Hilbert's 10th Problem, Undecidability, & The Halting Problem

A mini-lecture series

CSE498 Collaborative Design (W) - Secure and Efficient C++ Software Development

03/24/2025

Kira Chan

https://cse.msu.edu/~chanken1/

David Hilbert

- Brilliant mathematician in the early 1900s
- During his speech to the International Congress of Mathematicians in 1900, he presented 23 unsolved problems
- His 10^{th} problem:
  - For any given Diophantine equation, can we *decide* whether the equation has a solution?
- This problem remained unsolved for 70 years, until 1970

Decidability

- In computer theory, an undecidable problem is one which it is impossible to construct an algorithm that can always lead to a yes or no answer
- Hilbert called it ***Entscheidungsproblem*** (decision problem)

- Example: Halting problem

Halting Problem

- Can you write a computer program *P* that:
  - For a given input, a computer program *i,* determine whether *i* will:
    - Halt (exit execution)
    - Run indefinitely

- To prove this, you must show that
  - Decide on ALL existing programs, even ones that have not existed
  - OR use prove by contradiction (show an instance of the problem that cannot logically exist)

Alan Turing's Paper in 1936

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

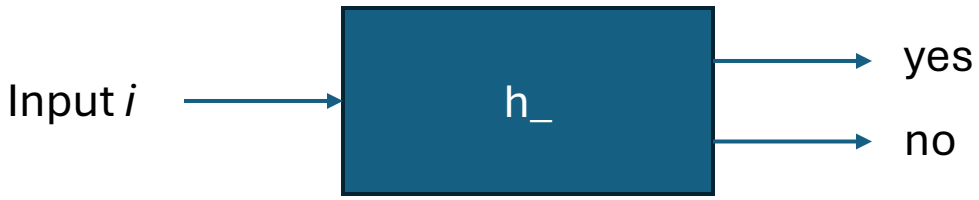
By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

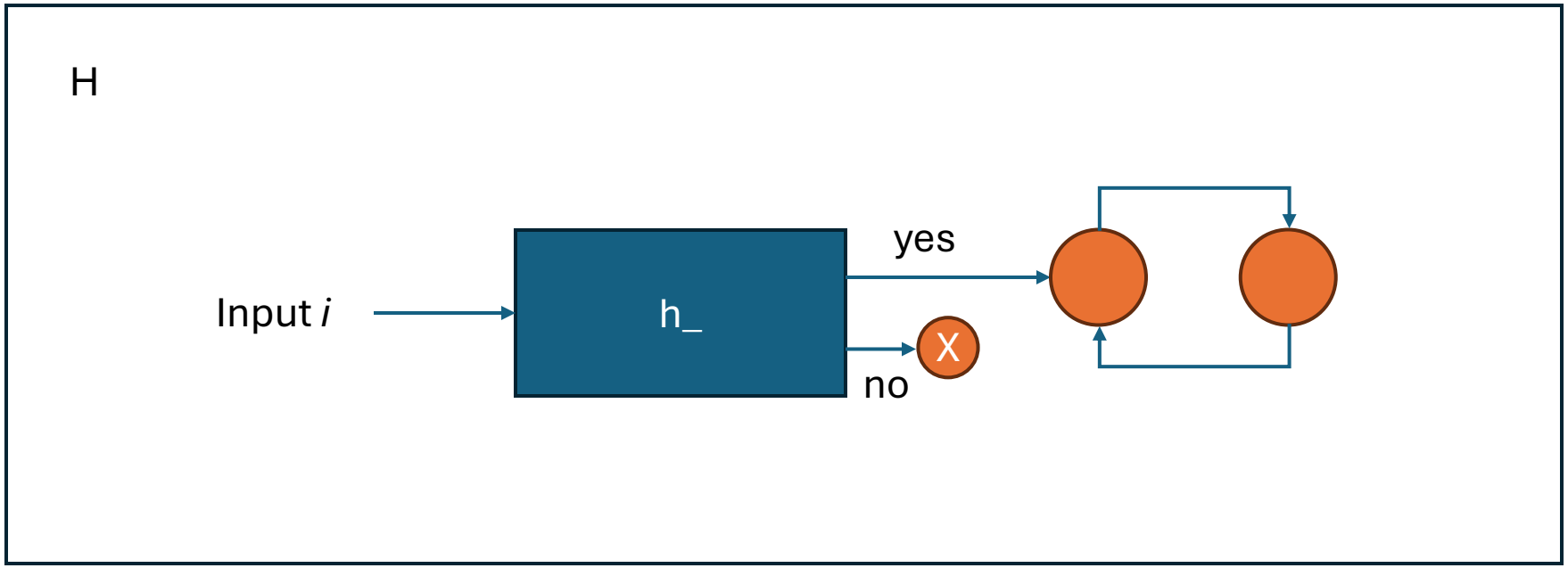
Halting Problem

- Suppose there exist a program *h_* that can magically decide whether a program halts or not



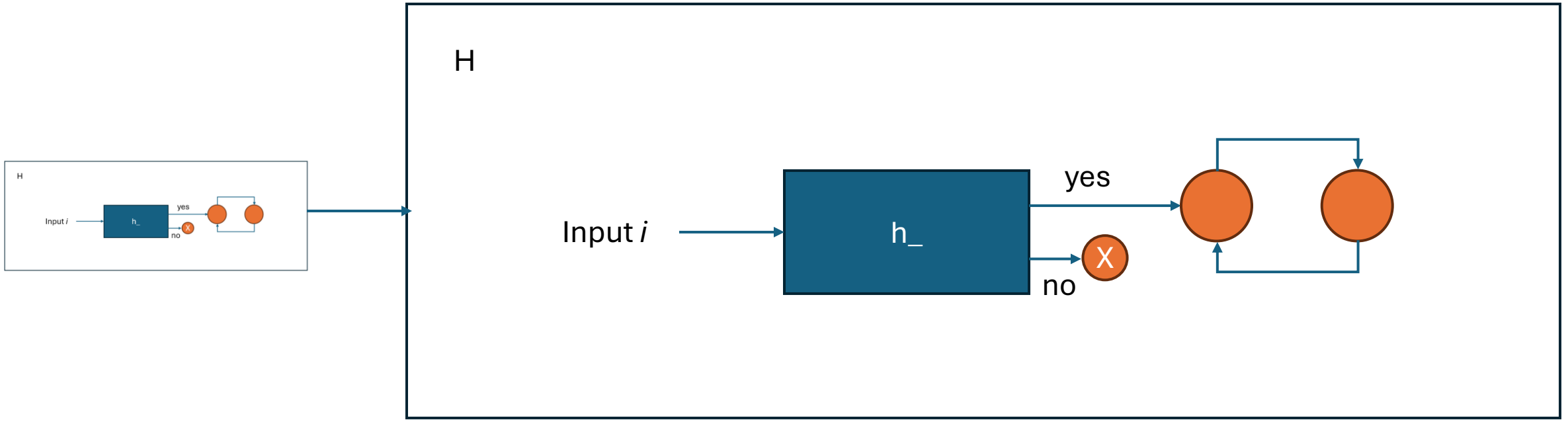
Halting Problem

- Let's construct a new program *H*, based on *h_* as follows
  - If *h_* says that the problem halts, then run forever
  - Else, halt



Contradiction

- What happens when we feed the program an instance of itself?
  - If *H* halts, then *h_* returns yes; but it runs forever
  - If *H* does not halt, then *h_* returns no; but it does halt



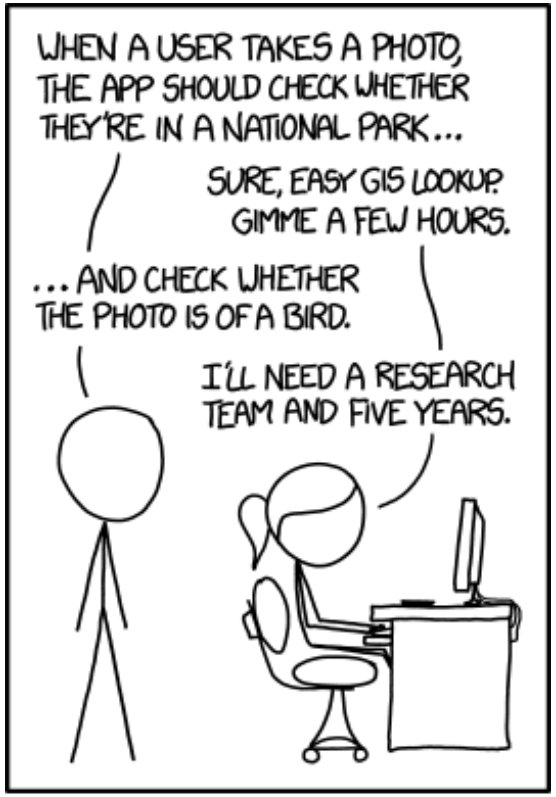
Halting Problem - Results

- The halting problem showed an instance of a problem that *cannot be solved by computers*
- This problem showed that some functions are mathematically definable, but not computable

Implications

- Actually equivalent (can be reduced) to Hilbert's 10^{th} problem
  - Proved by Matiyasevich
- Previously, we discussed that exhaustive testing is mathematically infeasible
- Microsoft rolls out a new update, and they want to make sure that it does not break backwards compatibility at all. This is actually just the halting problem
- Practically, it is to be able to tell your boss that "what you are asking for is mathematically impossible, and I can prove it"
  - Write a program that can automatically debug other programs

XKCD - 1425


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Summary

- One of the most important results in our field
- Halting problem showed that there exist problems in which computers cannot compute
- Some what defines the limits of what programming can do

Person of the day
David Parnas

- Another prominent figure in software engineering
- Information hiding
- Modularity, component-based reuse
- Separation of concerns
- Software of defence planes
- Abstraction
- Scalability
- Design patterns are really based on things that Parnas came up with