Computer Assisted Mathematical Proofs: Improving Automation using Machine Learning

Herman Geuvers[1]

Radboud University Nijmegen
The Netherlands

November 18, 2016
Chinese Academy of Science, Beijing

[1] Thanks to Freek Wiedijk, Josef Urban, Cezary Kaliszyk

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 ...

- How does it work?
- Some state of the art
- What needs to be done: proof automation

Overview

- What are Proof Assistants?
- How can a computer program guarantee correctness?
- Challenges

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

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

HOL Light

LCF tradition (Milner):
LCF $\rightarrow$ HOL $\rightarrow$ HOL Light
Stanford, US $\rightarrow$ Cambridge, UK $\rightarrow$ Portland, US
Based on: higher order logic

John Harrison
proves correctness of floating point hardware at Intel
formalises mathematics in his spare time

very simple and elegant system
easy to extend (add your own tactics)
not user friendly

Isabelle

'successor' of HOL
Based on: higher order logic

cooperation between two universities:
Cambridge, UK
focus: computer security
München, Germany
focus: mathematics and programming languages

balanced system
nice proof language
quite powerful automation

Coq

Based on: type theory

INRIA en Microsoft
Institut National de Recherche en Informatique et en Automatique

system with the most impressive formalisation so far
system used most at Nijmegen

integrated programming language
$\approx$ Haskell

mathematically expressive
the built in logic is **intuïtionistic**

Mizar

Andrzej Trybulec
Białystok, Poland

also: Nagano, Japan
Based on: set theory

most mathematical of all proof assistants

largest library of formalised mathematics
2,1 miljon lines of code

user friendly
sometimes hard to follow

What Proof Assistants are not

Doing mathematics on a computer

- **Computing:** *numbers*
  numerical mathematics, visualisation, simulation
- **Computing:** *formulas*
  computer algebra
- **Proving:** *by the computer*
  automatic theorem proving
- **Proving:** *by a human*, with the aid of a computer
  proof assistant

Why Proof Assistants

Doing mathematics on a computer

- **Numerical Mathematics** and **Computer Algebra:** No proofs
- **Automated Theorem Provers:** No interesting mathematics
- **Proof Assistants:** proofs and interesting mathematics

  the price to pay:
  user has to do a lot

  proof assistant = **interactive theorem prover**
  interplay between human and computer

Proof Assistants: what are they used for

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

Proof Assistants for software verification

Holy Grail

> *'Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.'*

Bill Gates, 18 april 2002

Different styles of formalised proofs

- procedural
  tell **what to do**
  Go out of the train, to the right, down the stairs, to the right, out of the exit, to the right, cross the pedestrian crossing, take the metro line 10, ...
- declarative
  tell **where to go**
  Go to the platform, go to the north exit of the station, go to the mero, then go to Peking University, ...

Different styles of formalised proofs

procedural (tactics)

```
Theorem double_div2 : forall (n:nat), div2 (double n) = n.
simple induction n; auto with arith.
intros n0 H.
rewrite double_S; pattern n0 at 2; rewrite <- H; simpl; auto.
Qed.
```

Different styles of formalised proofs

declarative

```
Theorem double_div2 : forall (n:nat), div2 (double n) = n.
proof.
 assume n:nat.
   per induction on n.
     suppose it is 0.
       thus thesis.
     suppose it is (S m) and IH:thesis for m.
       have (div2 (double (S m))= div2 (S (S (double m)))).
            ~= (S (div2 (double m))).
       thus ~= (S m) by IH.
   end induction.
end proof.
```

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.

LCF approach [Milner]:
Have an abstract data type of theorems `thm`, where the only constants of this data type are the axioms and the only functions to this data type are the inference rules of the logic.

Why would we believe a proof assistant?

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

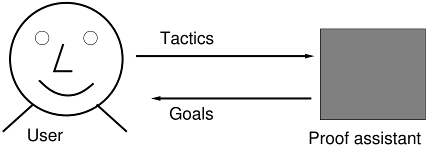
Other possibilities to increase the reliability of the proof assistant

- Check the proof checker. Verify the correctness of the proof assistant in a proof assistant (e.g. the system itself).
  Example Coq in Coq: Construct a model of *Coq* in *Coq* itself and show that all tactics are sound with respect to this model
  NB. Gödel's incompleteness ..., so we need to assume something.
- The De Bruijn criterion
  A proof assistant satisfies the D.B. criterion if it generates proof objects that can be checked independently of the system that created it using a simple program that a skeptical user can write him/herself.

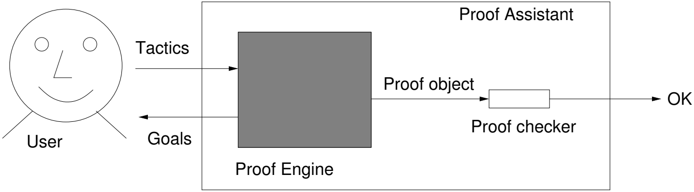
Why would we believe a proof assistant?

Separating the proof checker ("simple") from the proof engine ("powerful")

Proof Assistant (Interactive Theorem Prover)



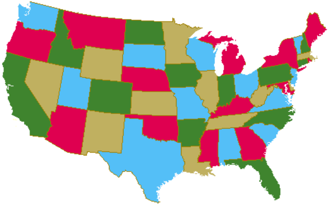
Proof Assistant with a small kernel that satisfies the De Bruijn criterion



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.

Mathematical users of Proof Assistants

Flyspeck project: Formalizing a proof of the Kepler Conjecture

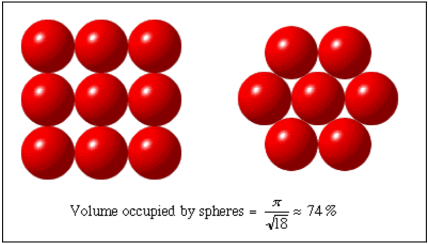
---

http://code.google.com/p/flyspeck/

Tom Hales, CMU Pittsburgh

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

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

Use computer programs to verify these inequalities.

Flyspeck project

- Hales: formalise the proof of Kepler's conjecture using Proof Assistants Write the computer code in the PA, prove it correct in the PA and run it in the PA.
- Proof Assistants used: HOL-light, Isabelle, (Coq)

Essential Computer Assistance in the Flyspeck formal proof

The proof of Hales rests on a number of computer calculations:

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 and exported to ML.

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. INRIA 2006)

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

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, ...

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.*

Some other large formalization projects in Computer Science

- Formalization of the C standard in Coq, by Krebbers and Wiedijk, Nijmegen 2015.
- the ARM microprocessor, proved correct in HOL4 by Anthony Fox University of Cambridge, 2002
- the L4 operating system, proved correct in Isabelle by 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
- 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
- Combination of Theorem Proving and Machine Learning (Urban, Kaliszyk et al.)
  Use ML to produce a hint databse that can be fed to an Automated Theorem Prover: the Hammer approach
- Domain Specific Tactics and Automation

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 (Urban, Kaliszyk and others):
  - Let's develop AI on large formal mathematical libraries!

Why AI on large formal mathematical libraries?

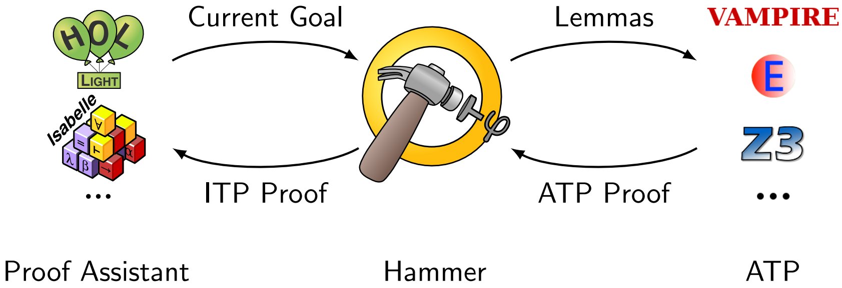
- Hundreds of thousands of proofs developed over centuries
- Thousands of definitions/theories encoding our abstract knowledge
- All of it completely understandable to computers (*formality*)
- solid semantics: set/type theory
- built by safe (conservative) definitional extensions
- unlike in other “semantic”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

Deduction and induction over large formal libraries

Large formal libraries allow:

- strong deductive methods – Automated Theorem Proving
- inductive methods like Machine Learning (the libraries are large)
- combinations of deduction and learning
- examples of positive deduction-induction feedback loops:
  solve problems $\rightarrow$ learn from solutions $\rightarrow$ solve more problems …

The “Hammer” approach



- Based on the Current Goal G and the 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 an ITP-proof, using M.

Premise Selection

Premise selection

Intuition

Given:

- set of theorems T (together with proofs)
- conjecture c

Find: minimal subset of T that can be used to prove c

More formally

$$\arg\min_{t \subseteq T} \{|t| \mid t \vdash c\}$$

In machine learning terminology

Multi-label classification

Input: set of samples $\mathbb{S}$, where samples are triples $s, F(s), L(s)$

- s is the sample ID
- $F(s)$ is the set of features of s
- $L(s)$ is the set of labels of s

Output: function f that predicts list of n labels (sorted by relevance) for set of features

Sample add_comm ($a + b = b + a$) could have:

- F(add_comm) = {"+", "=", "num"}
- L(add_comm) = {num_induct, add_0, add_suc, add_def}

Not exactly the usual machine learning problem

Observations

- Labels correspond to premises and samples to theorems
  - Very often same
- Similar theorems are likely to have similar premises
- A theorem may have a similar theorem as a premise
- Theorems sharing logical features are similar
- Theorems sharing rare features are very similar
- Fewer premises = they are more important
- Recently considered theorems and premises are important

Not exactly for the usual machine learning tools

Classifier requirements

- Multi-label output
  - Often asked for 1000 or more most relevant lemmas
- Efficient update
  - Learning time + prediction time small
  - User will not wait more than 10–30 sec for all phases
- Large numbers of features
  - Complicated feature relations

k-NN

Standard k-NN

Given set of samples $\mathbb{S}$ and features $\vec{f}$

1. For each $s \in \mathbb{S}$, calculate distance $d'(\vec{f}, s) = \|\vec{f} - \vec{F}(s)\|$
2. Take k samples with smallest distance, and return their labels

Feature weighting for k-NN: IDF

- If a symbol occurs in all formulas, it is boring (redundant)
- A rare feature (symbol, term) is much more informative than a frequent symbol
- IDF: Inverse Document Frequency:
- Features weighted by the logarithm of their inverse frequency

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

- This helps a lot in natural language processing
- Smoothed IDF also helps:

$$\mathrm{IDF}_1(t, D) = \frac{1}{1 + |\{d \in D : t \in d\}|}$$

Features

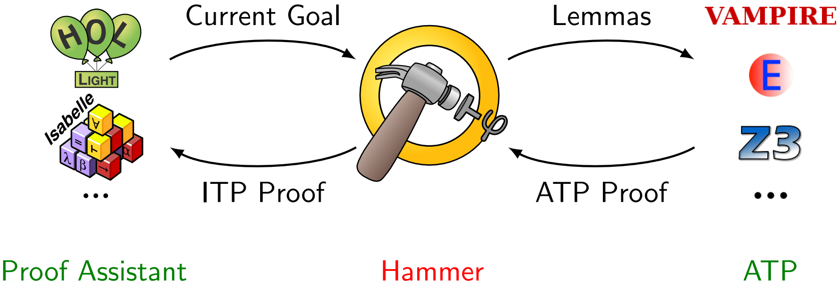
Features used so far for learning

- Symbols
  - symbol names or type-instances of symbols
- Types
  - type constants, type constructors, and type classes
- Subterms
  - various variable normalizations
- Meta-information
  - theory name, presence in various databases

Semantic Features

- The features have to express important semantic relations
- The features must be efficient
- In this work, features for:
  - Matching
  - Unification
- Efficiency achieved by using optimized ATP indexing trees:
  - discrimination trees
  - substitution trees
- Connections between subterms in a term
  - Paths in Term Graphs
- Validity of formulas in diverse finite models
  - semantic, but often expensive

The “Hammer” approach: how much can one do?



- Flyspeck formalization in HOL-light — HOL(y)Hammer
- Mizar Mathematical Lirbrary — MizAR
- Isabelle — Sledgehammer

$\sim$ 45% *success rate*