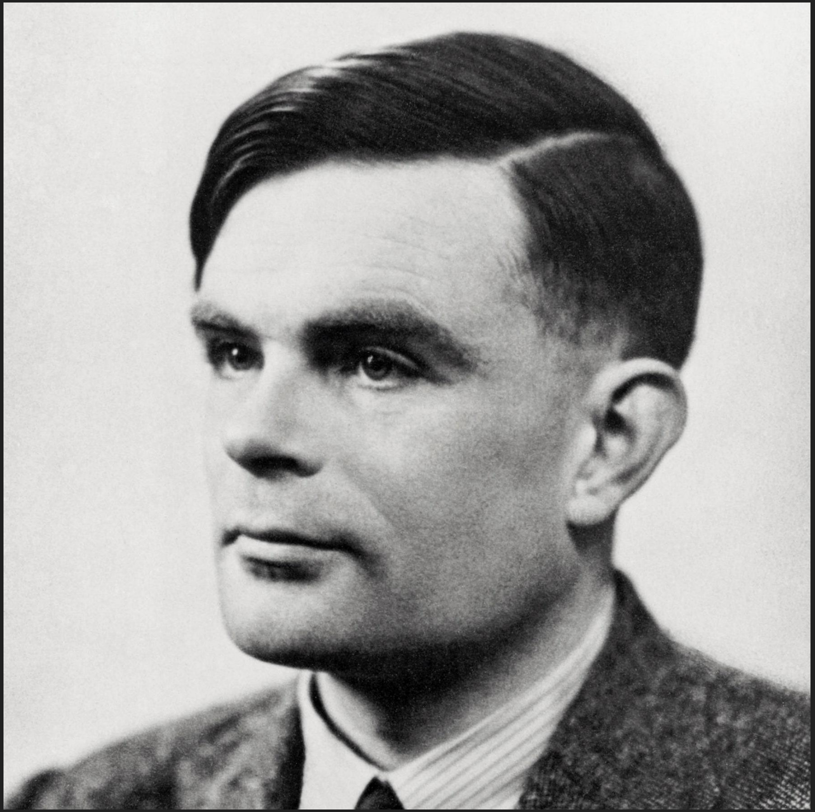
So can we solve the halting problem?

No.

😔

Proof





Alan Turing (1912-1954)

A. M. Turing

Proof by Contradiction

When given an absolute statement (e.g. "all problems can be solved"), the easiest way to disprove it is often a counterexample, and the easiest way to prove it is often by contradiction.

Usually, the inverse of an absolute statement can be phrased as an existence statement and vice versa.

Proof by contradiction is where you assume the inverse of the premise, and then show that a logical contradiction exists.

EXAMPLE:

"No positive number has a square root of zero."

There exists a positive number x with a square root of zero.

Then $\sqrt{x} = 0$ and $x > 0$

Then $(\sqrt{x})^2 = 0^2 = 0$

So $x = 0$ which is a contradiction because $x > 0$

So it follows that the premise "No positive number has a square root of zero" is true

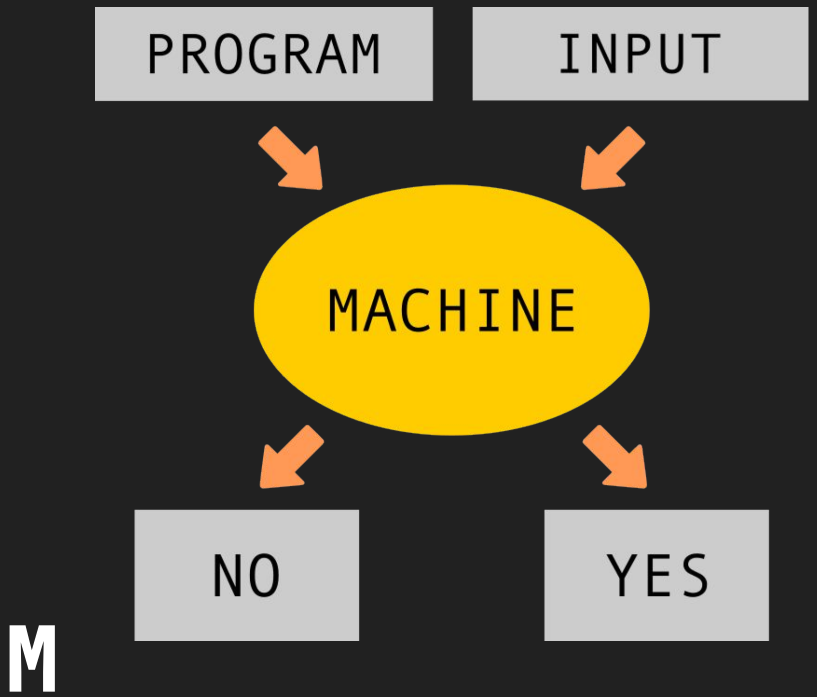
STATEMENT:

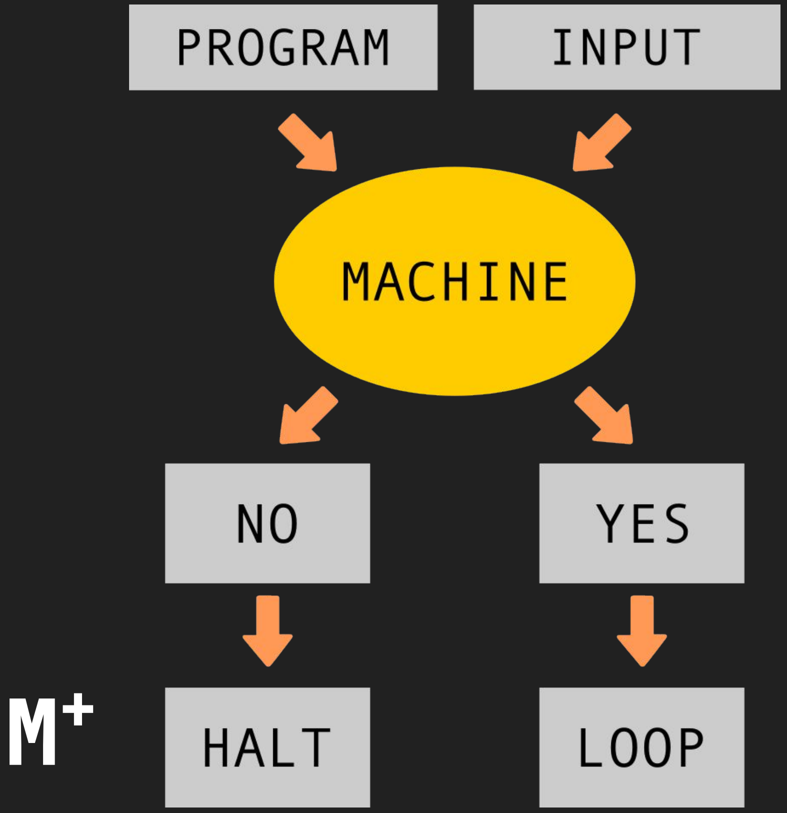
"There is no possible machine that solves the halting problem."

INVERSE:

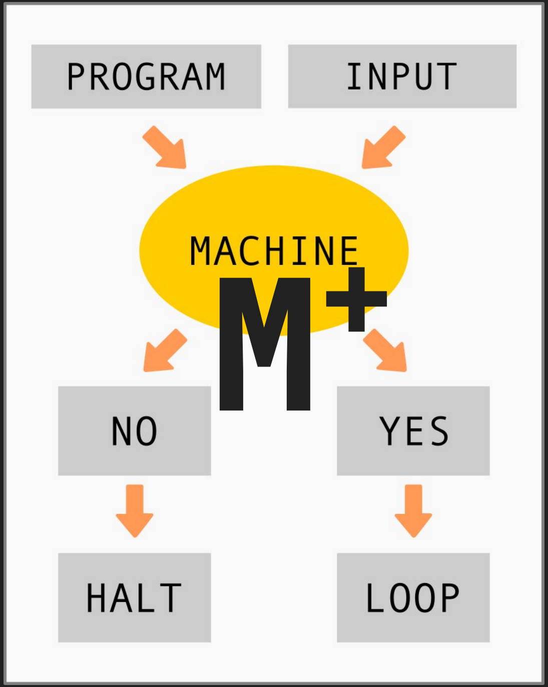
"There exists a machine that solves the halting problem."

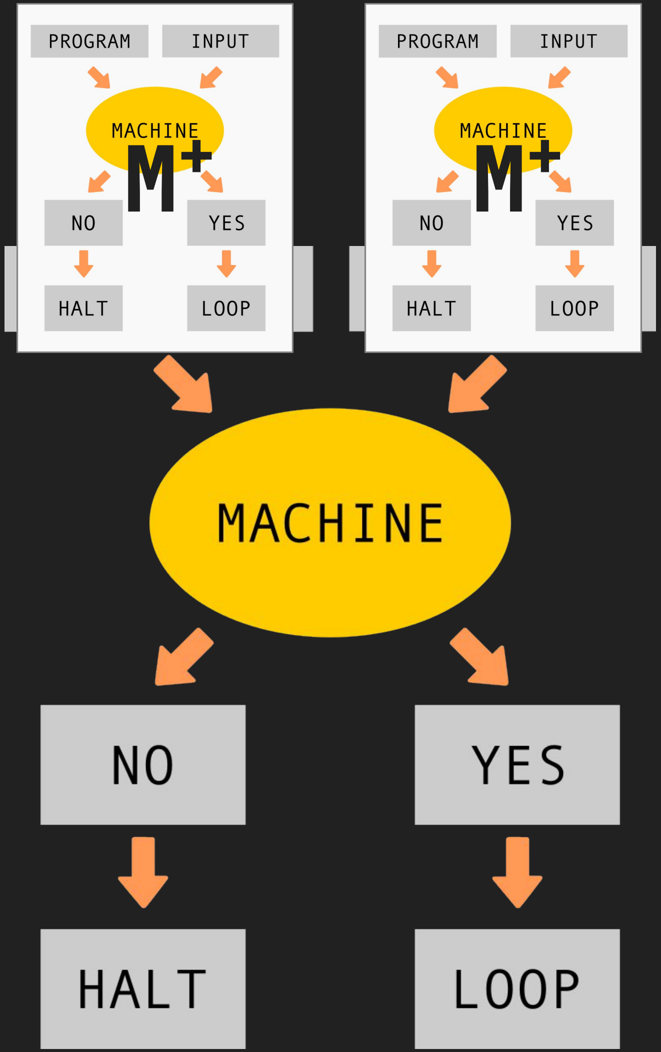
Suppose a Such a Machine Exists…





Feed **M**$^+$ Into Itself…





Feed **M**$^+$ Into Itself…

Now **M** is trying to specifically answer:

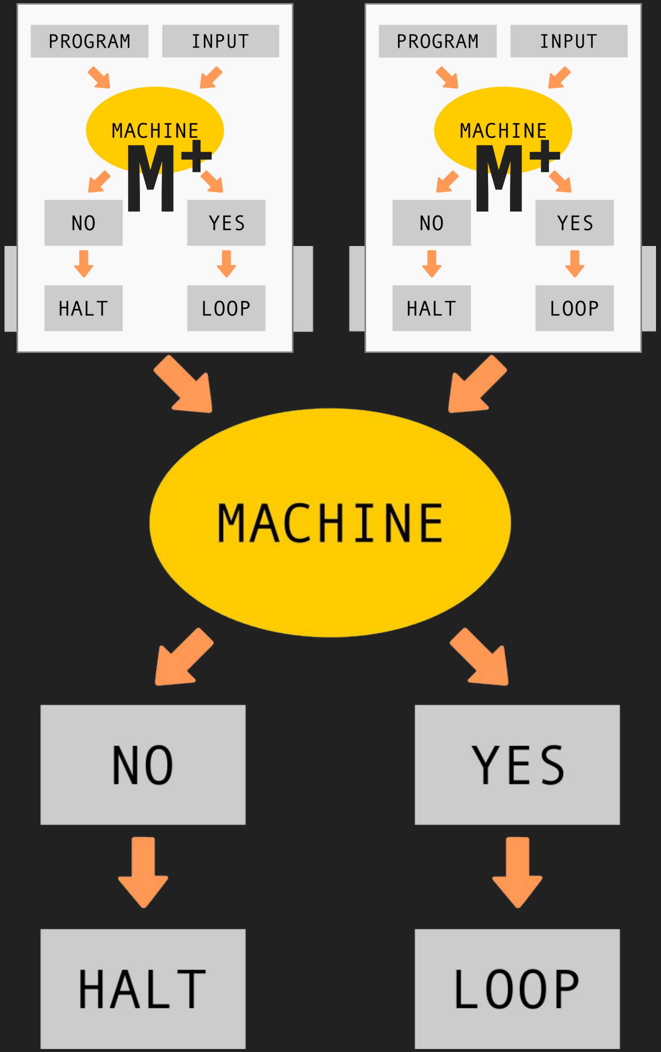
"Would **M**$^+$ halt when given input **M**$^+$?"

If it would halts, then our **M**$^+$ loops forever and does not halt

If it would not halt, then our **M**$^+$ does halt

So if it halts, it does not halt, and if it does not halt it halts 🤔

This is our contradiction 🥳



So there is no possible machine that solves the halting problem.

Consequences

Cannot Verify Programs Will Terminate

For programs running on Turing machines (i.e. using a Turing-complete language), it's not possible to run a program that will verify that the routine will terminate.

Hard real-time computing often requires guarantees that subroutines will finish and finish before a deadline. So you could use:

1. Restricted style of Turing-complete languages
   - MISRA C (C), SPARK (Ada)
2. Languages that are *near* Turing-complete
   - Coq, BNF, Charity, Regex, BlooP (Bounded Loop), SQL92

Other times, heuristics are used to approximate a solution.

Possibly Decidable Systems/Problems

- ✅ Zeroth order logic
- ❌ First order logic
- ❌ Higher order logic
- ✅ Chess
- ❌ Minesweeper
- ⁉️ Social interaction
- ❌ Matrix multiplication decomposition
- ✅ Regex

Turing's Proof with Computation Machines

Turing's proof introduces the notion of computation by machine rather than using general recursive functions (lambda-definable functions).

The proof extends to any model of computation that is equivalent to Turing machines:

- Markov algorithm
- Lambda calculus
- Post systems
- register machines
- tag systems

This proof was a step in a series of papers and discoveries that led to modern computing

More Abstract Consequences

What does it mean that we cannot write code that check every flaw in other code that we write?

From the beginning of computing, we knew that we would be able to create things that were in some sense unknowable or unverifyable. Is that dangerous or concerning?

What does it mean that certain logical problems are unsolvable/undecidable, and what does that imply for the human spirit of ingenuity?

This presentation does halt.