# Main message for today

What is this whole stuff all about?

Philosophical perspective: ambiguous, needs clarification, context dependent

Mathematical perspective: several well defined models of computation with non-trivial, interesting theorems

# Some intro and history

- First recorded algorithm is that of Euclid ($\sim$300 BC) for finding the greatest common divisor.
- The term *algorithm* derives from the name *al-Khwarizmi* (9th century), he writes in his book *The calculation of reduction and restoration:*

  > *"... what is easiest and most useful in arithmetic, such as men constantly require in cases of inheritance, legacies, partition, law-suits, and trade, and in all their dealings with one another, or where the measuring of lands, the digging of canals, geometrical computations, and other objects of various sorts and kinds are concerned."*

- Mechanical calculators. Goes back to ancient China

# Some history and intro

- Pascal's calculator 1642: invented a machine which was able to perform computations that were previously thought to be only humanly possible: addition and subtraction.
- Leibniz 1672: improved Pascal's calculator, was able to perform multiplications and divisions. (Leibniz believed that the principles of reasoning can be reduced to a formal symbolic system, an algebra or calculus of thought, in which controversy would be settled by calculations).
- Babbage:
  - Difference Engine No.1 (1832): compile mathematical tables.
  - Analytical Engine: intended to perform not just one mathematical task but any kind of calculation. His follower, Lady Lovelace, described how the engine might calculate Bernoulli numbers (the first 'computer program'). Though, the machine *never have built*.

# Some history and intro

- Leibniz, after having constructed his calculator, dreamt of building a machine that could manipulate symbols in order to determine the truth values of mathematical statements. He realized that the first step would have to be a clean formal language, and much of his subsequent work was directed towards that goal.
- Hilbert 1928 "Entscheidungsproblem": asks for an algorithm that takes as input a first-order formula and outputs "yes" or "no" according to whether the formula is valid. (By the completeness theorem: equivalent to ask whether an algorithm can decide a given formula is provable).
- 1936: Great year in history
  - Civil war in Spain
  - Economic recession
  - Berlin Olympics
  - Church and Turing published independent papers showing that a general solution to the Entscheidungsproblem is impossible.

# Some history and intro

- Church 1936: No computable function which decides for two given $\lambda$-calculus expressions whether they are equivalent. Credits also go to Kleene.
- Turing 1936: First machine-based model of what it means for a number-theoretic function to be computable, and the description of *Universal Turing Machine*[1]. Reduced the halting problem for Turing machines to the Entscheidungsproblem, relied on the idea of Gödel-numbering.

Both assumed that the intuitive notion of "computable" is captured by the functions computable by a Turing machine or by those expressible in the lambda calculus. This assumption is now known as the Church–Turing thesis.

[1] The idea of today's stored-program computers (von Neumann) descends from the universal machine.

# Mal'cev's criteria for algorithms (from: *Algorithms and Recursive Functions*)

a. An algorithm is a process for the successive construction of quantities. It proceeds in discrete time so that at the beginning there is an initial finite system of quantities given and at every succeeding moment the system of quantities is obtained by means of a definite law (program) from the system of quantities at hand at the preceding moment of time (*discreteness of the algorithm*).

b. The system of quantities obtained at some (not the initial) moment of time is uniquely determined by the system of quantities obtained in the preceding moments of time (*determinacy of the algorithm*).

c. The law for obtaining the succeeding system of quantities from the preceding must be simple and local (*elementarity of the steps of the algorithm*).

d. If the method obtaining the succeeding quantity from any given quantity does not give a result, then it must be pointed out what must be considered to be the result of the algorithm (*direction of the algorithm*).

e. The initial system of quantities can be chosen from some potentially infinite set (*massivity of the algorithm*).

The intuitive concept of an algorithm, although it is non-rigorous, is clear to the extent that in practice there are no serious cases when mathematicians disagree in their opinion about whether some concretely given process is an algorithm or not.

- Duplicability is an essential characteristic of any computer procedure (really?). Consider: machine throws a dice and prints its value. Mal'cev's criteria do not rule out this function is computable unless we are willing to argue that the entire system of machine and dice is not physically determined.
- Breaking the Turing barrier: hyper-computation, black holes, etc. Do mathematicians really agree? Martin Davies, *The Myth of Hypercomputation*:

  *"Despite the fact that is has been known for over a century that it is impossible to devise a construction for dividing a given angle into three equal parts using only straight-edge and compass, hopeful amateurs continue to bring forth construction that purport to do exactly that. [...] Because the computing machines that became available beginning in the 1950s were, in an important sense, physical realizations of the idealized computational models that had been developed by the logicians, it was generally held that it is impossible to construct a physical device capable of solving these "unsolvable problems". However in recent years a number of researchers, marching under the banner of "hypercomputation," unwilling to accept as fact this impossibility, have been making proposals to overcome this barrier. [...]"*

  (See also Gandy's thesis.)

# Approaches to formalizing computability

1. *Representability in a formal system* (Church, 1933, Gödel and Herbrand 1934). A formal system of arithmetic is taken and a function is declared computable if for every $m$ and $n$ for which $f(m) = n$ we can *prove* $f(m) = m$ in the system.
2. $\lambda$-*calculus* (Church, 1936). Closely related to the previous one. Church takes a formal alphabet and a language together with a notion of derivability that mocks the idea of proof yet seems simpler. Everything is reduced to manipulation of symbols.
3. *Arithmetical descriptions* (Kleene, 1936). Based on generalizing the notion of definition by induction. A class of functions that includes $+$ and $\cdot$ is closed under some simple rules, like definition by induction. This yields the class of $\mu$-recursive functions.
4. *Machine-like descriptions*. There have been several attempts to give a mathematical model of machine. Each tries to formalize the intuitive notion by giving a description of every possible machine
   a. Turing machines (1936)
   b. Markov algorithms (1954)
   c. Unlimited register machines (Shepherdson and Sturgis, 1963)
   d. RAM machines.
5. Oracle machines; Various other exotic machines based on e.g. higher infinite ordinals, or fancy physics.

# Approaches to formalizing computability

These formalizations are all purely syntactical, despite the often anthropomorphic descriptions. Quotation from Mostowski, *Thirty years of Foundational Studies*:

> *"However much we would like to "mathematize" the definition of computability, we can never get completely rid of the semantic aspect of this concept. The process of computation is a linguistic notion (presupposing that our notion of language is sufficiently general); what we have to do is to delimit a class of those functions (considered as abstract mathematical objects) for which there exists a corresponding linguistic object (a process of computation)."*

Assignment:

- Hermes: Enumerability, Decidability, Computability
- Odifreddi: Classical Recursion Theory, Chapter I.8 (issue of a process being mechanical vs. being computable).

# Plan

Our plan for the semester, roughly:

- Primitive recursive, recursive and partial recursive functions
- Finite state automata, Turing machines
- Coding, equivalence, hierarchy, classical theorems
- Words on the Church-Turing thesis and its connection to set theory, constructivism, $V_\omega$.
- Surveying other models (e.g. RAM, hypercomputation)