COMP0017
Computability and Complexity Theory

Fabio Zanasi
http://www0.cs.ucl.ac.uk/staff/F.Zanasi/

Lecture one

1. Practicalities

Course Structure

- Part I (15 lectures), taught by Fabio Zanasi

Computability Theory

- Part II (15 lectures), taught by Ran Ben Basat

Computational Complexity Theory

Assessment

One final exam (100%)

To pass:

Score at least 40% in the exam

To train:

Exercise sessions every week

Quiz will be made available on Moodle

Weekly Appointments

- **Two lectures (on campus)**

  Tuesday 12-14 **and** Friday 13-14

- **One exercise sheet (on Moodle)**

  Solutions published after the exercise session

- **Three exercise sessions (on campus)**

  Wednesday 11-12 (in parallel)

- **Discussion on forums (on Moodle)**

  Available all week, moderated by the TAs

Teaching assistants

Group 1

- Alexander Gheorghiu, <alexander.gheorghiu.19@ucl.ac.uk>
- Wojciech Różowski <w.Rozowski@cs.ucl.ac.uk>

Group 2

- Leo Lobski. <leo.lobksi.21@ucl.ac.uk>
- Tao Gu, <tao.gu.18@ucl.ac.uk>

Group 3

- Jas Semrl, <jas.semrl.15@ucl.ac.uk>
- Mateo Torres Ruiz <m.torresruiz@cs.ucl.ac.uk>

If you have any question on the exercises, or the course itself, feel free to contact the teaching assistants for your group.

Office Hours

Office hours are for:

- Discussing topics related to the course, on which you are curious about, but that were only touched in class.

- Clarifying the content of the course, provided that you attended the lectures, actually studied it first and tried to understand it yourself.

Office hours are not for:

- Solving the exercises together (exercise sessions serve this purpose, and you can contact the teaching assistants for further help).

- Catching up on lectures that you have missed or not studied.

- Last-minute exam panic.

Exam

What is in the exam?

Anything we discussed during the lectures, unless explicitly stated otherwise.

Content: slides + my explanation (you can use Sipser's book *Introduction to the Theory of Computation*, for the technical details and review of what's in the slides)

Type of questions: similar to the ones presented in exercise sessions and in the quiz, but usually simpler.

Prerequisites

Basic Set theory and basic logic

How to write down a *clear* and *rigorous* mathematical proof

Bonus

Proofs by contradiction

Finite state automata, grammars, formal languages

Bibliography

M. Sipser - Introduction to the Theory of Computation

A.J. Kfoury, R. N. Moll, M. A. Arbib - A Programming Approach to Computability

J. Hopcroft, J. Ullman - Introduction to Automata Theory, Languages and Computation

Questions?

2. Overview of the course

Pressing questions

- What programming languages should I learn?
- What software should I be able to use?
- What professional capabilities should I acquire?
- What is important in job interviews?

Do you expect that your answer will be the same in 2, 5 or 10 years?

Pressing questions

Learning is a never-ending process

Programming languages will change

Tools will change

The **basic concepts** will not change

Basic Concepts

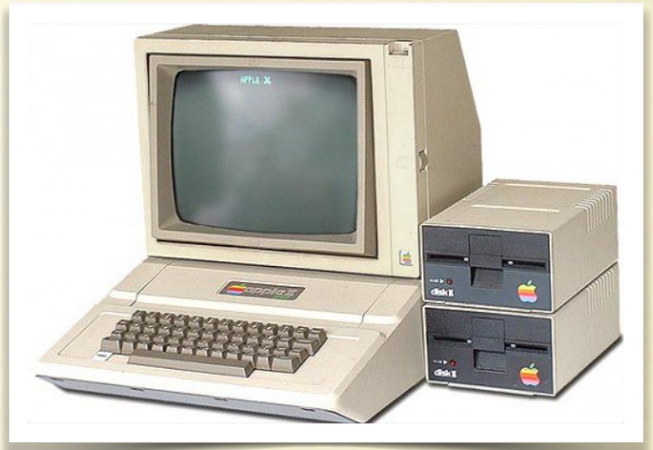
What is a computation?

What is a computer?

Are there limits to what a computer can do?

How should we *formally* approach these questions?
How can we *prove theorems* about them?

What is a computer?



What is a computer?

What distinguishes a computer from a non-computer?
What's the **essence** of computation?

A **theory** of computation demands an **abstract model** of what a computing machine is.

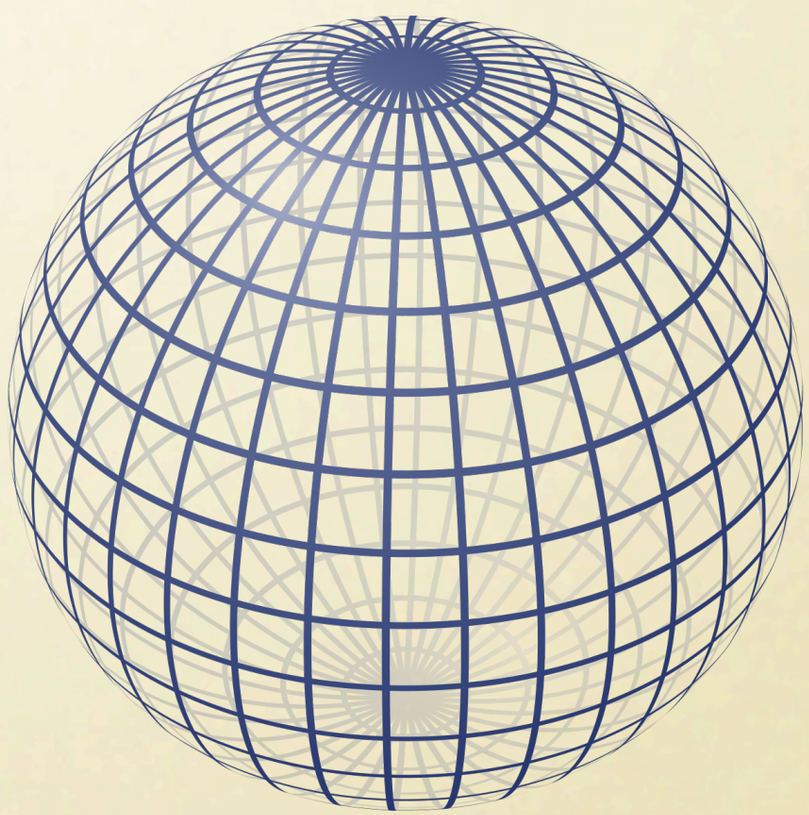
What is a computer?

Aristotle: distinction between the **essential** and **accidental** properties of a thing

A chair can be made of wood or metal, but this is accidental to its being a chair: that is, it is still a chair regardless of the material from which it is made.

What is a computer?

Geometry: in order to formally study **space**, we need to **abstract** the **essential** properties of objects



What is a computer?

We will see that this question has many **equivalent** answers.

It's a Turing Machine

It's a Register Machine

It's a WHILE programming language

It's a Recursive function

Church-Turing conjecture: anything that a human can compute following an algorithm can be computed by one of these abstract models.

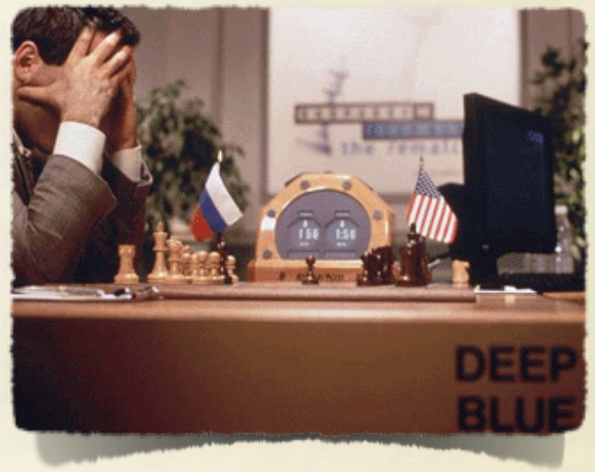
Basic Concepts

Are there limits to what a computer can do?

Are there limits to what a computer can do?

Can a computer beat the chess world champion?

1996



Can a computer beat the GO world champion?

2017

Can a computer drive a car safely?

???

Are there limits to what a computer can do?

There are problems that will *never* be solvable
by a computer.

A theory of computation can supply a
mathematical proof of this fact.

Unsolvable problems

Will the program P ever crash?

Will the program P halt on its input?

Given a program P and a function f, does P compute f?

Are any two given programs equivalent?

Unsolvable problems

In fact:

- most problems are undecidable.
  (Cantor's diagonalisation argument).
- all the non-trivial universal problems are undecidable.
  (Rice's Theorem)

Are we doomed?

- No, the trick is to consider restricted classes of a problem (e.g. replace "halt" with "halt in n seconds").

More unsolvable problems

In logic

Is the given first-order logic formula ϕ satisfiable?

In algebra

When do two words represent the same element in a group?

More unsolvable problems

In algebra

When a diophantine (= multivariable polynomial) equation has integer solutions

In interior design (!)

Can a given set of tiles used to cover any $n \times n$ area?

Unsolvable problems

Unsolvable problems are ubiquitous.

The power of abstraction: the majority has been discovered to be unsolvable **before** the first computer was ever built.

We will learn how to give a **mathematical proof** that a certain problem is unsolvable.

Basic Concepts

What is a computation?

What is a computer?

Are there limits to what a computer can do?

What about resources?

A solvable problem may still be **untractable** (take an unreasonable amount of time/memory space to be solved)

The second part of the course will focus on the **space/time complexity** of solvable problems.

Bibliography, revisited

M. Sipser - Introduction to the Theory of Computation

A.J. Kfoury, R. N. Moll, M. A. Arbib - A Programming Approach to Computability

J. Hopcroft, J. Ullman - Introduction to Automata Theory, Languages and Computation

Schedule

First week: introduction and the Turing machine

Second week: the Church-Turing thesis and other models of computation

Third week: the universal Turing machine. Undecidable problems in computation (halting and crashing)

Fourth week: Cantor's diagonal argument and Rice's theorem

Fifth week: undecidable problems in other fields (tiling problem, undecidability of first-order logic)

What's next

Our first abstract model of computation:

the Turing machine

Questions?