Computer Assisted Mathematical Proofs: using the computer to verify computers

Herman Geuvers

Radboud University Nijmegen

February 4, 2019
Huygens Colloquium
Science Faculty

What research I do

- Theoretical Computer Science/ Logic in Compuer Science
- Type Theory for programming and verification
- Proof Assistants and Formalizing Mathematics

Can the computer *really* help us to prove theorems?

Yes it can,
and we will rely more and more on computers for correct proofs

But it's hard ...

What are Proof Assistants – History

John McCarthy (1927 – 2011)
1961, Computer Programs for Checking Mathematical Proofs

> *Proof-checking by computer may be as important as proof generation. It is part of the definition of formal system that proofs be machine checkable.*
>
> *...*
>
> *For example, instead of trying out computer programs on test cases until they are debugged, one should prove that they have the desired properties.*

What are Proof Assistants – History

Around 1970 five new systems / projects / ideas for a

Computer system for interactively writing and automatically checking proofs

Nowadays: “Proof assistant” or “Interactive Theorem Prover”

- **Automath** De Bruijn (Eindhoven) now: Coq, Agda
- **Nqthm** Boyer, Moore (Austin, Texas) now: ACL2, PVS
- **LCF** Milner (Stanford; Edinburgh) now: HOL, Isabelle
- **Mizar** Trybulec (Białystok, Poland)
- **Evidence Algorithm** Glushkov (Kiev, Oekrain)

Why not automate this process completely?

Automated Theorem Proving

- For well-understood domains, fully automated theorem proving is possible (but often unfeasible).
- Any interesting fragment of logic is undecidable. (You can prove that you cannot write an algorithm that checks the validity of a statement.)

Proof Assistants: what are they used for

- Verify mathematical theorems
  Some mathematical proofs just become too large and complex: proof of the Kepler conjecture Flyspeck project

- Build up a formal mathematical library
  Mizar Mathematical Library

- Verify software and hardware design
  Safety critical systems are too complex and vital
  Compcert: verified C compiler

Why would we believe a proof assistant?

. . . a proof assistant is just another program . . .

To attain the utmost level of reliability:

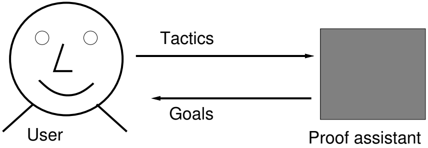
- Description of the rules and the logic of the system.
- A small "kernel". All proofs can be reduced to a small number of basic proof steps. high level steps are defined in terms of the small ones.

Why would we believe a proof assistant?

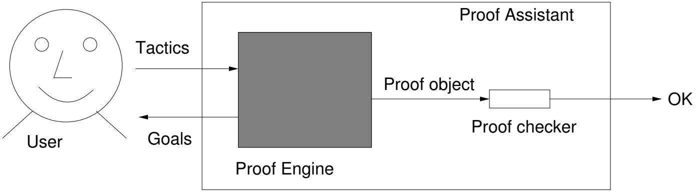
The De Bruijn criterion

⇒ Separate the proof checker ("simple") from the proof engine ("powerful")

Proof Assistant (Interactive Theorem Prover)



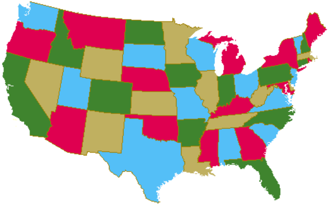
Proof Assistant with a small kernel that satisfies the De Bruijn criterion



Mathematical users of Proof Assistants

The 4 colour theorem

Kenneth Appel en Wolfgang Haken, 1976
Neil Robertson e.a., 1996
Coq: Georges Gonthier, 2004

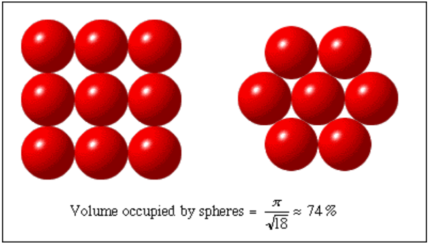


Can *every* map be coloured with only 4 different colours?

• Gonthier has two pages of Coq definitions and notations that are all that's needed to fully and precisely understand his statement of the 4 colour theorem.

Kepler Conjecture (1611)

The most compact way of stacking balls of the same size is a pyramid.



Kepler Conjecture (1611)

- Hales 1998: proof of the conjecture using computer programs (300 pages)

Thomas Hales, associate professor of mathematics, demonstrates his solution to the Kepler conjecture, a problem that mathematicians have been wrestling with since 1611. Tennis balls courtesy of the Varsity Tennis Club. Photo by Bob Kalmbach

- Annals of Mathematics: 99% correct . . . but we can't verify the correctness of the computer programs.

Hales' proof of the Kepler conjecture

Reduce the problem to the verification of inequalities of the shape

$$\frac{\begin{gathered}-x_1x_3 - x_2x_4 + x_1x_5 + x_3x_6 - x_5x_6 + \\ x_2(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6)\end{gathered}}{\sqrt{4x_2\left(\begin{gathered}x_2x_4(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6) + \\ x_1x_5(x_2 - x_1 + x_3 + x_4 - x_5 + x_6) + \\ x_3x_6(x_2 + x_1 - x_3 + x_4 + x_5 - x_6) \\ -x_1x_3x_4 - x_2x_3x_5 - x_2x_1x_6 - x_4x_5x_6\end{gathered}\right)}} < \tan\left(\frac{\pi}{2} - 0.74\right)$$

Use computer programs to verify these inequalities.

Flyspeck project: Computer checked proof of the Kepler conjecture

The formal proof of Hales consists of a number of steps where computer assistance was essential:

a. *A program that lists all 19.715 "tame graphs", that potentially may produce a counterexample to the Kepler conjecture.*
   This program was originally written in Java. Now, it is written and verified in Isabelle.

b. *A computer calculation that verifies that a list of 43.078 linear programs are unsolvable.*
   Each linear program in this list has about 100 variables and a similar list of equations.

c. *A computer verification that 23.242 non-linear equations with at most 6 variables hold.*
   This is the verification where originally interval-arithmetic was used.

Computer Science users of Proof Assistants

Compcert (Leroy et al.)

- verifying an optimizing compiler from C to x86/ARM/PowerPC code
- implemented using Coq's functional language
- verified using using Coq's proof language

Xavier Leroy

why?

- your high level program may be correct, maybe you've proved it correct ...
- ... but what if it is compiled to wrong code?
- compilers do a lot of optimizations: switch instructions, remove dead code, re-arrange loops, ...
- for critical software the possibility of miscompilation is an issue

Compcert

C-compilers are generally not correct

Csmith project *Finding and Understanding Bugs in C Compilers*, X. Yang, Y. Chen, E. Eide, J. Regehr, University of Utah.

> *... we have found and reported more than 325 bugs in mainstream C compilers including GCC, LLVM, and commercial tools.*
> *Every compiler that we have tested, including several that are routinely used to compile safety-critical embedded systems, has been crashed and also shown to silently miscompile valid inputs.*
>
> *As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task.*

other large formalization projects in Computer Science

- formalization of the C standard in Coq
  Krebbers and Wiedijk, Nijmegen 2015.

Robbert Krebbers

- the ARM microprocessor
  proved correct in HOL4
  Anthony Fox University of Cambridge, 2002

- the L4 operating system,
  proved correct in Isabelle
  Gerwin Klein NICTA, Australia, 2009
  200,000 lines of Isabelle
  20 person-years for the correctness proof
  160 bugs before verification
  0 bugs after verification

Gerwin Klein

- Conference Interactive Theorem Proving, every paper is supported by a formalization

Proof Assistants: What needs to be done

Automation

- Formalize all of the Bachelor undergraduate mathematics
- Domain Specific Tactics and Automation
- Combination of Theorem Proving and Machine Learning

AI for Formal Mathematics

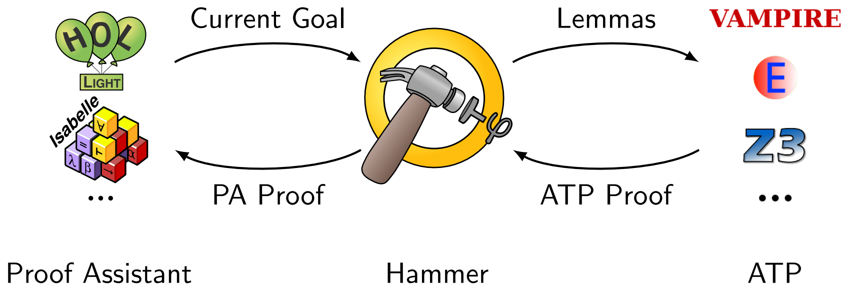
Inductive/Deductive AI over Formal Mathematics

- Alan Turing, 1950: *Computing machinery and intelligence*
- beginning of AI, Turing test
- last section of Turing's paper: *Learning Machines*
- Which intellectual fields to use for building AI?
  - *But which are the best ones [fields] to start [learning on] with?*
  - ...
  - *Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess, would be best.*
- New approach in the last decade:
  - Let's develop AI on large formal mathematical libraries!

Why AI on large formal mathematical libraries?

- Hundreds of thousands of proofs developed over centuries
- Thousands of definitions/theories encoding our abstract knowledge
- All of it completely understandable to computers (*formality*)
- solid semantics: set/type theory
- built by safe (conservative) definitional extensions
- unlike in other "semantic" fields, inconsistencies are not an issue, because in the end every proof is checked

The "Hammer" approach (Urban, Kaliszyk, Blanchette, ...)



- Based on current goal G and repository: select set L of potentially useful lemmas from the repository. Machine Learning
- Send G and L to an ATP. Automated theorem proving
- Let the ATP check if G follows from L and let it produce an ATP-proof.
  (ATP-proof $\simeq$ subset M of L that is really used to prove G)
- Let the (weak) automation inside the proof assistant construct a complete formally checked proof, using M.

Questions?