De Bruijn's ideas on the Formalization of Mathematics

Herman Geuvers

Radboud Universiteit Nijmegen & Technische Universiteit Eindhoven

Foundation of Mathematics for Computer-Aided Formalization

Padova, 9-11 January 2013

Dick de Bruijn

July 9, 1918 - February 17, 2012

Checking of Mathematical Proofs with a computer

Around 1970 five new systems / projects / ideas

- **Automath** De Bruijn
- **Nqthm** Boyer Moore (Austin, Texas)
- **LCF** Milner (Stanford; Edinburgh)
- **Mizar** Trybulec (Białystok, Poland)
- **Evidence Algorithm** Glushkov (Kiev, Ukrain)

In parallel (and before that):
Foundational research in type theoy and proof theory: Martin-Löf, Girard, (Howard, Curry, Church; Brouwer, Heyting, Kolmogorov)

Interaction between Foundations of Math and Formalization of Math.

This talk

- De Bruijn's ideas on formalization of mathematics
- Where we are now
- Comments on (minimalist) foundations

The Automath project

AUTOMATH is a language for expressing detailed mathematical thoughts. It is not a programming language, although it has several features in common with existing programming languages. It is defined by a grammar, and every text written according to its rules is claimed to correspond to correct mathematics. It can be used to express a large part of mathematics, and admits many ways of laying the foundations. The rules are such that a computer can be instructed to check whether texts written in the language are correct. These texts are not restricted to proofs of single theorems; they can contain entire mathematical theories, including the rules of inference used in such theories.

AUTOMATH, a language for mathematics, N.G. de Bruijn
TH-Report 68-WSK-05, November 1968

The Automath project

- Formal language
- Express large parts of mathematics, admitting many ways for laying a foundation.
- Computer can check correctness.
- Deduction rules can be part of the text (i.e. need not be part of the system)

The role of proofs in mathematics

1. A proof explains: why? Goal: understanding
2. A proof argues: is it true? Goal: verification, convincing

Notably for (2), computersupport can be helpfull.

the *desirability* of mechanical verification. In a short paper by E.W. Dijkstra on a number of processes that might sometimes block one another, the correctness of the algorithm was explained in a paragraph that ended with the remarkable sentence: "And this, the author believes, completes the proof". Indeed, the argument was a bit intuitive. I took it as a challenge and tried to build a proof that would be acceptable for mathematicians. What I achieved was long and very ugly. It might have been improved by developing efficient lemmas for avoiding the many repetitions in my argument, but I left it as it stood. Instead of improving the proof I got the idea that one should be able to instruct a machine to verify such long and tedious proofs. But of course I have to admit that it will be often more elegant and more efficient to try to streamline such an ugly proof before giving it to a machine.

The (future) role of formalised proofs in mathematics

Some translated quotes from de Bruijn

- One should keep in mind that the framework of formalised mathematics is not the same as mathematics itself.
- A machine that has checked formalised mathematics hasn't understood anything.
- Formalised mathematics is (just a poor) part of mathematics. Nevertheless, through the ages, mathematicians have tried to put down their ideas in such formalisms, free of unclarities or uncertainties.
- The final text is the end of the process of mathematical thought, not the process itself.

The different phases in a mathematical proof

1. finding a proof

   Everything goes: experiment, wild guesses, simplify, ...

   Is not preserved (goes to the paper bin), but crucial for students to learn to do math.

2. writing down a proof

   Contains some explanation why the theorem holds and why the proof is the way it is, but mainly proof-steps that together verify the result.

3. present and communicate a proof

   Explain to others, present in a seminar. Improve, simplify, change, generalise the proof.

Computers can play a major role in (2) and (3)

Why would we believe a proof assistant?

A proof assistant is just another program . . .

To increase the reliability of a PA:

- Describe the rules and the logic of the system.
- A small "kernel". All proofs can be translated to a small number of basic principles. High level steps are defined in terms of smaller ones.

How can we believe a proof assistant?

- Check the checker. Verify the correctness of the PA inside the system itself, or in another system.
- The De Bruijn criterion
  Some PAs generate proof objects that can be checked independently from the system by a simple program that a skeptical user could write him/herself.

Automath

Propositions-as-Types, Proofs-as-Terms

De Bruijn:

- A proposition is not a type, but for any proposition A we have $T(A)$, the type of proofs of A.
- Proofs-as-Terms (PAT) is the crucial novelty.
- Proof-terms are checkable (not necessarily computable . . .)

> An important thing I got from Heyting is the interpretation of a proof of an implication $A \to B$ as a kind of mapping of proofs of A to proofs of B. Later this became one of the motives to treat proof classes as types.

Automath

Propositions-as-Types, Proofs-as-Terms

Isomorphism T between formulas and the types of proofs:

$$\Gamma \vdash_{\text{logic}} \varphi \text{ iff } \overline{\Gamma} \vdash_{\text{type theory}} M : T(\varphi)$$

M codes (as a λ-term) the derivation of φ.

$\overline{\Gamma}$ contains

- declarations $x : A$ of free variables
- assumptions, of the form $y : T(\psi)$
- proven lemmas are definitions, recorded as $y := p : T(\psi)$ (y is a name for the proof p of ψ).

Automath

Propositions-as-Types, Proofs-as-Terms

Isomorphism T between formulas and the types of proofs:

$$\Gamma \vdash_{\text{logic}} \varphi \text{ iff } \overline{\Gamma} \vdash_{\text{type theory}} M : T(\varphi)$$

Consequence:

proof checking $=$ type checking

A simple typing algorithm suffices to satisfy the De Bruijn criterion. Automath systems had a small kernel, so for them the typing algorithm is relatively simple.

Automath

Logical Framework

Automath is a language for dealing with basic mathematical mechanisms like substitution, variable binding, creating and unfolding of definitions etc.

1.2 Properly speaking, the rules of AUTOMATH involve little more than the art of substitution. A text written in AUTOMATH consists of a sequence of lines.

Automath

Logical Framework

A user is free to add the logical rules that he/she wishes
$\Rightarrow$ Automath is a logical framework, where the user can do his/her own favourite logic (or any other formal system).

> Automath is a big restaurant where one can eat in any style. Those who want to eat kosher, can do that, but do not force others to do the same. Only intolerant people will be upset by the fact that there is room for people with a different opinion.

Pluralism!
Which should not lead to pillarisation ...

Automath

Logical Framework

De Bruijn's version of the proof-as-terms principle:

$$\Gamma \vdash_L \varphi \text{ iff } \Gamma_L, \overline{\Gamma} \vdash_{\text{type theorie}} M : T(\varphi)$$

where L is a logic, Γ_L is the context in which the constructions of the logic L are declared.

Choice: Which logical constructions do you put in the type theory and which constructions do you declare axiomatically in the context?

De Bruijn: Keep the framework as weak as possible (" A plea for weaker frameworks")

Philosofical implications of Automath

On Platonism

Before I started Automath I was slightly anti-platonistic. That is to say that I always sympathized with Kronecker's statement that only the natural numbers *really* exist. But building Automath I rapidly concluded that I *had* to be anti-platonistic. Before that, I had allowed myself to be a bit confused by statements like "3 is not a number, it is the *name* of a number". But if you have to talk to a machine that knows nothing about the real mathematical objects, then you know that you can handle names only. The "real" mathematical objects are irrelevant in the discussion with the machine. In the language used for communicating with

De Bruijn loves to cite Wittgenstein:

Don't ask for the meaning, ask for its use

Philosofical implications of Automath

constructivism vs. formalism

Remarkably, at that occasion Scott wrote: "de Bruijn had been, of course, personally influenced by Brouwer and wanted to present a suitably constructive notion of proof". I think this is confusing. Not just because I was never influenced by Brouwer's talking or writing, but because "constructive notion of proof" is different from "notion of constructive proof". The latter can be connected to Brouwer, but the former is much closer to Hilbert and his finitistic game with symbols and rigid rules. Brouwer's constructivity is (at least in Heyting's formalized form) is a matter of the *content* of axioms, and not the way these axioms are manipulated.

Beyond Automath

Constructive Type Theory

Formulas-als-types isomorphism translates proofs in constructive logic to typed λ-terms, seen as functional programs
Martin-Löf:

- foundations for mathematics
- inductive types and functions defined by wefounded recursion are the basic principles
- computational content of proofs

Proof Assistants based on CTT (and also on LCF and Automath): Nuprl (Constable, Cornell), Agda (Gothenburg), Coq (INRIA, France)

Present state of affairs

Is formalising proofs as simple as using LaTeX?

As a kind of dream I played (in 1968) with the idea of a future where every mathematician would have a machine on his desk, all for himself, on which he would write mathematics and which would verify his work. But, by lack of experience in such matters, I expected that such machines would be available within 5 years from then. But now, 23 years later, we are not that far yet. Anyway I expected in 1968 that the memory capacity of main frame computers would grow rapidly in the next few years, but that was a deception too. Implementing Automath on the quite advanced computers available to us in the years 1970–1975 was to a large extent a struggle for living with the limitations of fast accessible memory.

Question Why do mathematicians and engineers all use Computer Algebra systems and LaTeX, but not Proof Assistants?

Present state of affairs

Big formalisations

- Proof of the Odd Order Theorem (Walter Feit and John G. Thompson), completely machine-checked using Coq. Mathematical Components team lead by Georges Gonthier (MSR Cambridge) at the Inria Microsoft Research Joint Centre.
- Flyspeck Project to produce a formal proof of the Kepler Conjecture. Thomas Hales et al. in HOL-light.

Present state of affairs

Freek Wiedijk: The 100 greatest theorems, 88 formalised

1.	The Irrationality of the Square Root of 2	≥ 17
2.	Fundamental Theorem of Algebra	4
3.	The Denumerability of the Rational Numbers	6
4.	Pythagorean Theorem	6
5.	Prime Number Theorem	2
6.	Gödel's Incompleteness Theorem	3
7.	Law of Quadratic Reciprocity	4
8.	The Impossibility of Trisecting the Angle and Doubling the Cube	1
9.	The Area of a Circle	1
10.	Euler's Generalization of Fermat's Little Theorem	4
11.	The Infinitude of Primes	6
12.	The Independence of the Parallel Postulate	0
13.	Polyhedron Formula	1

...

google: 100 theorems

The best proof assistants . . .

five systems seriously used for matematics:

HOL	HOL Light	86
	ProofPower	42
	Isabelle	49
	Coq	49
	Mizar	57

Some comments on (minimalist) foundation

Putting more semantics into the type theory

Partial terms in Proof Assistants

Four possible approaches (J. Harrison)

1. Give each partial function a convenient value on points outside its domain. (ACL2; Mizar, HOL, Isabelle)
2. Give each partial function some arbitrary value outside its domain. (Coq, Mizar, HOL, Isabelle)
3. Encode the domain of the partial function in its type and make its application to arguments outside that domain a type error. (PVS, Coq, Nuprl)
4. Have a true logic of partial terms. (IMPS)

Putting more semantics into the type theory

Partial terms in Proof Assistants

Is $1/-$ a total function on IR? How to treat $1/0$?

- Coq standard library: $1/-$ is total . . . but then one cannot make a model of IR inside Coq.
- CoRN: $1/-$ needs a proof, $1/- : \Pi x : IR.x\#0 \to IR$ . . . , now one can make a model of IR in Coq, but we cannot even write down $1/0$.

Add domain conditions D, e.g. $D(x/y) = y \neq 0$

- First order logic with domain conditions, F. Wiedijk J. Zwanenburg, TPHOL 2003
- A Partial Functions Version of Church's Simple Theory of Types William M. Farmer, JSL 55 1990.

Comments on (minimalist) foundation I

More computation in the system

- Inductive types and (well-founded) recursive functions turn a PA (Coq, Matita, Agda, Nuprl, . . .) into a programming language.
- This allows programming automated theorem proving techniques inside the system. (Via Reflection)
- When the power of this was first shown to Per Martin-Löf (Kloster Irsee 1998), he strongly opposed to this . . . "These aren't proofs!"

Reflection

Trading in proofs for computations

- Say we have a class of problems P, that we can represent by the inductive type `Problem`. So we have $[\![p]\!] : \texttt{Prop}$ for $p : \texttt{Problem}$.
- Say we can also write a problem solver for `Problem`, that is $\texttt{solve} : \texttt{Problem} \to \texttt{Bool}$
- that we can prove correct:

$$\forall p : \texttt{Problem}, [\![p]\!] \longleftrightarrow \texttt{solve}\, p = \texttt{true}.$$

- Then we can replace a goal $? : A$ for $A \in \mathcal{P}$ by a computation $\texttt{solve}\, a$, if $[\![a]\!] = A$.
  M. Oostdijk, H.G. *Proof by Computation in Coq*, TCS 2001

Even more computation in the system?

Unbounded proof search using a fixed point combinator

- Say P is a decidable property over the natural numbers: $\forall x : N, P\, x = \texttt{true} \vee P\, x = \texttt{false}$. We want to prove $\exists x : N, P\, x$ by an unbounded search.
- For $f : N \to \exists x : N, P\, x$ and $n : N$, define

$$F\ f\ n := \texttt{if}\, P\ n\ \texttt{then}\ \langle n, \texttt{refl} \rangle\ \texttt{else}\ f\ (n + 1)$$

- Take $f := Y\ F$ (so f is a fixed-point of F).
- Now $f : N \to \exists x : N, P\, x$

Even more computation in the system?

Unbounded proof search using a fixed point combinator

- $f\ 0$ evaluates to
  - $\langle n, \texttt{refl} \rangle$ where n is the first number for which $P\ n$ holds (if such n exists)
  - nothings [runs forever] (if no such n exists).
- Theorem: adding Y to a type system in this way is conservative, i.e. in case $f\ 0$ terminates, it indeed finds a solution in the original system (without Y).

H.G., E. Poll, J. Zwanenburg, Safe Proof Checking in Type Theory with Y, CSL 1999.

Back to simple (linear time?) typechecking?

Storing a trace of the conversion in the proof-term

- $\lambda H :=$ the type theory λP with the following special rules

$$\frac{\Gamma \vdash t : A \quad H : A = B}{\Gamma \vdash t^H : B}\ \text{conversion} \qquad \frac{}{\epsilon(a) : a = |a|}$$

- We construct an expression H to record the *conversion trace* between A and B, $H : A = B$. This is just the usual $\beta(\delta\iota\zeta)$-path extended with an erasure step.

- In λH, type-checking is linear

H.G., F. Wiedijk A logical framework with explicit conversions. ENTCS 199 2008

Compatibility with HOL

Present day PAs

- The kernel of a PA is not small, except for HOL-light.
- HOL is much easier to explain to / convince mathematicians of then inductive types
  Any minimalist foundation should be compatible with (classical) HOL.

Compatibility with HOL

Are Set and Prop the same?

- `Prop` and `Set` should be distinct.
- HOL + `Prop`=`Set`is not conservative over HOL. (H.G. 1989, S. Berardi 1989)
- To put it more clearly (S. Berardi 1989):

$$\mathrm{HOL} + \texttt{Prop} = \texttt{Set} + \mathrm{EXT} + \mathrm{Arithmetic} \vdash \bot$$

$\mathrm{EXT} := \forall A, B : \texttt{Prop}, (A \leftrightarrow B) \rightarrow A = B.$

Foundations of Mathematics for Computer-Aided Formalization

A number of issues

- If LF is the system to use, or do we need a more foundational approach?
- Do we want to formalize what mathematicians do or do we want to change what mathematicians do?
  Hendrik Lenstra: Why avoid LEM? Then you can prove less theorems!? I want to prove more theorems!

Questions?