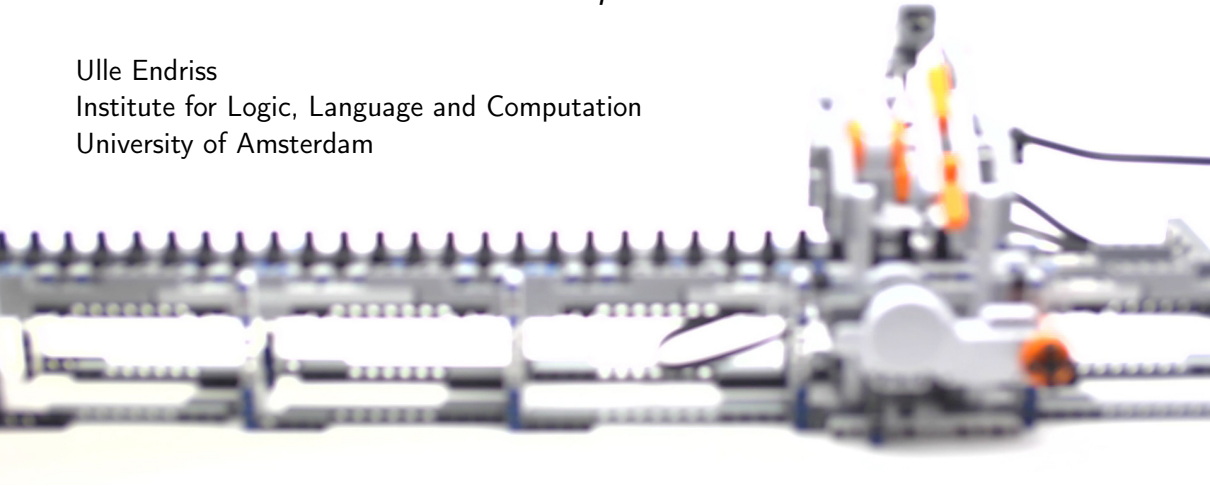
Turning Points in the Information Sciences

Lecture 4: *What can be computed?*

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam



Looking Back

So far you learned how to collect, encode, store, transmit, and quantify information.

Now we want to *do* stuff with information. Now we want to compute.

Looking Ahead

This part of the course (three lectures) is all about computing:

- What can be computed (at all)?
- What can be computed efficiently?
- What can be computed on a quantum computer?

Our focus will be on the very nature of computing and its fundamental limitations.
Such questions are studied in Theoretical Computer Science.

Computing

Computing is the process of transforming a given input into a desired output.

Both input and output must be finite pieces of information,
such as natural numbers or strings of symbols.

Examples: Given *Input*, Return *Output*

- Given a natural number, return the square of that number!
- Given a word (a string of letters), return the number of vowels in that word!
- Given a chess board configuration, return whether White can force a win (yes/no)!
- Given a natural number, return the next even number that cannot be written as the sum of two primes! Or return "*none*" if no such even number exists!
- Given a Python program and its input, return whether it will generate an error!

Exercise: *For which of these problems has humanity found solutions?*

`wooclap.com` → `UVAINF`

Exercise Solutions

- *Squaring numbers: we can do it!* 😀
- *Counting vowels: we can do it!* 😀
- *Solving chess: we cannot do it, but might one day.* 😕
- *Sum-of-primes: we cannot do it, but might one day.** 😕
- *Program correctness: we cannot do it . . . and we never will.* 🤔

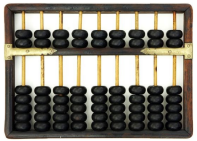
*This is Goldbach's Conjecture from 1742.

Beyond Computing

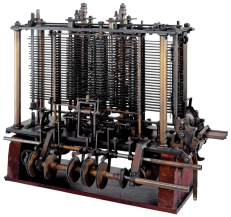
We only care about problems with well-defined solutions. So this does *not* qualify:

Given five keywords, return a beautiful poem about those keywords!

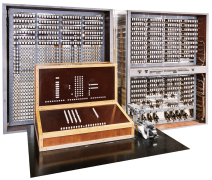
Early History of Computing



Abacus, since ∼1,000 BCE
arguably first calculating device



Charles Babbage's Analytical Engine, 1837
arguably first mechanical computer
"programmed" by Ada Lovelace



Konrad Zuse's Z3, 1941
arguably first electro-mechanical computer
(almost) universally programmable

The Need for Theory

"Will we ever solve problem X?"

This is not a question about some specific future technology.

It's a question about the fundamental limitations of *any* conceivable technology.

It's about theory.

Turning Point: Turing's 1936 Paper on Computability

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

decision problem

Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, **42**:230–265, 1936.

Turing Machines

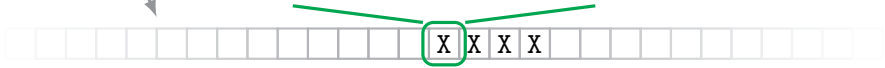
alphabet = set of symbols such as `0`, `1`, ..., `9`, `+`, `*`, `=`, `A`, `B`, ..., including blank ␣

set of states the TM might be in, including the initial state (often called q_0)

memory = infinite tape divided into cells to hold symbols (mostly blanks)

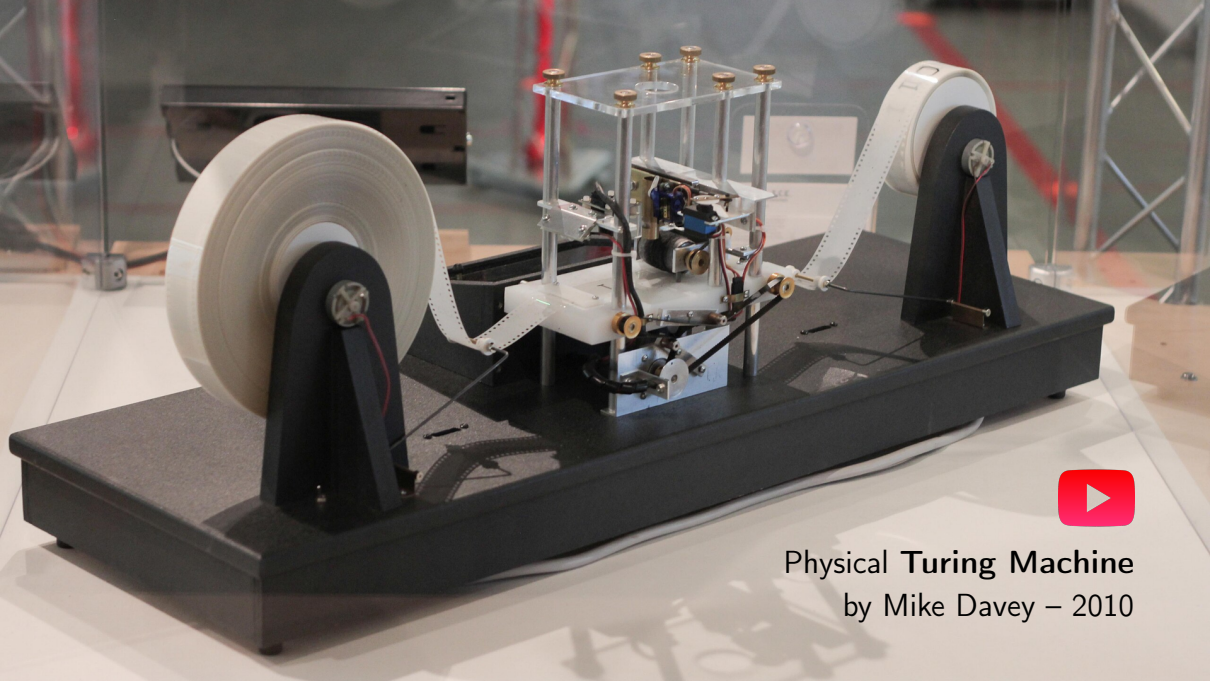
(state q_0, read `X`) ⇒
(state q_0, write `X`, »)
(state q_0, read ␣) ⇒
(state q_1, write `X`, «)
...

program = rules saying for given state and symbol read: state to switch to, symbol to write, direction to move in



head to read and write symbols (starts at first non-blank symbol)

Exercise: *What problem does this TM solve?*



Physical **Turing Machine**
by Mike Davey – 2010

The Church-Turing Thesis

The Church-Turing Thesis reflects the significance of Turing Machines:

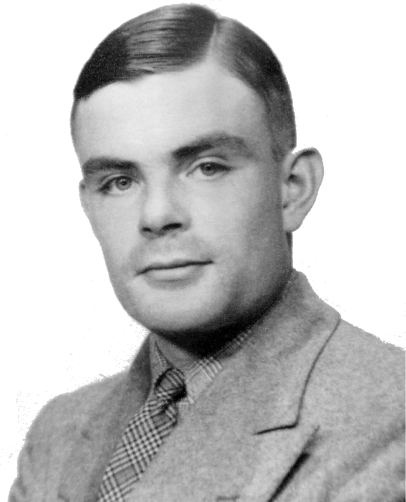
- Formal proof that Turing's definition of computation is logically equivalent to other accepted definitions (such as the one by Alonzo Church).
- Widely held conviction by experts that Turing's definition also coincides with the intuitive (but ultimately undefinable) concept of "computation".

A programming language is called Turing-complete if it can do everything a TM can.

Examples: Python, Java, C++, Haskell, Prolog, ...

Alan Turing

- Born 1912 in London
- Studied Mathematics at Cambridge
- Turing Machine paper published in 1936
- Code Breaking at Bletchley Park during WW2
- Turing Test paper published in 1950
- Indecency conviction (for homosexuality) in 1952
- Committed suicide in 1954 (at the age of 41)
- Pardoned by British Government in 2013



Food for Thought

Turing's 1936 paper is widely considered the beginning of Computer Science.

But he developed these ideas several years *before* there were electronic computers.

Message: Theorising about things that do not yet exist can have huge impact!

LEGO **Turing Machine**

Centrum Wiskunde & Informatica

Amsterdam – 2012

Limitations

Are there limits to what we can compute?

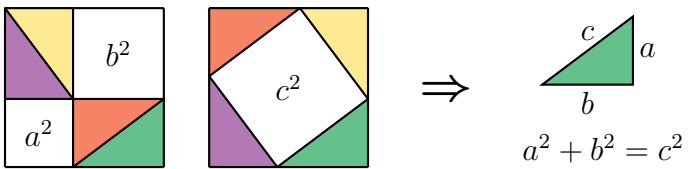
Even on the biggest and fastest computer in the world?

Even on a Turing Machine?

Foundations of Mathematics

Central to Mathematics is the notion of proof: demonstrating beyond any doubt that a given claim really follows from our assumptions. But not always clear what counts.

Example: *Do those "pictorial" proofs of Pythagoras' Theorem really count?*



Even more fundamentally, we might wonder whether for every true claim there exists a proof somewhere out there (even if it might be extremely difficult to find it).

Example: *Assuming it's true, will we ever find a proof of Goldbach's Conjecture?*

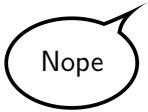
Remark: Note the parallels to our question regarding the limits of computation!

The *Grundlagenkrise* in Mathematics

The *Grundlagenkrise* [foundational crisis] refers to the profound and often antagonistic debate amongst mathematicians of the early 20th century regarding these questions.

David Hilbert, arguably the most prominent mathematician of his time, was convinced that every mathematical question will get settled eventually, one way or the other.

But in 1931 Kurt Gödel showed that, for any sufficiently rich system of Mathematics, there exist true statements that just cannot be *proved* true within that system.



Undecidability

Just as there are true yet unprovable mathematical claims,
there are undecidable computational problems.

Do you find this surprising?

Maybe not at first: of course, some problems are *really* difficult.
But then again: *How can you possibly prove that it is impossible to find a solution?*

Turing had an ingenious idea for obtaining such a proof,
and the remainder of this lecture is about understanding that idea.

The Halting Problem

A basic question we might ask about a computer program:

"Given a program P and some data x, will P halt eventually when applied to x?"

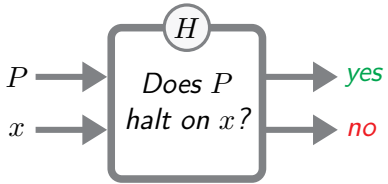
Can we build a machine / write a program to answer such questions?

In other words: Is the Halting Problem decidable?

(Never halting need not be bad. Think: operating system of your laptop.)

Undecidability of the Halting Problem

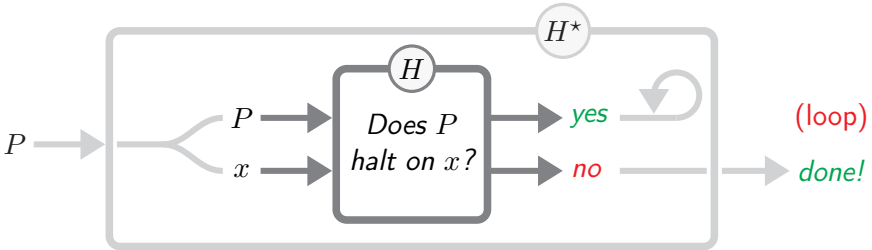
Assume (*for now*) a program H to solve the Halting Problem does exist.



Undecidability of the Halting Problem

Assume (*for now*) a program H to solve the Halting Problem does exist.

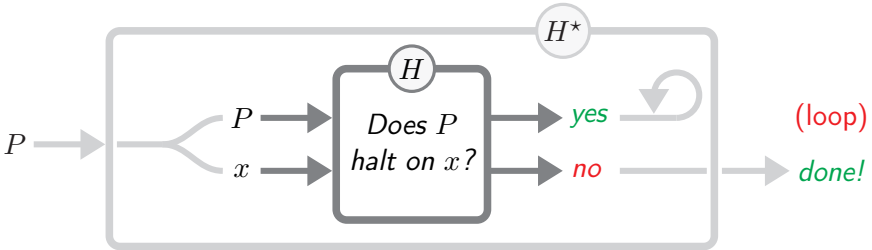
Then we can use H to construct a new program $H^\star$ that takes programs as input:



Undecidability of the Halting Problem

Assume (*for now*) a program H to solve the Halting Problem does exist.

Then we can use H to construct a new program $H^\star$ that takes programs as input:

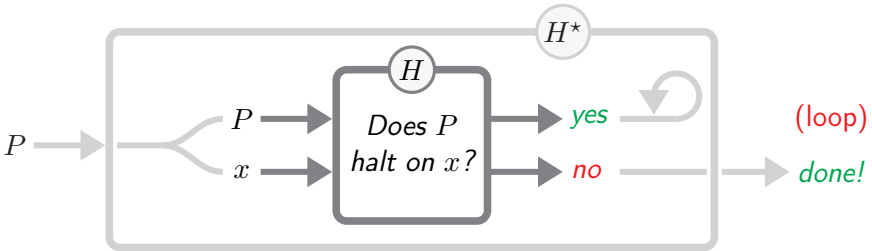


Exercise: *What happens if you apply $H^\star$ to itself? Will it halt?*

Undecidability of the Halting Problem

Assume (*for now*) a program H to solve the Halting Problem does exist.

Then we can use H to construct a new program $H^\star$ that takes programs as input:



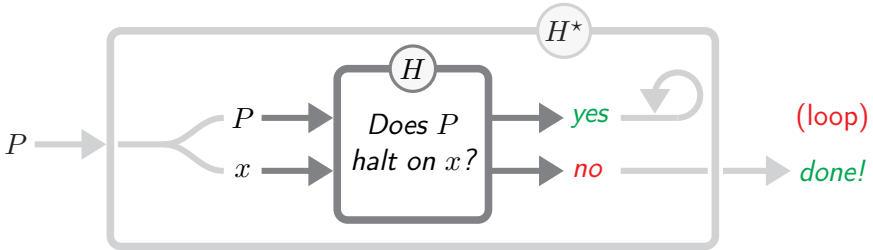
Exercise: *What happens if you apply $H^\star$ to itself? Will it halt?*

If it does, then it doesn't. If it doesn't, then it does.

Undecidability of the Halting Problem

Assume (*for now*) a program H to solve the Halting Problem does exist.

Then we can use H to construct a new program $H^\star$ that takes programs as input:



Exercise: *What happens if you apply $H^\star$ to itself? Will it halt?*

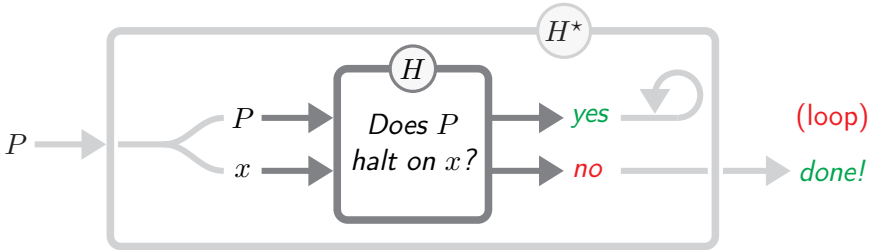
If it does, then it doesn't. If it doesn't, then it does.

So our assumption must have been wrong. The Halting Problem is undecidable. ✓

Undecidability of the Halting Problem

Assume (*for now*) a program H to solve the Halting Problem does exist.

Then we can use H to construct a new program $H^\star$ that takes programs as input:



Exercise: *What happens if you apply $H^\star$ to itself? Will it halt?*

If it does, then it doesn't. If it doesn't, then it does.

So our assumption must have been wrong. The Halting Problem is undecidable. ✓

Complete the quiz at wooclap.com (code: UVAINF) to test your understanding!

Tutorial

During the next tutorial, to prepare you for the next homework assignment, you will:

- Go over our proof of the undecidability of the Halting Problem once more
- Understand why also establishing program correctness is undecidable
- See an online Turing Machine simulator in action
- Program a Turing Machine to recognise palindromes

Recap

We started today's lecture by asking: *What can be computed?*

We then learned about two mind-boggling concepts to help us answer this question:

- Turing Machines: we saw how to give a general definition of "computing"
- Halting Problem: we saw that some problems cannot be solved by any computer

Ideas can be traced back to Alan Turing's 1936 paper on computability (turning point!).

What next? We'll refine our question and ask: *What can be computed efficiently?*

Picture Credits

Portrait of Ada Lovelace by Alfred Edward Chalon (Science Museum, London)

Photos of Alan Turing and Analytical Engine from Encyclopædia Britannica

Origin of photo of David Hilbert unclear

Photo of Kurt Gödel from Wikimedia Commons

Photo of Abacus from Computer History Museum, Mountain View

Photo of Z3 Replica from Deutsches Museum, Munich

Photos of LEGO Turing Machine by Andre Theelen (`vimeo.com/44202270`)

Photo of Mike Davey's Turing Machine found on Wikipedia (`aturingmachine.com`)

Emojis found on Emojipedia (`emojipedia.org`)

Other icons found on `flaticon.com`

Clipart from `pngwing.com`

Photo of AƁBA by Ollie Lindeborg / Getty Images