Limits of Computation

I - Intro & Motivation

Bernhard Reus

Let's step back in time

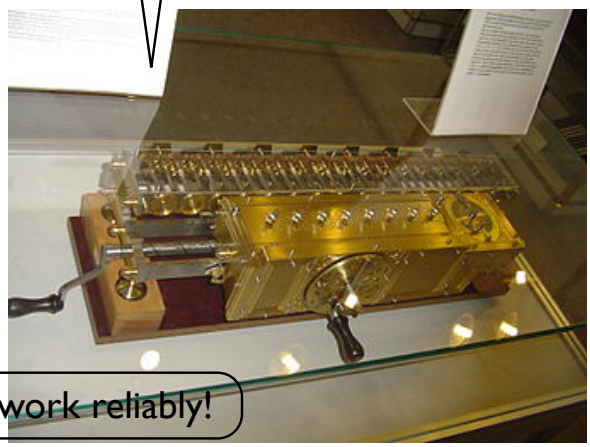
This man had a dream

1646-1716

maybe I'm the first computer scientist!

Gottfried Wilhelm Leibniz

The "stepped reckoner" (Staffelwalze): the first digital (mechanical) calculator performing arithmetic

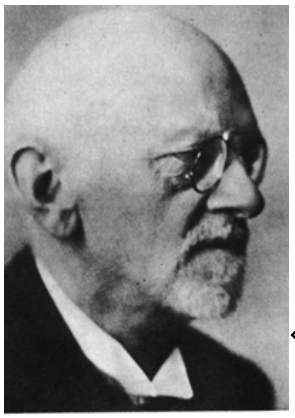


Alas, it did not work reliably!

invented differential & integral calculus (independently of Newton), modern formal logic, and more

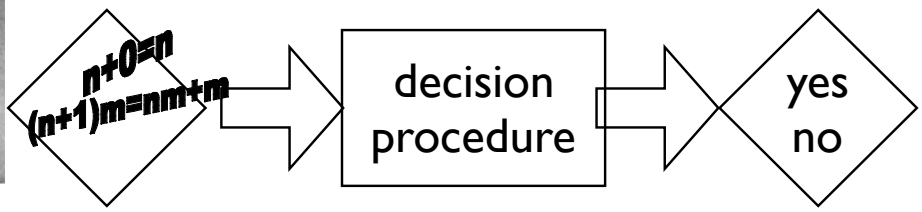
This man had a dream too

1928



Hilbert

"The *Entscheidungsproblem (German for "decision problem")* asks for a procedure which allows one to decide, using a finite number of operations, on the validity, respectively the satisfiability, of a given logical expression (in number theory)."



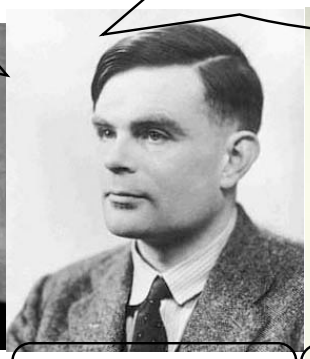
David Hilbert believed strongly that there *exists a solution* to the "*Entscheidungs*problem"

1932/1936

These guys shattered it

(formalisation of) arithmetic cannot be complete **and** consistent

true sentences of arithmetic cannot be decided



Kurt Gödel | Alan M Turing | Alonzo Church

"*Entscheidungs*problem"

There is no program/machine/procedure that can decide whether any given formula in arithmetic is actually true. Sorry David Hilbert!

Alan M. Turing: *'On computable numbers, with an application to the Entscheidungsproblem' from Proceedings of the London Mathematical Society, (Ser. 2, Vol. 42, 1937);*

Major Impact on Science

- not every problem (that can be clearly formalised and understood) can be solved (in this case the answer is yes/no only) by a program
- So, there are obviously limits to what programs can do!!

Sometimes we cannot compute it!

e.g. Hilbert's Decision Problem from earlier

Computability

- Are there problems that cannot be solved by a program on a computer?
- Does it matter which computer/language we use?
- Can one identify classes of problems that can be solved by programs or that can't be solved by programs?
- How can one see or find out that a problem can't be solved by a program?

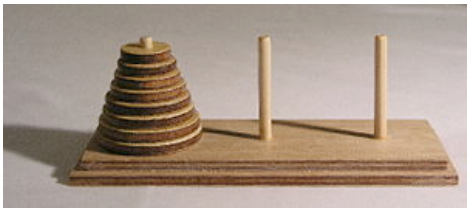
Sometimes we cannot afford it!

Annoyingly...

- ... even decidable (computable) problems sometimes are problematic.
- They may take a long time to compute.
  TOO LONG ACTUALLY!

Examples of *computable* problems that are *intractable*

Towers of Hanoi



- move all disks from 1st to 3rd rod
- only take/put disks from/on top of rods
- never put disk on a smaller disk.

Towers of Hanoi (cont'd)

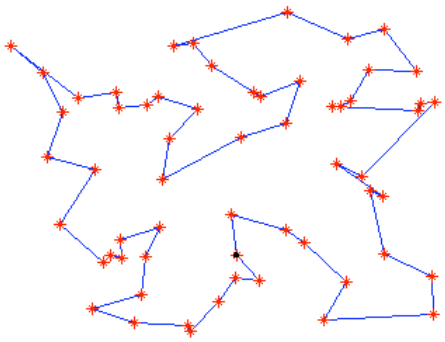
- for *N* disks it takes 2^N-1 moves to complete the task
- if you could move one million disks per second (!), for *N*=64 you'd still need about **585,000 years** to finish the job!

disks	moves
3	7
4	15
8	255
16	64535
32	4,294 bn
64	18,44 bn bn

Even for meditating monks this is too long!

Travelling Salesman Problem

- Given a number of cities and the costs of travelling from any city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the start?



Travelling Salesman Problem

- the number of tours to search through grows exponentially with the number of cities *N, namely (n-1)! / 2*
- Any brute force technique must fail. But are there clever ways?

cities	tours
3	1
4	3
8	2520
16	653.8 bn
32	4.11×10^{33}
64	0.99×10^{87}

There are only about 10^{80} atoms in the known universe.

Computational Complexity

- Which problems can be solved within certain limits of time (and space)?
- Are there resource limits within which certain combinatorial problems cannot be solved?
- Are intractable problems good for anything?
- Does adding resources allow one to solve more problems?
- How does one deal with intractable problems in practice?
- Can new emerging computing paradigms like *Quantum computing* and *Molecular computing* help?

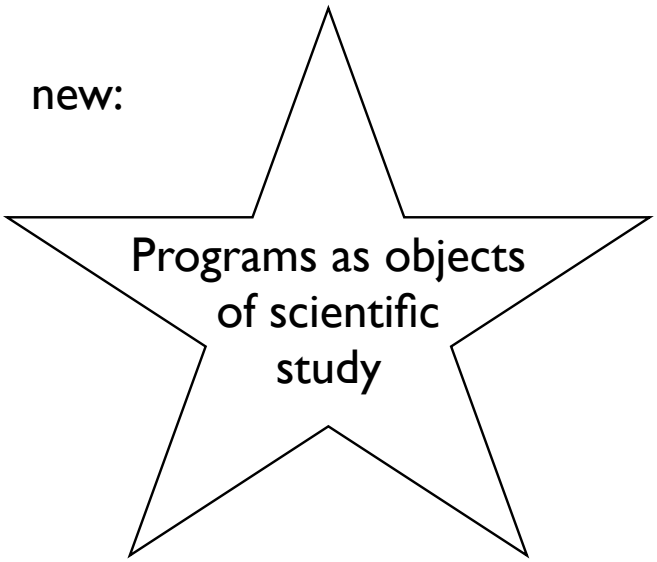
Why are these important questions?

Audience Participation

So what is this module for?

- you know the principles of programming
  - ➳ Programming Concepts
  - ➳ Data Structures & Algorithms
  - ➳ Program Analysis
- you know how to write programs

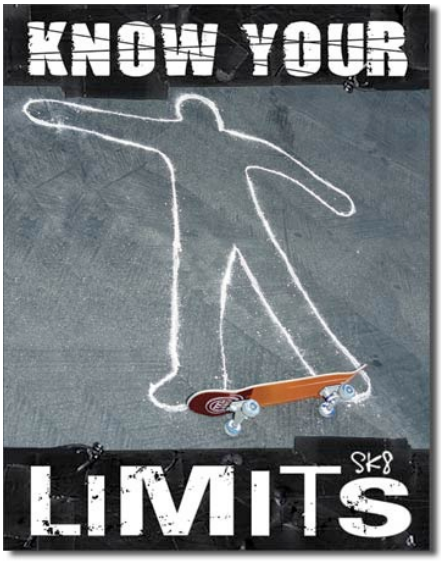
new:



Organisation 2020

Scientific Questions About Programs

- What are programs? What are their limitations?
- **What** are their limits in terms of what we can achieve with programs?
- How can we spot the limits?
- How can we circumnavigate them?
- Can we exploit those limits for a benefit?

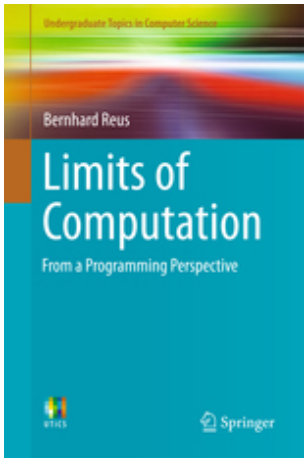


Organisation 2020

- **Lectures**
  Thurs 10am (Pev1-1A6), Fr 9am (Shawcross AS01)
- **Classes**
  Thurs 4pm, 5pm start in teaching week 2
  Sussex Direct tells you which group you're in, please stay there throughout the term
- **Coursework**
  Problem Set (60% week 6, March 5th)
  + Test (40% week 11, April 30th)
- **Assessment** 50% CWK + 50% UEX
- **Web** Canvas Site (with all info)

Course Text

There is a Reading List for this module on Canvas.



eBook £26.99
price for United Kingdom (gross)
Buy eBook
ISBN 978-3-319-27889-6
Digitally watermarked, DRM-free
Included format: EPUB, PDF
ebooks can be used on all reading devices
Immediate eBook download after purchase
Softcover £33.99

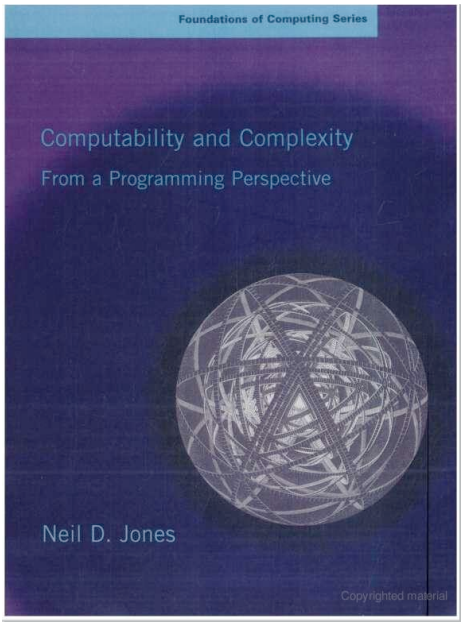
shameless plug

a few copies in the library

condensed notes will be available for free on Canvas

http://www.springer.com/services+for+this+book?SGWID=0-1772415-3260-0-9783319278872

Other literature



- Neil D Jones: *Computability and Complexity From a Programming Perspective*, MIT Press, 1997. **Online copy** on Canvas Links page.
- Also useful maybe: Hopcroft, Motwani, Ullmann: *Introduction to Automata Theory, Languages, and Computation,* Addison Wesley

END

Next time:
We define "Algorithms and Algorithmic Problems"