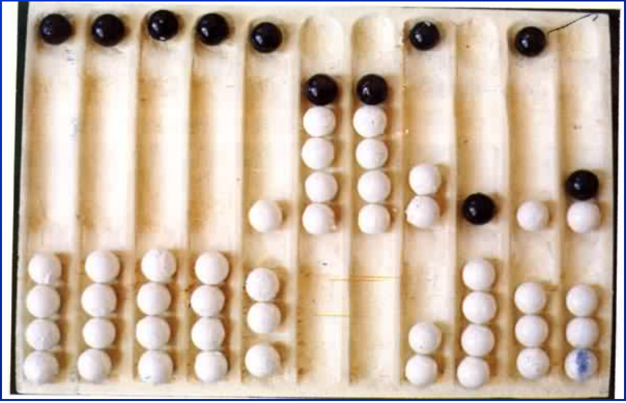
From the Foundation of Mathematics to the Birth of Computation

Fairouz Kamareddine
Heriot-Watt University
Edinburgh, UK

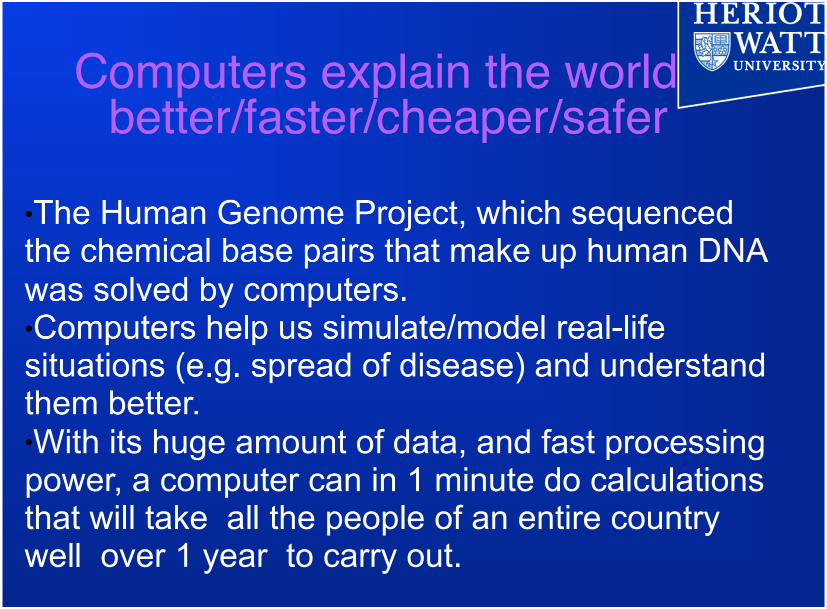
Goiana, Brasil, 17 May 2017

Computers are everywhere

An Old Computer: ABACI (2700 years BC) Mesopotamia Leonardo de Pisa (Fibonacci) (libre Abaci)



A New Computer 5000 years later



Computers explain the world better/faster/cheaper/safer

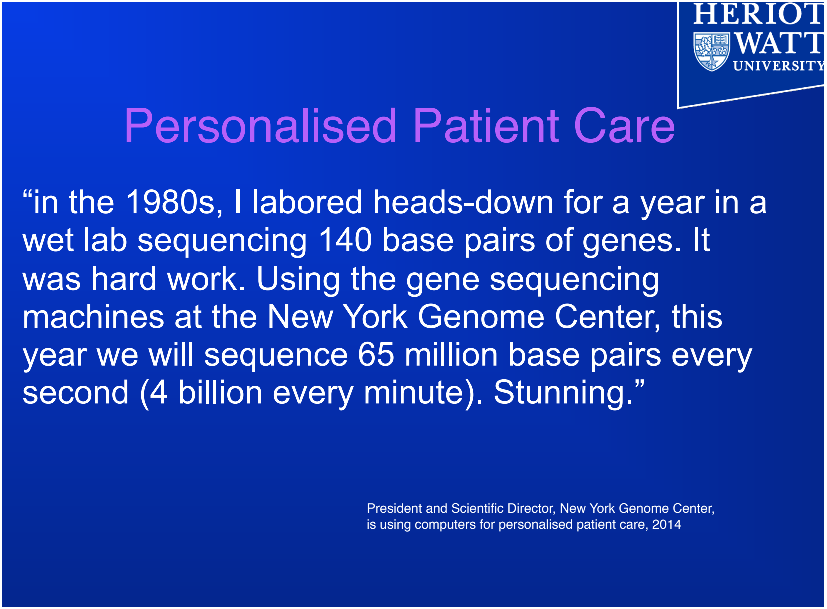
- The Human Genome Project, which sequenced the chemical base pairs that make up human DNA was solved by computers.
- Computers help us simulate/model real-life situations (e.g. spread of disease) and understand them better.
- With its huge amount of data, and fast processing power, a computer can in 1 minute do calculations that will take all the people of an entire country well over 1 year to carry out.

Are you safe in the air without computers?

Computers improve farming

Computers are used in surgery

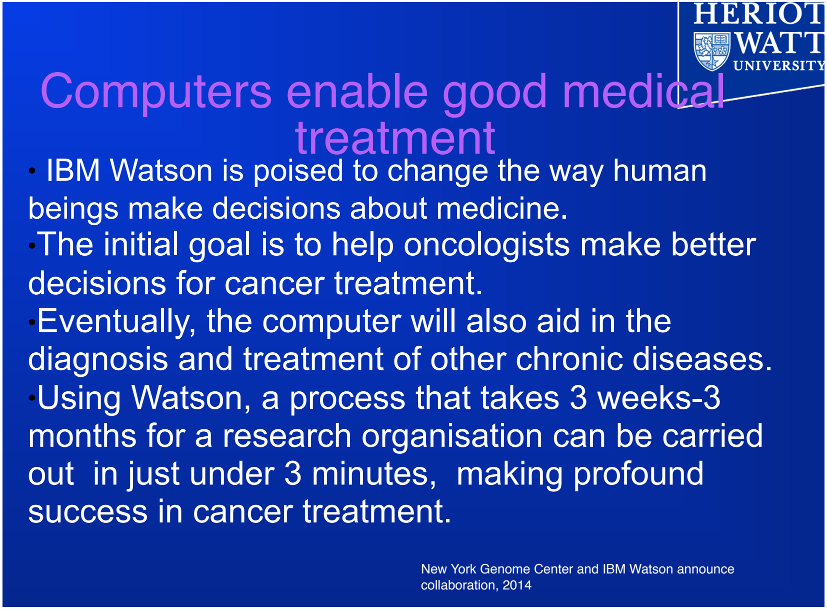
Computer Tech Scans Use computer programs to explain people's internal structure



Personalised Patient Care

“in the 1980s, I labored heads-down for a year in a wet lab sequencing 140 base pairs of genes. It was hard work. Using the gene sequencing machines at the New York Genome Center, this year we will sequence 65 million base pairs every second (4 billion every minute). Stunning.”

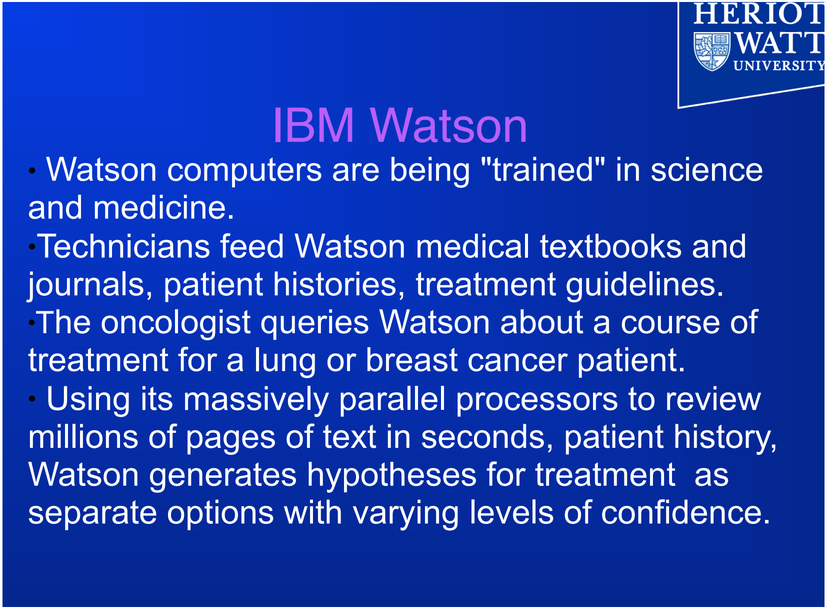
President and Scientific Director, New York Genome Center, is using computers for personalised patient care, 2014



Computers enable good medical treatment

- IBM Watson is poised to change the way human beings make decisions about medicine.
- The initial goal is to help oncologists make better decisions for cancer treatment.
- Eventually, the computer will also aid in the diagnosis and treatment of other chronic diseases.
- Using Watson, a process that takes 3 weeks-3 months for a research organisation can be carried out in just under 3 minutes, making profound success in cancer treatment.

New York Genome Center and IBM Watson announce collaboration, 2014

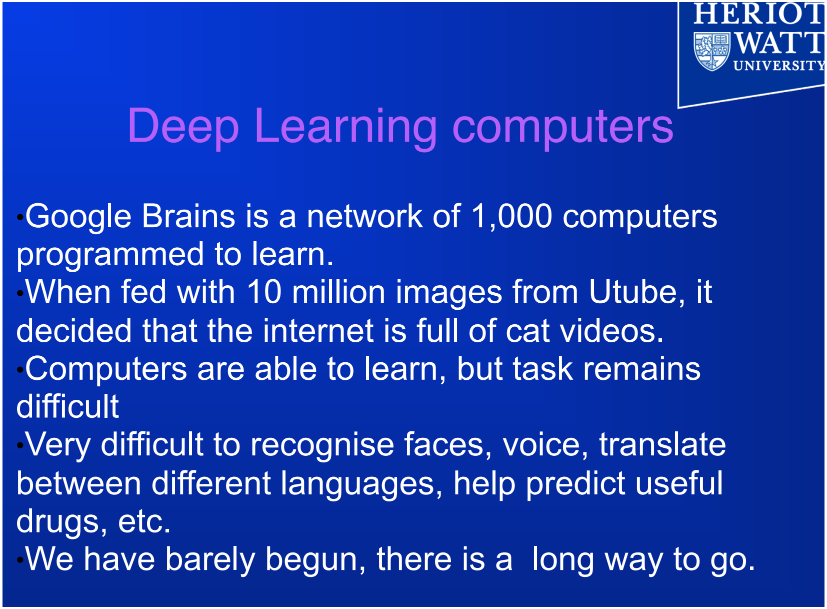


IBM Watson

- Watson computers are being "trained" in science and medicine.
- Technicians feed Watson medical textbooks and journals, patient histories, treatment guidelines.
- The oncologist queries Watson about a course of treatment for a lung or breast cancer patient.
- Using its massively parallel processors to review millions of pages of text in seconds, patient history, Watson generates hypotheses for treatment as separate options with varying levels of confidence.

They help us solve crimes

Deep Learning computers

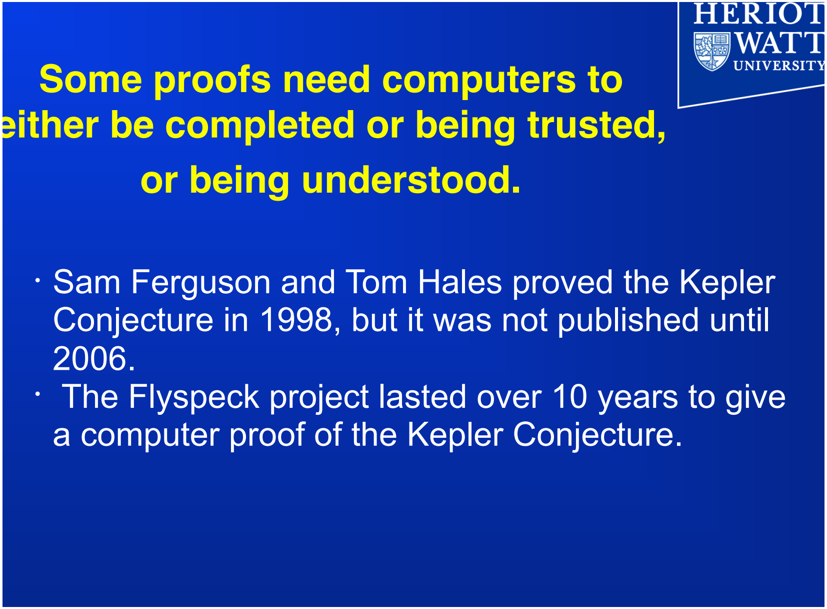


Deep Learning computers

- Google Brains is a network of 1,000 computers programmed to learn.
- When fed with 10 million images from Utube, it decided that the internet is full of cat videos.
- Computers are able to learn, but task remains difficult
- Very difficult to recognise faces, voice, translate between different languages, help predict useful drugs, etc.
- We have barely begun, there is a long way to go.

Some proofs need computers to either be completed or being trusted, or being understood.

The Kepler Conjecture 17th century: no packing of congruent balls in Euclidean space has density greater than the density of the face-centered cubic packing (74.04%).

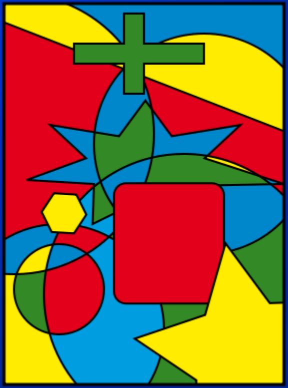


Some proofs need computers to either be completed or being trusted, or being understood.

- Sam Ferguson and Tom Hales proved the Kepler Conjecture in 1998, but it was not published until 2006.
- The Flyspeck project lasted over 10 years to give a computer proof of the Kepler Conjecture.

The Four colour theorem:

given any separation of a plane into contiguous regions, producing a figure called a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color



First Major theorem proved using computer

- In any separation of a plane into contiguous regions producing a map, no more than four colours are required to colour the regions so that no 2 adjacent regions have the same colour.
- 1976 by Appel and Haken: huge bits by hand and huge bits by computer.
- Proof not acceptable: Computer assisted part infeasible for human to check by hand.

Revised proof and full computer checking

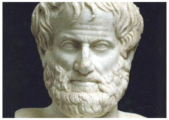
- A revised version of earlier proof was given in 1997.
- Would not have been possible to give it if we didn't have the original proof.
- The correctness of theorem was computer checked in 2005.

Fifty Years of a whole variety of proof assistants and programming languages

- The proof of the Kepler's conjecture was checked in a mixture of Isabelle and HOL Light.
- The proof of the four color theorem was checked in Coq.
- Landau book on foundations of Analysis was checked in Automath.
- 60% of the Compendium of Complete Lattices, was checked in Mizar.
- Numerous other provers/checkers. Why were they created? And what are they used for?
- Not to forget a huge number of programming lanaguages/paradigms each with a different purpose.

Logic/Mathematics/Computation: A word of warning

- Logic is *OLD*. Mathematics is *OLD*. But, *SO IS* computation.
- Just like in the times of ancient China, of Aristotle in Greece, of Euclides in Alexandria Egypt, or of AlHambra/Andalucia or of Modern Europe (Frege/Russell), deduction/Logic was taken as a foundation for thought.
- Computation was also taken throughout as an essential tool in mathematics, in astronomy, in architecture, in farming, etc.
- Our ancestors used sandy beaches to compute the circomference of a circle, and to work out approximations/values of numbers like π.
- The word algorithm dates back centuries? Algorithms existed in anciant Egypt at the time of Hypatia. The word is named after Al-Khawarizmi.
- But even more impressively, the following important 20th century (un)computability result was known to Aristotle.



- Assume a problem Π,
  - If you *give* me an algorithm to solve Π, I can check whether this algorithm really solves Π.
  - But, if you ask me to *find* an algorithm to solve Π, I may go on forever trying but without success.
- But, this result was already known to Aristotle:
- Assume a proposition Φ.
  - If you *give* me a proof of Φ, I can check whether this proof really proves Φ.
  - But, if you ask me to *find* a proof of Φ, I may go on forever trying but without success.
- In fact, *programs* are *proofs*:
  - *program* = *algorithm* = *computable function* = *λ-term*.
  - By the PAT principle: *Proofs* are *λ-terms*.

Much later than Aristotle, Leibniz (1646–1717) conceived of **automated deduction**, i.e., to find

- a language L in which arbitrary concepts could be formulated, and
- a method to determine the correctness of statements in L.

In other words, Leibniz wanted a language and a method that could carry out proof checking and proof finding. However, according to Aristotle and (later results by) Gödel and Turing, such a method can not work for every statement.

A short history of numbers

- *natural numbers* like 0, 1, 2, which were used to count (sheep for example);
- *integers* like 0, 1, -1, 2, -2, etc. which were also used to count;
- *rational numbers* which are the quotients or fractions of integers like $2/3$ and which were used to measure (the anciants used *anthyphairesis/Alternated substitution* to evaluate Ratios);
  - $2/3 = 0.6666666...$ where 6 repeats over and over
  - $41/333 = 0.123123123123...$ where 123 repeats over and over.
- *irrational numbers* like $\sqrt{2}$, $\sqrt{3}$, π;
  - $\pi = 3.14159265358979323846264338...$
  - $\sqrt{2} = 1.4142135623730950488016887242096980785696718753769 4...$

The calculus and the paradoxes of motion

- Zeno of Elea, C. 590-525 B.C.E., devised some paradoxical arguments against the possibility of motion.
- Since the calculus was developed partly to deal with motion, these paradoxical arguments are important for the foundations of analysis.
- Three of the most important of these are in Aristotle in his Physics.
  - *Dichotomy. There is no motion, because what moves must arrive at the middle of its course before it reaches the end.* In other words, to leave the room, you first have to get halfway to the door, then you have to get halfway from that point to the door, etc. No matter how close you are to the door, you have to go half the remaining distance before proceeding.
  - *Arrow. The flying arrow is at rest,* because a thing is at rest when occupying its own space at a given time, as the arrow does at every instant of its alleged flight.

It started with incommensurability

- According to Webster's dictionary, *commensurability* is *divisibility without remainder by a common unit.*
- Hence 6 and 9 are commensurable (since they are both divisible by 3).
- Attempts to find the unit which measures exactly the side and diagonal of a square led to the proof of the *incommensurability* of the side and diagonal of a square.
- This result on incommensurability implies that $\sqrt{2}$ *is not a rational number*. That is, it cannot be represented as the quotient of two integers.

With incommensurability, number was no longer everything

- The discovery of the incommensurability of the side and diagonal of a square showed that the Pythagorean idea that *number is everything would not work.*
- The Pythagoreans needed to treat quantities which are not numbers.
- For them, numbers are rationals and quantities are the incommensurable (which we call *real numbers*).
- The Greeks constructed geometric figures (recall Euclid), but took numbers as given. They separated numbers, which are discrete, from continuous magnitudes (quantities/real numbers).
- They did not use fractions to approximate continuous magnitudes.
- They did not construct the reals, nor multiply them nor divide them, etc.

Euclid used Euclid used anthyphairesis to find the greatest common divisor of two numbers

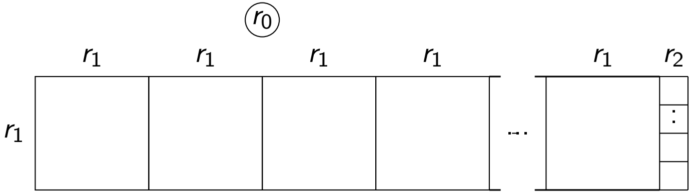


Figure 1: Anthyphairesis

Ratio of 12 to 5

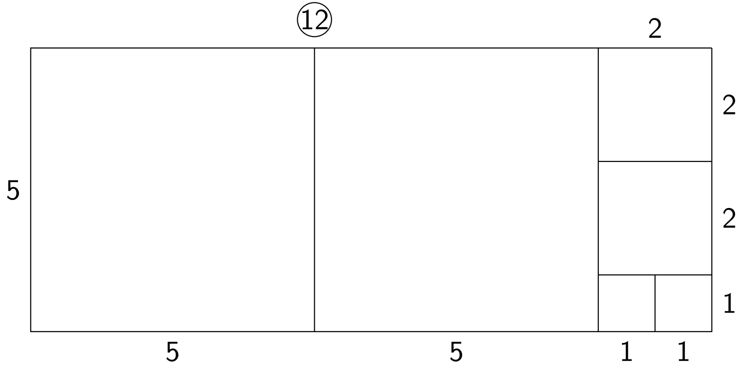


Figure 2: Ratio of 12 to 5

Ratio of 22 to 6

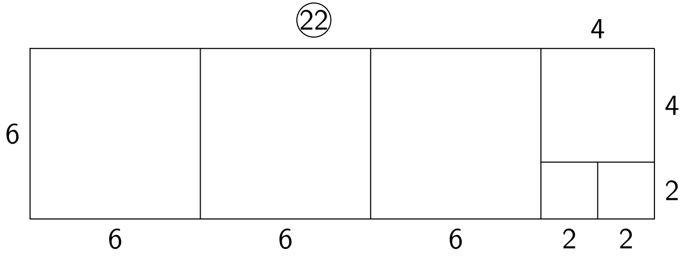


Figure 3: Ratio of 22 to 6

Ratio of $\sqrt{2}$ to 1

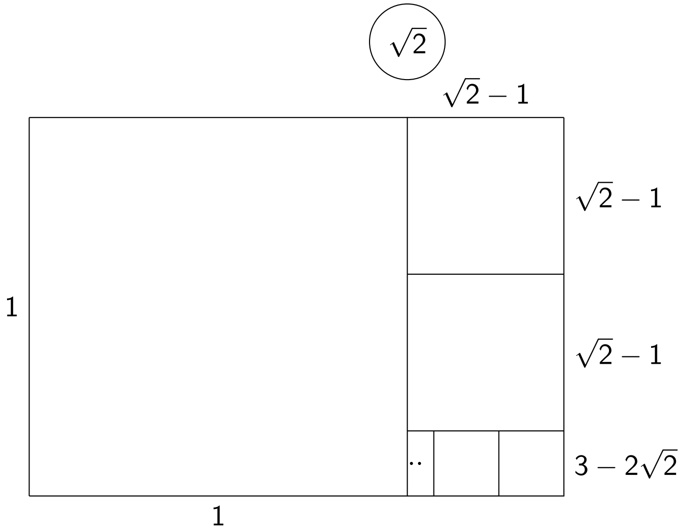
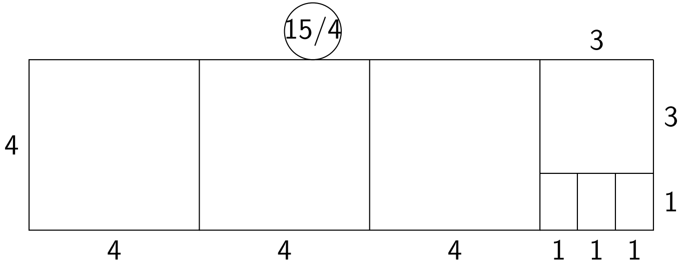
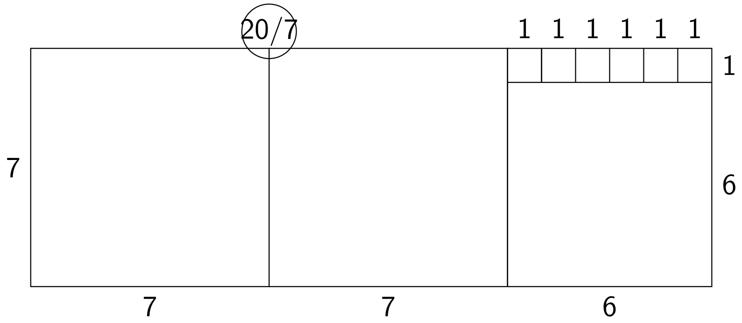


Figure 4: Ratio of $\sqrt{2}$ to 1

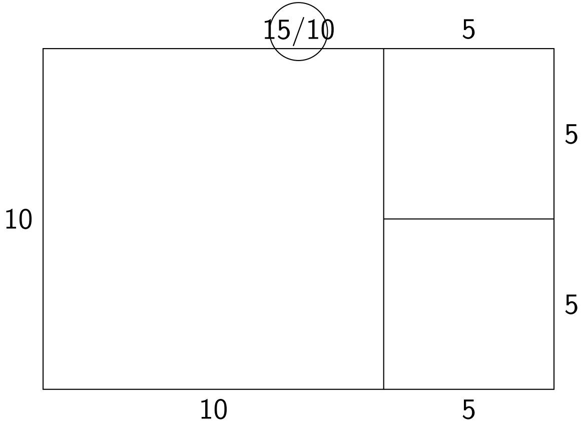
The ratio of 15 to 4 is $[3, 1, 3]$



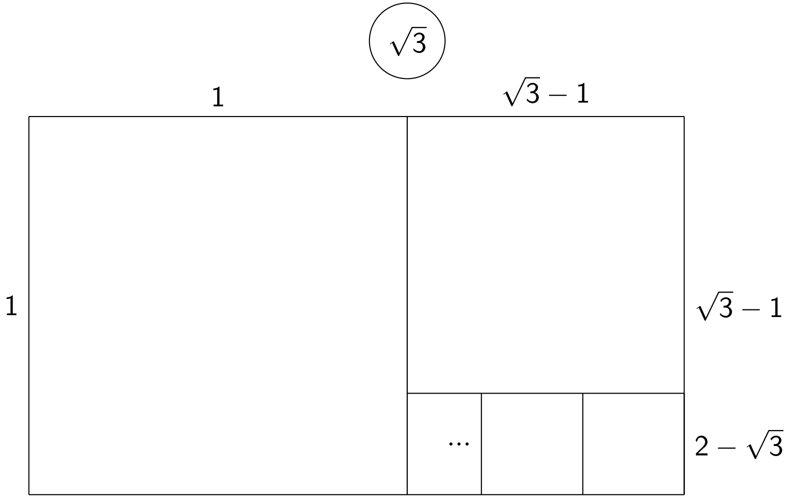
The ratio of 20 to 7 is $[2, 1, 6]$



The ratio of 15 to 10 is $[1, 2]$



The ratio of $\sqrt{3}$ to 1 is characterised by $[1, 1, 2, 1, 2, 1, 2, ...] = [1, \overline{1, 2}]$



Real numbers need to be constructed (using approximations like Dedekind cuts, Cauchy sequences, etc.)

- The idea of using fractions to approximate continuous magnitudes developed first in the Arab world during the middle ages in Europe, and came to Europe in the 16th and 17th centuries.
- This idea would have been assumed by both Newton and Leibniz.
- Although the Greeks did not construct magnitudes (real numbers), they still studied them after discovery of incommensurability.

In 18th and 19th century, irrational numbers were divided into two categories: *Algebraic* and *transcendental*

- A number is algebraic if it is the root of a non-zero polynomial with rational coefficients. For example, $\sqrt{2}$ is algebraic since it is the solution to $x^2 - 2 = 0$.
- An irrational number that is not algebraic is called *transcendental* (i.e. cannot be made of algebraic equations). For example, π is transcendental.
- Transcendental was coined by Leibniz in 17th cenetury who showed that $sin(x)$ is not an algebraic function of x.
- Until the discovery of irrationals like $\sqrt{2}$, the pythagorean expected all numbers to be rational.

The discovery of transcendental numbers

- Until the 17th century it was expected that numbers *should fit the algebraic mould.*
- In the *18th century* it was shown that π is irrational and *conjectured* that π is transcendental.
- In the 19th century, *proofs were given* of the existence of transcendtal numbers and that π *is transcendental.*
- Once π was shown transcendtal meant that the old problem of *squaring the circle became impossible.*

Researchers in the 19th century continued to go deeper into numbers

- 1821: Many controversies in analysis were solved by *Cauchy.* E.g., he gave a precise definition of convergence in his *Cours d'Analyse*.
- 1872: Due to the more *exact definition of real numbers given by Dedekind*, the rules for reasoning with real numbers became even more precise.
- 1895-1897: *Cantor began formalizing set theory* and made contributions to number theory.
- 1889: *Peano formalized arithmetic*, but did not seriously treat logic or quantification.
- *Cantor's diagnolisation argument and the size of the natural numbers versus the size of the real numbers* will impact the size of what can be *computable* versus what cannot.

- Cantor proved that *algebraic numbers are countable*.
- Hence there are only *as many algebraic numbers as there are natural numbers.*
- Cantor proved that *the transcendental numbers are uncountable*.
- Cantor proved that the size of the algebraic numbers is infinite, but is the *smallest infinite that exists*.
- The size of the transcendental numbers is a much much larger infinite.
- Think of the first inifinite number (the size of the algebraic numbers) as a dot of water.
- Think of the second infinite number (the size of the transcendental numbers) as a whole lake.

Depressing implication for computation

Later on it was shown that:

- The size of the computable functions is the size of the algebraic numbers, the smallest infinite $\aleph_0$.
- The size of the non-computable functions is the size of the transcendental numbers (the monster infinite), which according to Cantor's *Continuum hypothesis* is the infinite $\aleph_1$ which is the next one up after $\aleph_0$.
- This means that there are a lot more functions that are impossible to compute than there are computable functions.

Formal systems in the 19th century symbols (not natural language) define logical concepts

- This *need for a more precise* style in mathematics arose, *because controversial results* had appeared in *analysis*.
- We mentioned Cantor, Peano, Dedekind, Cauchy, etc.
- 1879:

  Frege was not satisfied with the use of *natural language in mathematics*:

  > *". . . I found the inadequacy of language to be an obstacle; no matter how unwieldy the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain the precision that my purpose required."*
  >
  > (Begriffsschrift, *Preface)*

  Frege therefore presented *Begriffsschrift*, the first formalisation of logic giving logical concepts via symbols rather than natural language.

A general definition of functions

"[Begriffsschrift's] first purpose is to provide us with the most reliable test of the validity of a chain of inferences and to point out every presupposition that tries to sneak in unnoticed, so that its origin can be investigated."

(Begriffsschrift, *Preface)*

- The introduction of a *very general definition of function* was the key to the formalisation of logic. Frege defined the Abstraction Principle.

Abstraction Principle

"If in an expression, [...] a simple or a compound sign has one or more occurrences and if we regard that sign as replaceable in all or some of these occurrences by something else (but everywhere by the same thing), then we call the part that remains invariant in the expression a function, and the replaceable part the argument of the function."

Russell's paradox due to self-application of functions Hilbert's program

- 1892-1903 Frege's *Grundgesetze der Arithmetik*, could handle elementary arithmetic, set theory, logic, and quantification.
- *Self-application of functions* (not in Begriffsschrift) was at the heart of *Russell's paradox 1902* [14].
- Also in the early 1900s, Hilbert, a master in posing difficult problems wanted to believe that any logical statement can either have a proof or be disproved.
- More than 30 years later, Hilbert's wish was negatively answered by Turing (Turing machines), Goedel (incompleteness results) and Church (λ-calculus).

Can we solve/compute everything?

- Following from Leibniz and Frege, interest grew and researchers were calling for *logical methods that could decisively answer questions at hand.*
- *Hilbert* was one of these most influential voices.
- Hilbert believed that every mathematical problem should *either have a solution* or we should definitely know that *no such solution exists.*
- For example, if given the problem of finding *an integer whose square is 2*, you should reply that there are no such integers (hence you know that this problem has *no solutions*).
- On the other hand, if given the problem of finding *an integer whose square is 4*, you should give *either integer 2 or integer -2* (hence this problem has 2 solutions).
- Hilbert did not want to allow for the unknown. He is famous for saying:

  "We must Know. We will know."

  (Hilbert)

- Back in 1928, Hilbert posed a problem which became known as *the Entscheidung Problem* or the *Decision Problem*.
- This problem dates back to *Leibniz* and asks for an *algorithm* that takes as input a statement of first order logic and returns as output one of two possible answers: *yes* when the statement is always valid, or *no* otherwise.
- So, for the statement *is there an integer whose square is 2* the answer is *no*.
- For the statement *is there an integer whose square is 4* the answer is *yes*.
- Hilbert advocated the idea (which became known as *Hilbert's program*) that there should be
  - *a complete* (i.e., every true formula can be derived) and
  - *consistent* (i.e., does not contain a contradiction)

  axiomatization of all of mathematics such that *every mathematical problem should either have a solution or we should definitely know that no such solution exist.*

- The results of the 1930s would establish the limitations of computers even before computers were built.
- No matter how fast and advanced computers get (and they are advancing at an amzing speed, considering that they did not exist in 1930s).
- Before we knew what computers could do, we had results telling us what computers could never do.
- These results of the limitations of the computer, will never change.
- They are set in stone just like the impossibility of squaring a circle.

Can we solve/compute everything?

- Turing answered the question via a *machine for running/computing programs*.
  a function f is computable iff f can be computed on a Turing machine.
- Church invented the λ-calculus, a *language for describing programs*.
  a function f is computable iff f can be described in the λ-calculus.
- Note that Church's λ-calculus was initially intended as a language of programs and logic, but it turned out to be inconsistent (Kleene and Rosser) and Church restricted the λ-calculus to programs.
- Goedel's result meant that *no absolute guarantee can be given that many significant branches of mathematics are entirely free of contradictions.*
- This means: we can compute a very small (∞ly countable, size of $\mathbb{N}$) amount compared to what we will never be able to compute (uncountable, size of $\mathbb{R}$).
- Hilbert's dream was shattered. According to historian of Mathematics Ivor Grattan-Guinness, Hilbert behaved coldly towards Goedel.

How did this foundational work influence programming?

- By the 1950s we had the computer, we knew what a computable functon is, and programming languages started in earnest.
- For example, untyped λ-calculus was adopted by John McCarthy in 1958 in the language LISP.
- Algol 60 (1958) and Algol 68 (1958) were also developed.
- Also, the earlier work of Frege, Russell and Whitehead, Hilbert, etc., on the formalisaton of mathematics, were now being complemented/replaced in the 1960s by the computerisation of mathematics.
- De Bruijn's Automath and Trybulec's Mizar were conceived around 1967.
- But before we can talk more about programming languages, theorem provers or the computerisation of mathematics, we need to go back and look at the Paradoxes and their solution and how this influenced on expressivity.
- I will only discuss the solution to the paradoxes using type theory (I will not discuss set theory or category theory).

Why Type Theory?

- To *avoid paradox* Russell controlled function application via *type theory*.
- Russell [15] *1903* gives the first type theory: the *Ramified Type Theory* (RTT). But, *types* existed since the time of *Euclid* (325 B.C.). And Frege did use typing to avoid paradoxes (still the paradoxes sneaked from the backdoor).
- RTT is used in Russell and Whitehead's Principia Mathematica 1910–1912.
- *Simple theory of types* (STT): Ramsey [11] *1926*, Hilbert and Ackermann [7] *1928*.
- Church's *simply typed λ-calculus* $\lambda\!\!\to$ 1940 = λ-calculus + STT.
- Simply typed λ-calculus was adopted in theorem provers like HOL and was used to make sense of other programming languages (e.g., pascal).

- Then, simple types were independently extended to *polymorphic* logic and programming languages.
- *Dependent* types (necessary for reasoning about proofs inside the system) were also introduced in Automath by de Bruijn.
- Polymorphic types are used in programming languages like ML although not the full 2nd order λ-calculus since type Checking and typability in the 2nd order λ-calculus is undecidable (this was an open problem for over 25 years and was shown in 1995 by Joe Wells).
- And the search continues for better and better programming languages.
- *Types* continue to play an influential role in the design and implementation of programming languages and theorem provers.

Russell suggests types to avoid paradoxes

- Logicians considered these paradoxes to be *out of the scope of logic*: The *Liar's Paradox* can be regarded as a problem of *linguistics*. The *paradoxes of Cantor and Burali-Forti* occurred in what was considered in those days a *highly questionable* part of mathematics: Cantor's Set Theory.
- The Russell Paradox, however, was *a paradox that could be formulated in all* the systems that were presented at the end of the 19th century (except for Frege's *Begriffsschrift*). It was at the very basics of logic. It could not be disregarded, and a solution to it had to be found.
- In 1903-1908, Russell suggested the use of *types* to solve the problem [16].

Types and vicious circle principle

- *"In all the above contradictions there is a common characteristic, which we may describe as self-reference or reflexiveness.* [...] *In each contradiction something is said about* all *cases of some kind, and from what is said a new case seems to be generated, which both is and is not of the same kind as the cases of which* all *were concerned in what was said."*

  *(*Mathematical logic as based on the theory of types*)*

- Russell's plan was, *to avoid the paradoxes* by *avoiding all possible self-references*. He postulated the *"vicious circle principle"*:
- "Whatever involves all of a collection *must not be one* of the collection."

  (Mathematical logic as based on the theory of types)

- Russell implements this principle *very strictly* using *types*.

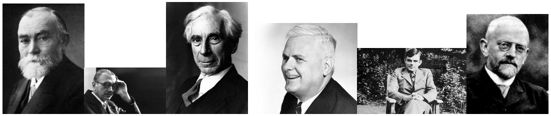
Church's λ-calculus

- Church wanted the λ-calculus to be a theory of *functions* and *logic.*
- The logic part turned out to be inconsistent, so Church restricted the λ-calculus to a theory of functions (Church 1932).
- $A := x \mid AB \mid \lambda x.A$
- This type free λ-calculus is the language of the computable function: f is computable iff f can be written in the type free λ-calculus.
- To incorporate logic, Church added the simple types of Ramsey's STT, giving us the simply typed λ-calculus (Church 1940).
- $\lambda\!\to$ is very restrictive.
- Numbers, booleans, identity, etc., have to be defined at every level.
- We can represent (and type) terms like $\lambda x : o.x$ and $\lambda x : \iota.x$.
- We cannot type $\lambda x : \alpha.x$, where α can be instantiated to any type.
- This led to new (modern) type theories that allow more general notions of functions (e.g, *polymorphic*).

The evolution of functions with Frege, Russell and Church

- Historically, *functions* have been treated as *meta-objects*.
- Function *values* were the important part, not *abstract functions*.
- In the *low level/operational approach* there are only function values.
- The *sine-function*, is always expressed with a value: $\sin(\pi)$, $\sin(x)$ and properties like: $\sin(2x) = 2\sin(x)\cos(x)$.
- In many mathematics courses, one calls $f(x)$—not f—the *function*.
- *Frege*, *Russell* and *Church* wrote $x \mapsto x + 3$ resp. as $x + 3$, $\hat{x} + 3$ and $\lambda x.x + 3$.
- Principia's *functions are based on Frege's Abstraction Principles* but can be first-class citizens. Frege used courses-of-values to speak about functions.
- Church made every function a first-class citizen. This is *rigid* and does not represent the development of logic in 20th century.

Summary



The Challenge

- Is to design calculi that work well for all the features needed for expressive computation languages:
    - Data Types,
    - Inference rules,
    - logic and mathematics,
    - capture-free substitution within a symbolic expression,
- while having a clear syntax, semantics and the desired properties (Correctness, Termination, Computerisation).

Church's Simply Typed λ-calculus in modern notation

- Terms $A ::= x \mid AB \mid \lambda x{:}\sigma.B$
- Types $::= T \mid \mid \sigma \to \tau$
- Γ is an environment (set of declaration).
- Rules:

(start)	$\dfrac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$
(λ)	$\dfrac{\Gamma, x{:}\sigma \vdash A : \tau}{\Gamma \vdash \lambda_{x:\sigma}.A : \sigma \to \tau}$
(app_Π)	$\dfrac{\Gamma \vdash A : \sigma \to \tau \qquad \Gamma \vdash B : \sigma}{\Gamma \vdash AB : \tau}$

Common features of modern types and functions

- We can *construct* a type by abstraction. (Write $A : *$ for *A is a type*)
  - $\lambda_{y:A}.y$, the identity over A *has type* $A \to A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity *has type* $\Pi_{A:*}.A \to A$
- We can *instantiate* types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{y:A}.y)[A := \mathbb{N}]$ *has type* $(A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ *has type* $(\Pi_{A:*}.A \to A)\mathbb{N} = (A \to A)[A := \mathbb{N}]$ or $\mathbb{N} \to \mathbb{N}$.
- $(\lambda x{:}\alpha.A)B \to_\beta A[x := B]$ $\qquad$ $(\Pi x{:}\alpha.A)B \to_\Pi A[x := B]$
- Write $A \to A$ as $\Pi_{y:A}.A$ when y not free in A.

The Barendregt Cube

- Syntax: $A ::= x \mid * \mid \Box \mid AB \mid \lambda x{:}A.B \mid \Pi x{:}A.B$
- Formation rule:

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \qquad \textit{if } (s_1, s_2) \in \boldsymbol{R}$$

The β-cube: $\rightarrow_\beta$ + conv$_\beta$ + app$_\Pi$

(axiom)	$\langle\rangle \vdash * : \Box$
(start)	$\dfrac{\Gamma \vdash A : s \qquad x \notin \textsc{dom}\,(\Gamma)}{\Gamma, x{:}A \vdash x : A}$
(weak)	$\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \quad x \notin \textsc{dom}\,(\Gamma)}{\Gamma, x{:}C \vdash A : B}$
(Π)	$\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2 \quad (s_1, s_2) \in \boldsymbol{R}}{\Gamma \vdash \Pi_{x:A}.B : s_2}$
(λ)	$\dfrac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash \Pi_{x:A}.B : s}{\Gamma \vdash \lambda_{x:A}.b : \Pi_{x:A}.B}$
(conv$_\beta$)	$\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$
(app$_\Pi$)	$\dfrac{\Gamma \vdash F : \Pi_{x:A}.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x{:=}a]}$

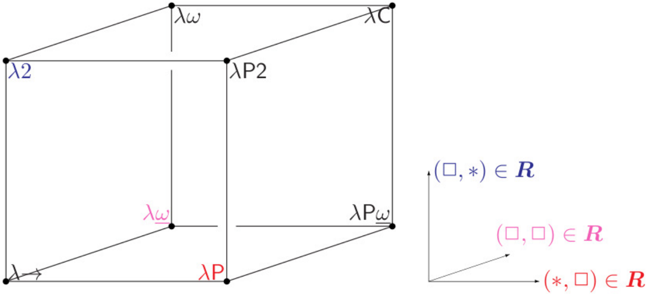
6 desirable properties of a type system with reduction r

- *Types are correct (TC)*
  If $\Gamma \vdash A : B$ then $B \equiv \Box$ or $\Gamma \vdash B : s$ for$s \in \{*, \Box\}$.
- *Subject reduction (SR)* If $\Gamma \vdash A : B$ and $A \twoheadrightarrow_r A'$ then $\Gamma \vdash A' : B$.
- *Preservation of types (PT)* If $\Gamma \vdash A : B$ and $B \twoheadrightarrow_r B'$ then $\Gamma \vdash A : B'$.
- *Strong Normalisation (SN)* If $\Gamma \vdash A : B$ then $\mathrm{SN}_{\rightarrow_r}(A)$ and $\mathrm{SN}_{\rightarrow_r}(B)$.
- *Subterms are typable (STT)* If A is $\vdash$-legal and if C is a sub-term of A then C is $\vdash$-legal.
- *Unicity of types*
  - *(UT1)* If $\Gamma \vdash A_1 : B_1$ and $\Gamma \vdash A_2 : B_2$ and $A_1 =_r A_2$, then $\Gamma \vdash B_1 =_r B_2$.
  - *(UT2)* If $\Gamma \vdash B_1 : s$, $B_1 =_r B_2$ and $\Gamma \vdash A : B_2$ then $\Gamma \vdash B_2 : s$.

	Simple	Poly-morphic	Depend-ent	Constr-uctors	Related system	Refs.
$\lambda\!\rightarrow$	$(*,*)$				λ^τ	[2, 1, 8]
$\lambda 2$	$(*,*)$	$(\Box,*)$			F	[5, 13]
λP	$(*,*)$		$(*,\Box)$		AUT-QE, LF	[4, 6]
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$	POLYREC	[12]
λP2	$(*,*)$	$(\Box,*)$	$(*,\Box)$			[9]
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$	Fω	[5]
λP$\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$		
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$	CC	[3]

The 8 Systems

The Barendregt Cube



$\lambda\omega$
λC
$\lambda 2$
$\lambda P2$
$\lambda\underline{\omega}$
$\lambda P\underline{\omega}$
$\lambda{\to}$
λP
$(\square, *) \in \boldsymbol{R}$
$(\square, \square) \in \boldsymbol{R}$
$(*, \square) \in \boldsymbol{R}$

From Frege's low level functions to PTSs that capture strong normalisation

- Kamareddine and Wells 2017, has incorporated Frege's low level of functions to create PTSs with intersection types which contain all the ordinary PTSs (including the β-cube given above and its extensions with parameters/Frege's functions.
- The f-cube is the β-cube extended with finite set declarations in the form of ordinary mathematical notion of function.
- **Theorem:** If $\Gamma \vdash_f A : B$ then A and B are strongly normalising.
- **Theorem:** If a type free term of the λ-calculus M is strongly Normalising then M is typable in the f-cube.
- Urzyczyn proved $U = (\lambda r.\, h(r(\lambda f \lambda s.\, f\, s))(r(\lambda q.\lambda g.\, g\, q)))(\lambda o.\, o\, o\, o)$ is untypable in Fω. Hence U is untypable in any system of the cube.
- But U is strongly normalising.
- Kamareddine and Wells 2017 prove that U is typable in the f-cube: There are Γ, A such that $\Gamma \vdash_f U : A$.

Conclusion

- A hierarchy of systems that classify important properties of CR, SN, SR.
- Not only types are used to derive important properties and avoid paradoxes and non termination, but also types classify non termination.
- We are far away still from having computer help to prove these properties. It would be nice to have a system that can help us with the proofs.

[1] H[endrik] P[ieter] Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[2] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[3] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Inform. & Comput.*, 76:95–120, 1988.

[4] N.G. de Bruijn. The mathematical language Automath – its usage and some of its extensions. In L. Laudet, D. Lacombe, and M. Schuetzenberger, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61, Heidelberg, 1970. Springer-Verlag. Reprinted in [10, A.2].

[5] J[ean]-Y[ves] Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. Thèse d'Etat, Université de Paris VII, 1972.

[6] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings Second Symposium on Logic in Computer Science*, pages 194–204, Washington D.C., 1987. IEEE.

[7] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Die Grundlehren der Mathematischen Wissenschaften in

Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928.

[8] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

[9] G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. Technical Report CMU-CS-88-131, Carnegie Mellon University, Pittsburgh, USA, 1988.

[10] Rob Nederpelt, J. H. Geuvers, and Roel C. de Vrijer. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1994.

[11] F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society,* 2nd series, 25:338–384, 1926.

[12] G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Information and Computation*, 99:154–177, 1991.

[13] J. C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *LNCS*, pages 408–425. Springer-Verlag, 1974.

[14] B. Russell. Letter to Frege. English translation in [17], pages 124–125, 1902.

[15] B. Russell. *The Principles of Mathematics*. Allen & Unwin, London, 1903.

[16] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [17], pages 150–182.

[17] J. van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, 1967.