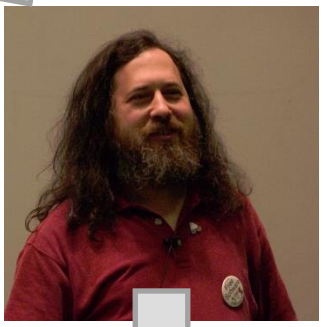
Microsoft Research

Programming Proofs and Proving Programs

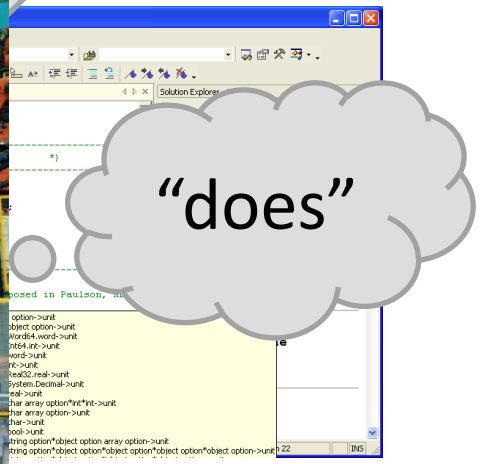
Nick Benton

Microsoft Research, Cambridge

Coffee



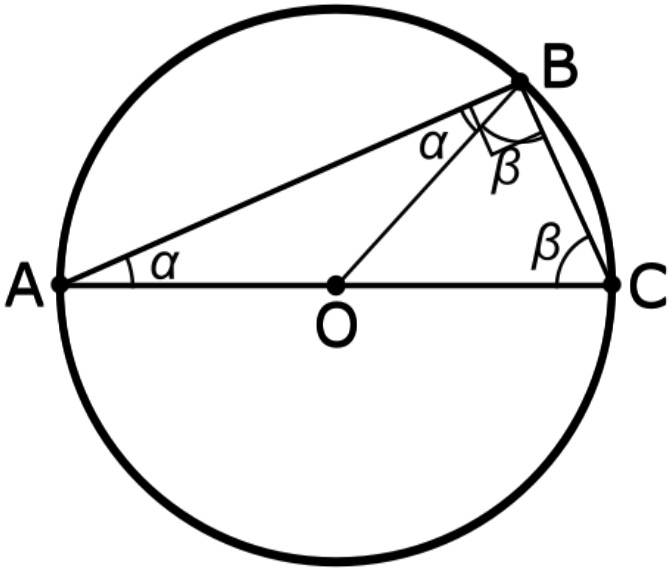
"is"



"does"

Greek

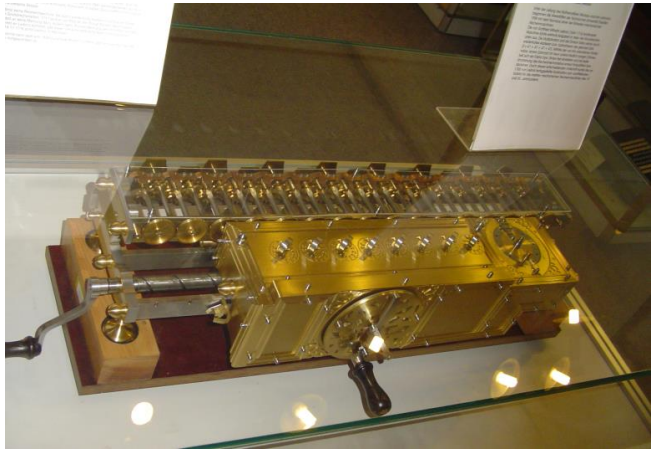
1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any centre and distance.
4. That all right angles are equal to one another.
5. That, if a straight line falling on two straight lines make the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the two right angles.



- All babies are illogical.
- Nobody is despised who can manage a crocodile.
- Illogical persons are despised.

Leibniz

- The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: *calculemus*, without further ado, to see who is right.



Boole

- "Mathematical Analysis of Logic" (1847)
- "An Investigation into the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities" (1854)

PROPOSITION IV.

That axiom of metaphysicians which is termed the principle of contradiction, and which affirms that it is impossible for any being to possess a quality, and at the same time not to possess it, is a consequence of the fundamental law of thought, whose expression is $x^2 = x$.

Let us write this equation in the form

$$x - x^2 = 0,$$

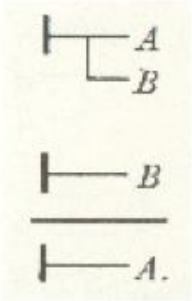
whence we have

$$x\,(1 - x) = 0\,; \qquad (1)$$

both these transformations being justified by the axiomatic laws of combination and transposition (II. 13). Let us, for simplicity

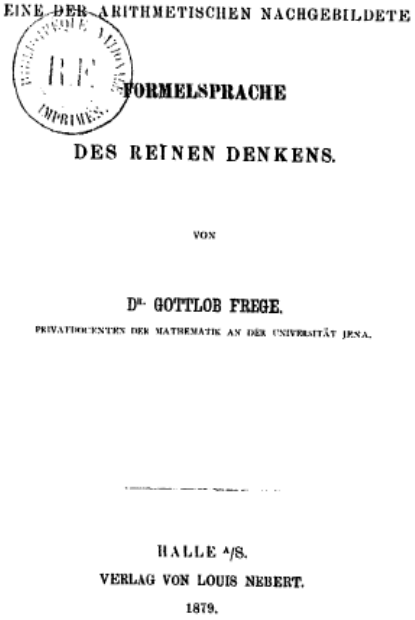
Frege

- "Begriffsschrift" Concept Script (1879)
  - introduced quantifiers, ∀, ∃
  - notation for inferences:



- "Grundgesetze der Arithmetik" Basic Laws Arithmetic (1893,1903)
  - Logicism: arithmetic reduced to logic

> "Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion."



BEGRIFFSSCHRIFT,

EINE DER ARITHMETISCHEN NACHGEBILDETE

FORMELSPRACHE

DES REINEN DENKENS.

VON

Dr. GOTTLOB FREGE,

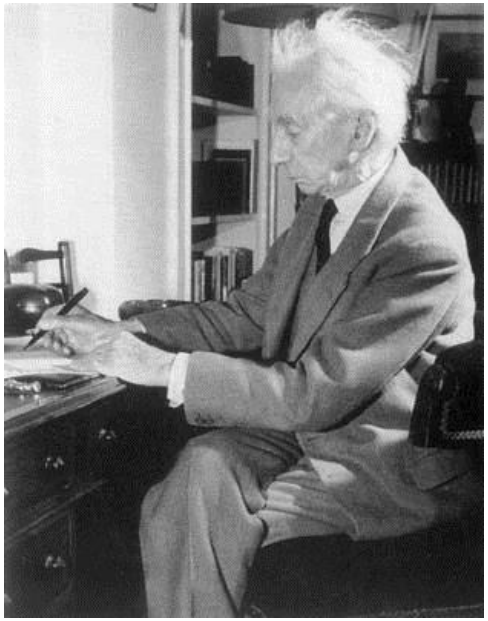
PRIVATDOCENTEN DER MATHEMATIK AN DER UNIVERSITÄT JENA.

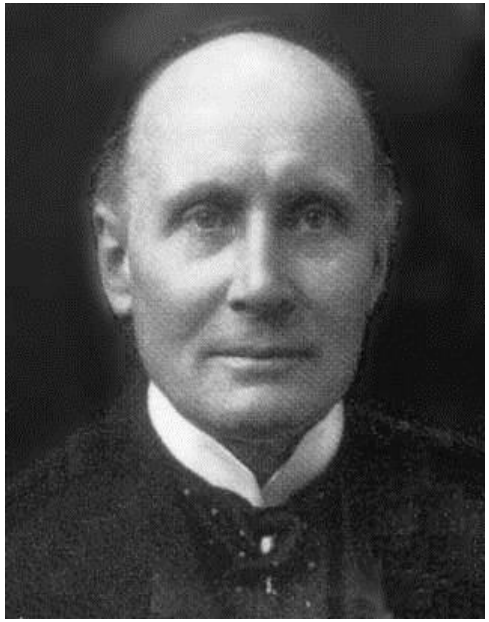
HALLE a/S.

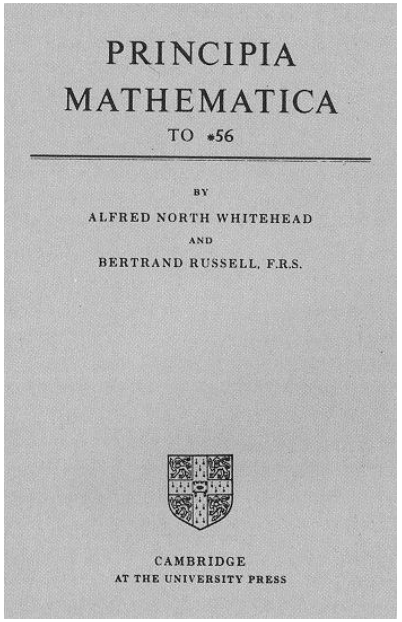
VERLAG VON LOUIS NEBERT.

1879.

Russell







- Russell's paradox showed inconsistency of naive foundations such as Frege's: $\{X \mid X \notin X\}$
  - *"The set of sets which are not members of themselves"*
- Theory of Types and *Principia Mathematica* (1910,1912,1913)
- following the logicist programme, got as far as sets, cardinals, ordinals, reals
- other fix: Zermelo's set theory (Foundation, von Neumann (1925))

$*54{\cdot}43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54{\cdot}26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

[$*51{\cdot}231$] $\equiv . \iota`x \cap \iota`y = \Lambda .$

[$*13{\cdot}12$] $\equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$

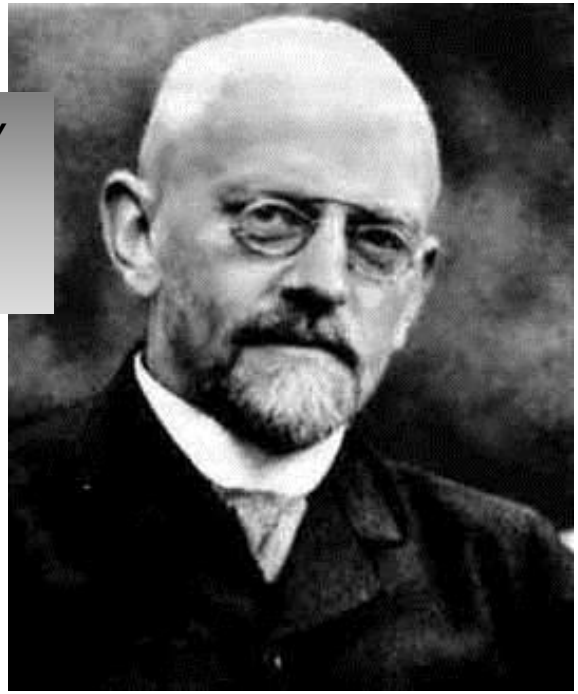
$\vdash :. (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . *11{\cdot}54 . *52{\cdot}1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

(page 379 with proof completed on p86 of volume 2)

Hilbert



There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no ignorabimus.

Wir müssen wissen, wir werden wissen

"The definitive clarification of the nature of the infinite has become necessary, not merely for the special interests of the individual sciences but for the honour of human understanding itself."

• Aimed to reconstitute infinitistic mathematics in terms of a formal system which could be proved (finitistically):

• *Consistent*: It should be impossible to derive a contradiction (such as 1=2).

• *Complete*: All true statements should be provable.

• *Decidable*: There should be a (definite, finitary, terminating) procedure for deciding whether or not an arbitrary statement is provable. (The *Entscheidungsproblem*)

Microsoft Research

Gödel

- "On formally undecidable propositions of *Principia Mathematica* and related systems" (1931)
- One can encode the propositions and rules of inference of a formal system as natural numbers, so that statements about the system become statements about arithmetic.
- Thus, if the system is sufficiently powerful to prove things about arithmetic, it can talk (indirectly) about itself.
- The key idea is then to construct a proposition *P* which, under this interpretation, asserts

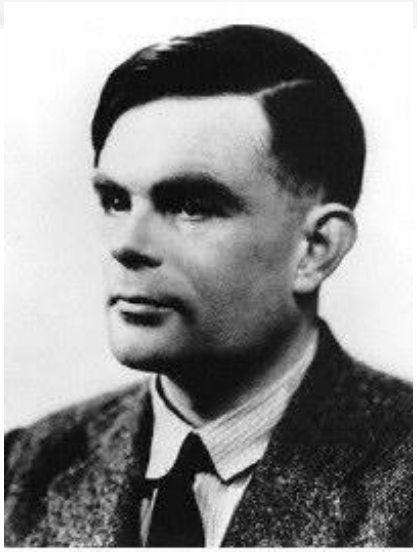
  P is not provable

- Then P must be true (for if P were false, P would be provable and hence, by consistency, true - a contradiction!)
- So P is true and unprovable, i.e. the system is *incomplete*

Turing

Microsoft Research

- “On computable numbers with an application to the Entscheidungsproblem” (1936)
- Introduced the Turing machine, showed undecidability of halting problem
  - By a diagonal argument very like that used by Godel
- Church-Turing thesis
- Fixed point combinator (1937)
- Breaking Enigma at Bletchley (Bombe)
- ACE and Manchester Mk.1



How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
- Checking a Large Routine (1949)

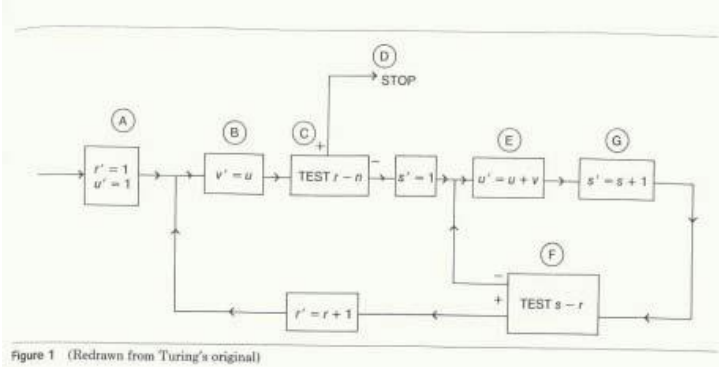


Figure 1 (Redrawn from Turing's original)

STORAGE LOCATION	(INITIAL) Ⓐ $k=6$	Ⓑ $k=5$	Ⓒ $k=4$	(STOP) Ⓓ $k=0$	Ⓔ $k=3$	Ⓕ $k=1$	Ⓖ $k=2$
27 28 29 30 31	n	r n $\lvert r$	r n $\lvert r$ $\lvert r$	n $\lvert n$	s r n $s\lvert r$ $\lvert r$	$s+1$ r n $(s+1)\lvert r$ $\lvert r$	s r n $(s+1)\lvert r$ $\lvert r$
	TO Ⓑ WITH $r'=1$ $u'=1$	TO Ⓒ	TO Ⓓ IF $r=n$ TO Ⓔ IF $r<n$		TO Ⓖ	TO Ⓑ WITH $r'=r+1$ IF $s \geqslant r$ TO Ⓔ WITH $s'=s+1$ IF $s<r$	TO Ⓕ

Figure 2 (Redrawn from Turing's original)

Microsoft Research

Church

- "A Set of Postulates for the Foundation of Logic" (1932,1933)
- Aimed at foundation for logic more natural than Russell's type theory or Zermelo's set theory, taking functions as the basis
  - M,N := x | M N | λx.M
  - (λx.M) N $\rightarrow$ M[N/x]
- Eschewed excluded middle
- Showed how λ-terms could encode arithmetic
- "An unsolvable problem of elementary arithmetic" (1936) showed λ-convertibility undecidable (resolving Entscheidungsproblem and just pipping Turing)
- Church's logic found inconsistent, but the calculus of functions and binding turned out to be rather important

"...there may indeed be uses for the system other than as a logic."

Microsoft Research

Typed Lambda Calculus

- goes back to e.g. Church “A Formulation of the Simple Theory of Types” (1940)
- but this is a modern, programming language-centric version

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.M) : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathrm{in}_1 M : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \mathrm{in}_2 M : A + B}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x_1 : A \vdash N_1 : C \quad \Gamma, x_2 : B \vdash N_2 : C}{\Gamma \vdash \mathrm{case}\ M\ \mathrm{of}\ \mathrm{in}_1 x_1 \Rightarrow N_1 \mid \mathrm{in}_2 x_2 \Rightarrow N_2 : C}$$

Microsoft Research

Gentzen

- Natural deduction (1935)

$$\frac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \supset B}\,(\supset_{\mathcal{I}}) \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ A \supset B & A\end{array}}{B}\,(\supset_{\mathcal{E}})$$

$$\frac{\begin{array}{cc}\vdots & \vdots\\ A & B\end{array}}{A \wedge B}\,(\wedge_{\mathcal{I}}) \qquad \frac{\begin{array}{c}\vdots\\ A \wedge B\end{array}}{A}\,(\wedge_{\mathcal{E}}) \qquad \frac{\begin{array}{c}\vdots\\ A \wedge B\end{array}}{B}\,(\wedge_{\mathcal{E}})$$

$$\frac{\begin{array}{c}\vdots\\ A\end{array}}{A \vee B}\,(\vee_{\mathcal{I}}) \qquad \frac{\begin{array}{c}\vdots\\ B\end{array}}{A \vee B}\,(\vee_{\mathcal{I}}) \qquad \frac{\begin{array}{ccc}\vdots & [A] & [B]\\ A \vee B & \vdots & \vdots\\ & C & C\end{array}}{C}\,(\vee_{\mathcal{E}})$$

$$\frac{\dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \supset B}\,(\supset_{\mathcal{I}}) \qquad \begin{array}{c}\vdots\\ A\end{array}}{B}\,(\supset_{\mathcal{E}}) \quad \longrightarrow_{\beta} \quad \begin{array}{c}\vdots\\ A\\ \vdots\\ B\end{array}$$

Microsoft Research

turn it sideways...

$$\Gamma, \quad A \vdash \quad A$$

$$\frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \quad A \supset B} \qquad \frac{\Gamma \vdash \quad A \supset B \quad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$\frac{\Gamma \vdash \quad A \quad \Gamma \vdash \quad B}{\Gamma \vdash \quad A \wedge B} \qquad \frac{\Gamma \vdash \quad A \wedge B}{\Gamma \vdash \quad A} \qquad \frac{\Gamma \vdash \quad A \wedge B}{\Gamma \vdash \quad B}$$

$$\frac{\Gamma \vdash \quad A}{\Gamma \vdash \quad A \vee B} \qquad \frac{\Gamma \vdash \quad B}{\Gamma \vdash \quad A \vee B}$$

$$\frac{\Gamma \vdash \quad A \vee B \quad \Gamma, \quad A \vdash \quad C \quad \Gamma, \quad B \vdash \quad C}{\Gamma \vdash \quad C}$$

change notation...

$$\Gamma, \quad A \vdash \quad A$$

$$\frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \qquad A \to B} \qquad \frac{\Gamma \vdash \quad A \to B \quad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$\frac{\Gamma \vdash \quad A \quad \Gamma \vdash \quad B}{\Gamma \vdash \qquad A \times B} \qquad \frac{\Gamma \vdash \quad A \times B}{\Gamma \vdash \quad A} \qquad \frac{\Gamma \vdash \quad A \times B}{\Gamma \vdash \quad B}$$

$$\frac{\Gamma \vdash \quad A}{\Gamma \vdash \qquad A + B} \qquad \frac{\Gamma \vdash \quad B}{\Gamma \vdash \qquad A + B}$$

$$\frac{\Gamma \vdash \quad A + B \quad \Gamma, \quad A \vdash \quad C \quad \Gamma, \quad B \vdash \quad C}{\Gamma \vdash \qquad\qquad\qquad\qquad\qquad\qquad C}$$

Microsoft Research

add terms, et voila!

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.M) : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

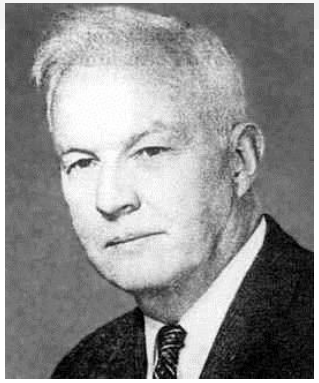
$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathrm{in}_1 M : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \mathrm{in}_2 M : A + B}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x_1 : A \vdash N_1 : C \quad \Gamma, x_2 : B \vdash N_2 : C}{\Gamma \vdash \mathrm{case}\ M\ \mathrm{of}\ \mathrm{in}_1 x_1 \Rightarrow N_1 \mid \mathrm{in}_2 x_2 \Rightarrow N_2 : C}$$

Microsoft Research

Curry - Howard

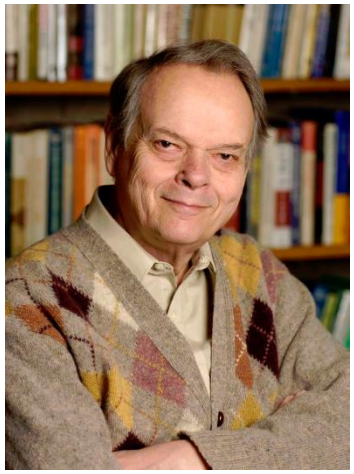


- So we get a correspondence

Constructive Logic	Programming Languages
Propositions	Types
Proofs	Programs
Conjunction $\wedge$	Pairing $\times$
Disjunction $\vee$	Disjoint union +
Implication $\supset$	Function space $\rightarrow$
(Proof normalization)	(Operational semantics)

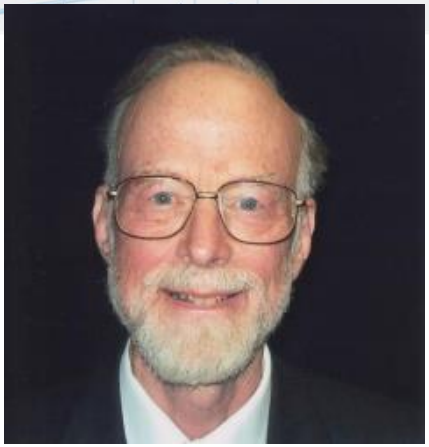
Girard - Reynolds

- Impredicative second order propositional intuitionistic logic aka System F (1971)
- Polymorphic Lambda Calculus (1974)
- Haskell, C# generics,...
- Can encode inductive datatypes using polymorphism
- Second-order existential quantification models abstract datatypes (Mitchell, Plotkin)



Microsoft Research

Hoare



$$\frac{}{\{A\}\texttt{skip}\{A\}} \qquad \frac{}{\{A[ie/X]\}X := ie\{A\}}$$

$$\frac{\{A\}C_1\{A'\} \qquad \{A'\}C_2\{A''\}}{\{A\}C_1\,;C_2\{A''\}} \qquad \frac{\{A \wedge be\}C_1\{A'\} \qquad \{A \wedge \neg be\}C_2\{A'\}}{\{A\}\texttt{if}\ be\ \texttt{then}\ C_1\ \texttt{else}\ C_2\{A'\}}$$

$$\frac{\{A \wedge be\}C\{A\}}{\{A\}\texttt{while}\ be\ \texttt{do}\ C\{A \wedge \neg be\}} \qquad \frac{\models (A \Rightarrow A') \qquad \{A'\}C\{B'\} \qquad \models (B' \Rightarrow B)}{\{A\}C\{B\}}$$

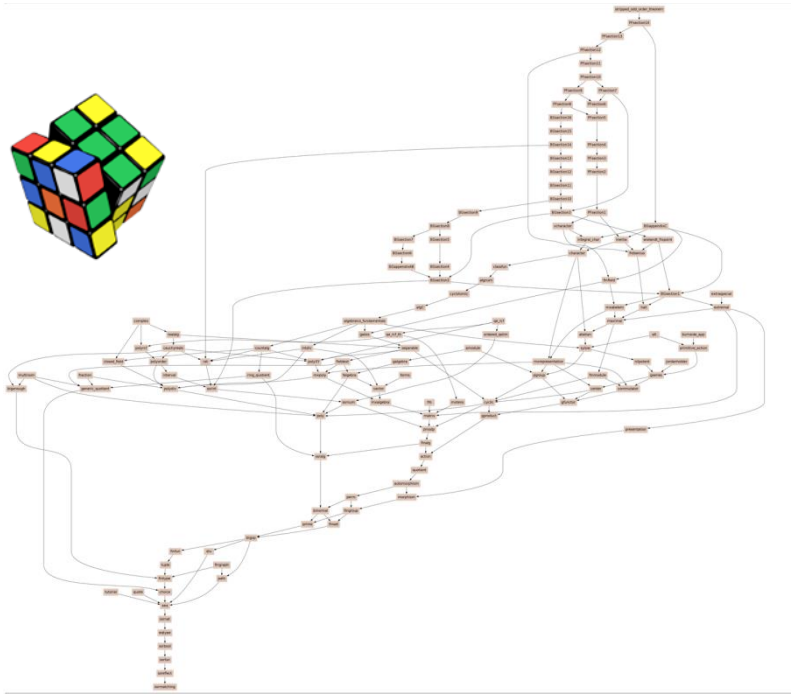
Microsoft Research

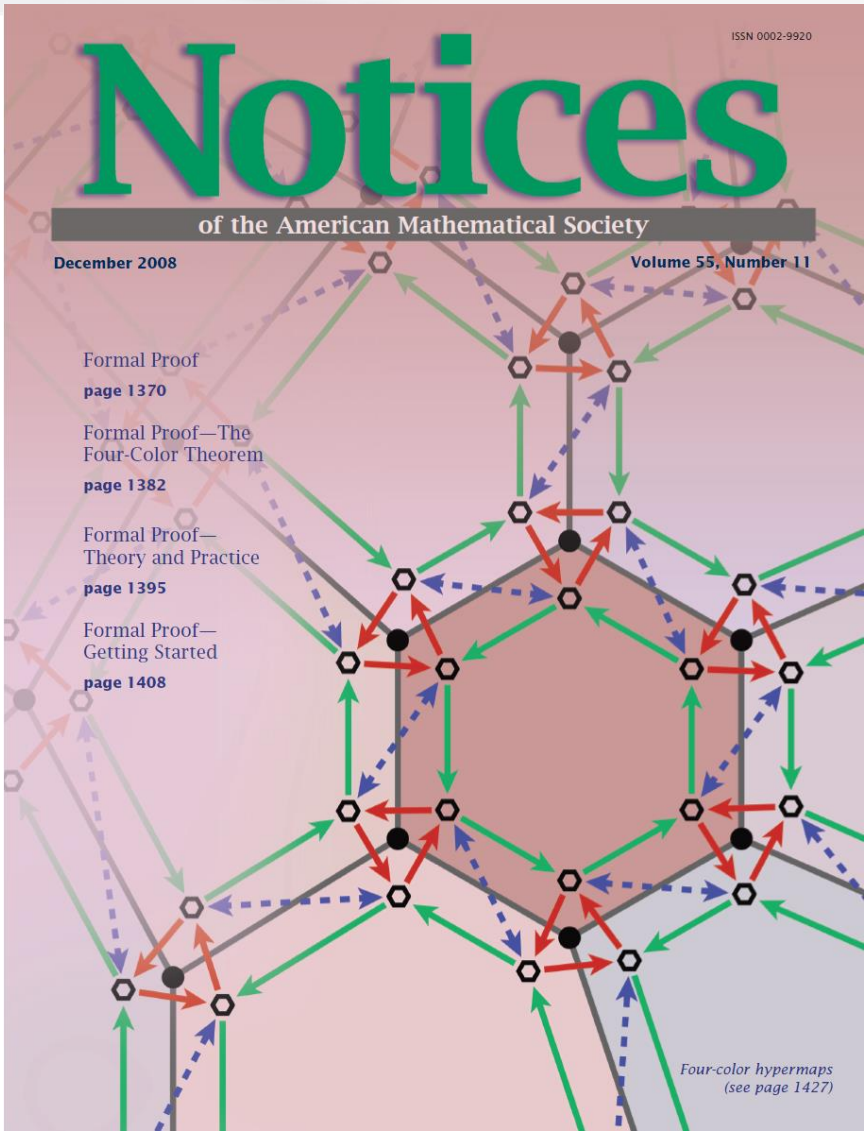
Coquand, Huet

- Calculus of ((co)inductive) constructions (1986)
- Types and terms in a single syntactic structure
- Dependently typed
  - Types can express arbitrary specifications
- Hierarchy of sorts: Prop, Set, Type(i)
- Remarkably simple (though subtle) typing and conversion rules
- Implemented in Coq, a system that is simultaneously
  - A pure functional programming language with an extremely expressive type system
  - A rich place in which to do mathematics (interactively)

Microsoft Research

Gonthier, Hales





Microsoft Research

Explosion of formal verification in PL

- Operating system kernel (sel4, Klein)
- Compiler for C-like language (Leroy)
- Just in our group:
  - Compilers for functional languages
  - Domain theory, concurrency
  - Dimension types for F#
  - Computational cryptography
  - Module systems
  - Refinement type systems
  - Separation logics
  - Foundation of termination analysis
  - …
  - and Simon adds ever more Coq-like features to Haskell

Generating & Proving x86 Code in a Proof Assistant

Language

```
Definition allocImp (heapinfo:DWORD)
      (bytes:nat) (fail:DWORD) :=
  mov ESI, heapinfo;;
  mov EDI, [ESI];;
  add EDI, bytes;;
  jc fail;;  (* wrap-around *)
  cmp [ESI+4], EDI;;
  jc fail;;  (* no memory *)
  mov [ESI], EDI.
```

Specification

```
Definition allocSpec n fail inv code :=
  Forall i, Forall j, (
      safe @ (EIP ~= fail ** EDI?) //\\
      safe @ (EIP ~= j ** Exists p,
                EDI ~= p +# n **
                memAny p (p +# n))
    -->>
      safe @ (EIP ~= i ** EDI?))
    @ (ESI? ** OSZCP_Any ** inv)
    <@ (i -- j :-> code).
```

Logic

```
Lemma spec_at_or_and S R1 R2
     {HNeg: AtContra S}:
  S @ (R1 \\// R2) |-- S @ R1 //\\ S @ R2.
Proof.
  rewrite ->land_is_forall, lor_is_exists.
  transitivity (Forall b,
  S @ (if b then R1 else R2)); last first.
  - apply: lforallR => [[|]].
    - by apply lforallL with true.
    - by apply lforallL with false.
  apply: at_ex'.
Qed.
```

Compiler

```
| SHIFTOP dword op dst ShiftCountCL =>
  encodeOpcode dword #x"D2" $$
  writeNext (inj op, dst)

| IMUL dst src =>
  writeNext #x"0F" $$
  writeNext #x"AF" $$
  writeNext (inj dst, src)
```

```
| COND x c y
let:noblocks
let:yesblock
let (cc,cv)
    (prefix
mov EAX, (ma
cmp EAX, (makeLocalMemSpec frameReg y);
JCC cc cv (nth #0 cmap (size yesblocks).-1
);
jmp (nth #0 cmap (size noblocks).-1)
```

Coq

Proof

```
(* mov [ESI], EDI *)
specintro. move/eqP => Hcarry0.
subst carry0.
specapply MOV_M0R_rule.
- by ssimpl.
rewrite <-spec_reads_frame.
apply limplValid.
autorewrite with push_at.
 apply: landL2. cancel1.
 rewrite /OSZCP_Any /flagAny /regAny
    /allocInv. ssplits.
```

Binary

```
"BB 00 80 0B 00 E9 0B 00 00
 43 01 4F FF C3 FF C3 81 FB
 82 E9 FF FF FF BE 00 80 0B
 00 E9 10 00 00 00 8B 06 89
 00 00 81 C7 04 00 00 00 81
 0F 82 E4 FF FF FF B9 14 00
 00 00 BF D1 06 30 00 C1 E2
 02 03 FA C1 EA 07 C6 04 4F
 00 00 0F 84 07 00 00 00 FF
 00 B9 00 00 00 00 81 FA 31
 07 00 00 00 FF C2 E9 05 00
```

```
Inductive NonSPReg := | EAX | EBX | ECX |
EDX | ESI | EDI | EBP.
(* General purpose registers,
   including ESP *)
Inductive Reg :=
| nonSPReg :> NonSPReg -> Reg
| ESP.
(* All registers, including EIP
   but excluding EFL *)
Inductive AnyReg :=
| regToAnyReg :> Reg -> AnyReg
| EIP.
```

x86 Architecture

```
| MUL src =>
 let! v1 = getRegFromProcState EAX;
 let! v2 = evalRegMem src;
 let res := fullmulB v1 v2 in
 let cfof := high 32 res == #0 in
 do! setRegInProcState EAX (low 32 res);
 do! setRegInProcState EDX (high 32 res);
 do! updateFlagInProcState CF cfof;
 do! updateFlagInProcState OF cfof;
 do! forgetFlagInProcState SF;
 do! forgetFlagInProcState PF;
 forgetFlagInProcState ZF
```

x86 Semantics

Coq is an interactive proof assistant: one can formalize, and have the computer check, arbitrary mathematics (see Georges Gonthier's lecture). It is also a programming language, with a *very* expressive type system.

Starting with operations on bits and words, we build a Coq model of a subset of the x86 ISA, including decoding and execution.

On top of that, we define languages and compilers, such as a macro-assembler. These execute within Coq and the resulting binaries boot on real hardware.

We also define custom specification languages and program logics in Coq; here a form of Hoare logic for heap data and code pointers. The meaning and correctness of the logics are formally proved right down to the machine model.

The correctness of particular programs can then be proved within Coq. This yields end-to-end correctness with the very highest level of assurance.

Conclusion

- We’re making significant progress towards realizing the dreams of
  - Leibniz, Frege & Russell: Fully formalized mathematics
  - Hoare, Scott: Formally specified and verified software
  - and getting powerful, expressive programming languages

Mathematicians think like machines for perfect proofs

Updated 10:30 26 June 2013 by **Jacob Aron**

Read full article

Continue reading page | 1 | 2

It's difficult to get computers to think like humans, so mathematicians are trying the opposite. A proposed mathematical framework forces humans to think more like machines in order to harness the remarkable ability of computers to rapidly check proofs.

The framework provides the possibility of proofs that can't be wrong – and so wouldn't need to be laboriously checked by humans. It could also be the first step towards computers carrying out mathematics by themselves, and perhaps even more advanced forms of artificial intelligence.

Mathematical proofs are becoming so complex that even other mathematicians have a hard time following them.

One solution is to use computer-verified proofs, in which software double-checks each logical step in a proof to ensure its correctness. There's just one problem – the current foundations of mathematics, laid down around a hundred years ago, make it difficult to translate human-written proofs into ones that machines can follow.

Now a team of mathematicians led by Vladimir Voevodsky of the Institute for Advanced Study in Princeton say they have a new, 21st century way to do

Machine-minded mathematicians will rule the future *(Image: Petrovich9/Getty)*