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

Herman Geuvers[1]

Radboud University Nijmegen
and
Eindhoven University of Technology
The Netherlands

ICT.Open 2011
Veldhoven, November 2011

[1]Thanks to Freek Wiedijk & Foundations group, RU Nijmegen

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

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

Yes it can

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

Yes it can

But it's hard ...

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

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

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)
- **Nqthm** Boyer, Moore (Austin, Texas)
- **LCF** Milner (Stanford; Edinburgh)

What are Proof Assistants – History

Around 1970 five new systems / projects / ideas

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

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, Duitsland
focus: mathematics and programming languages

balanced system
nice proof language
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, Polen

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**

- **Proving**

What Proof Assistants are not

Doing mathematics on a computer

- **Computing:** *numbers*
  numerical mathematics, visualisation, simulation
- **Computing:** *formulas*
  computer algebra
- **Proving**

What Proof Assistants are not

Doing mathematics on a computer

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

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

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

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
- Build up a formal mathematical library
- Verify software and hardware design

Proof Assistants: what are they used for

- Verify mathematical theorems
  Some mathematical proofs just become too large and complex: proof of a Kepler's conjecture
- Build up a formal mathematical library

- Verify software and hardware design

Proof Assistants: what are they used for

- Verify mathematical theorems
  Some mathematical proofs just become too large and complex: proof of a Kepler's conjecture
- Build up a formal mathematical library
  Mizar Mathematical Library

- Verify software and hardware design

Proof Assistants: what are they used for

- Verify mathematical theorems
  Some mathematical proofs just become too large and complex: proof of a Kepler's conjecture
- Build up a formal mathematical library
  Mizar Mathematical Library

- Verify software and hardware design
  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.'

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

How a Proof Assistant works

The different phases in a mathematical proof

1. find a proof
   Everything goes: experiment, guess, simplify, ...
   Will not be preserved in the future, but crucial for students to learn the subject.

How a Proof Assistant works

The different phases in a mathematical proof

1. find a proof
   Everything goes: experiment, guess, simplify, ...
   Will not be preserved in the future, but crucial for students to learn the subject.
2. write down a proof
   Contains explanation why the stated theorem holds and why the proof is the way it is, but also small proof steps that together provide a verification of the theorem.
3. present and communicate a proof
   Explain to others, present in a talk. Improve a proof, simplify it, change it, generalize it.

How a Proof Assistant works

The different phases in a mathematical proof

1. find a proof
   Everything goes: experiment, guess, simplify, ...
   Will not be preserved in the future, but crucial for students to learn the subject.
2. write down a proof
   Contains explanation why the stated theorem holds and why the proof is the way it is, but also small proof steps that together provide a verification of the theorem.
3. present and communicate a proof
   Explain to others, present in a talk. Improve a proof, simplify it, change it, generalize it.

Proof assistant plays a role in (2) and a bit in (3); in the future possibly in (1)

Different styles of formalised proofs

- procedural

- declarative

Different styles of formalised proofs

- procedural
  tell **what to do**

- declarative
  tell **where to go**

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 Limbo trail, ...
- declarative
  tell **where to go**

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 Limbo trail, ...
- declarative
  tell **where to go**
  Go to the platform, go down to the tunnel, to the north exit of the station, go to the KvK building, then go to the "Zwarte Doos", ...

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 . . .

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.

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?

. . . 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).

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.

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

Why would we believe a proof assistant?

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

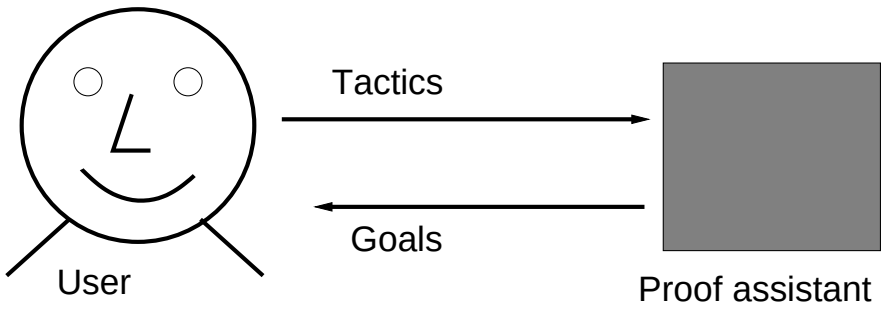
Other possibilities to increase the reliability of the proof assistant

- 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.

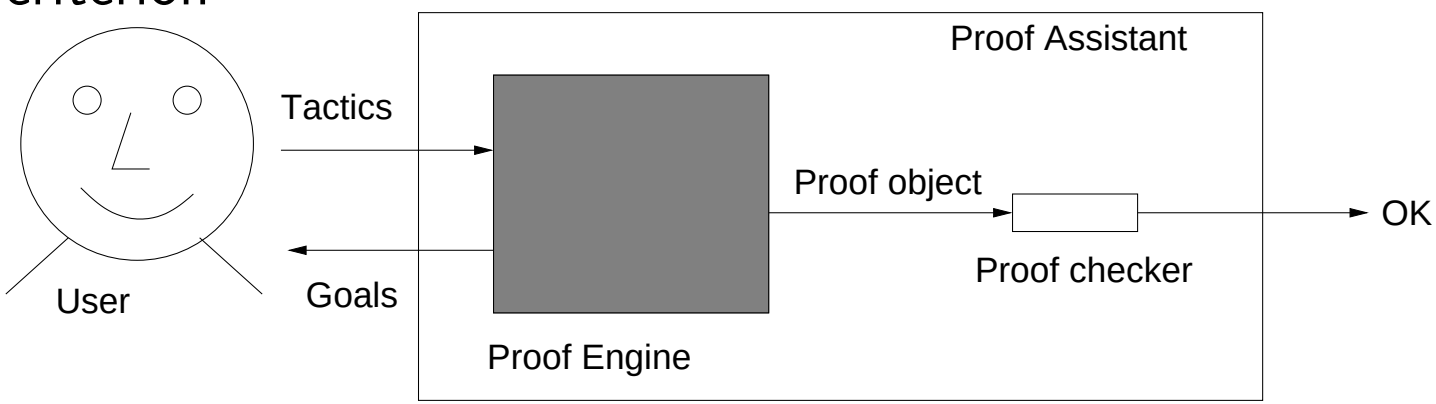
Why would we believe a proof assistant?

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



How would we believe a proof assistant?

Does the formula on the screen correspond to what we have proven?

- Proof Assistants have (sophisticated) notation and rendering mechanisms to make formulas better readable.
- Can I make “True” look like “False” ??

How would we believe a proof assistant?

Does the formula on the screen correspond to what we have proven?

- Proof Assistants have (sophisticated) notation and rendering mechanisms to make formulas better readable.
- Can I make "True" look like "False" ??

Pollack consistency [Wiedijk]: One cannot introduce notation that makes $0 = 1$ provable.

How would we believe a proof assistant?

Does the formula on the screen correspond to what we have proven?

- Proof Assistants have (sophisticated) notation and rendering mechanisms to make formulas better readable.
- Can I make "True" look like "False" ??

Pollack consistency [Wiedijk]: One cannot introduce notation that makes $0 = 1$ provable.

. . . None of the used proof assistants are Pollack consistent . . .

How would we believe a proof assistant?

Does the formula on the screen correspond to what we have proven?

Given that I trust the proof assistant,
how much proof code (definitions) do I need to read (and understand) to believe that the final theorem is the one I wanted to see proven?

How would we believe a proof assistant?

Does the formula on the screen correspond to what we have proven?

Given that I trust the proof assistant,
how much proof code (definitions) do I need to read (and understand) to believe that the final theorem is the one I wanted to see proven?

That's an issue ...
The situation seems different between mathematics and computer science.

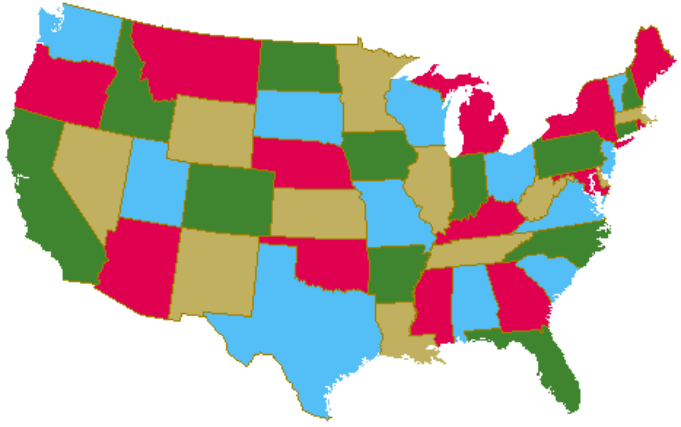
How to believe that we have actually proven a theorem?

Example: The 4 colour theorem

Kenneth Appel en Wolfgang Haken, 1976

Neil Robertson e.a., 1996

Coq: Georges Gonthier, 2004

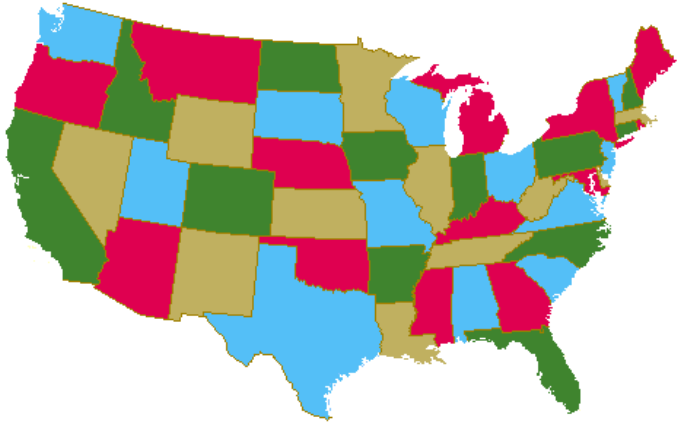


Can *every* map be coloured with only 4 different colours?

How to believe that we have actually proven a theorem?

Example: 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.

How to believe that we have actually proven a theorem?

Example: Compcert (Leroy et al. INRIA 2006)

Verifying an optimizing C-compiler

How to believe that we have actually proven a theorem?

Example: Compcert (Leroy et al. INRIA 2006)

Verifying an optimizing C-compiler

Just stating what the correctness of a C-compiler means already takes several pages . . .

Mathematical users of Proof Assistants

Flyspeck project: Formalizing a proof of the Kepler Conjecture

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

Tom Hales, CMU Pittsburgh

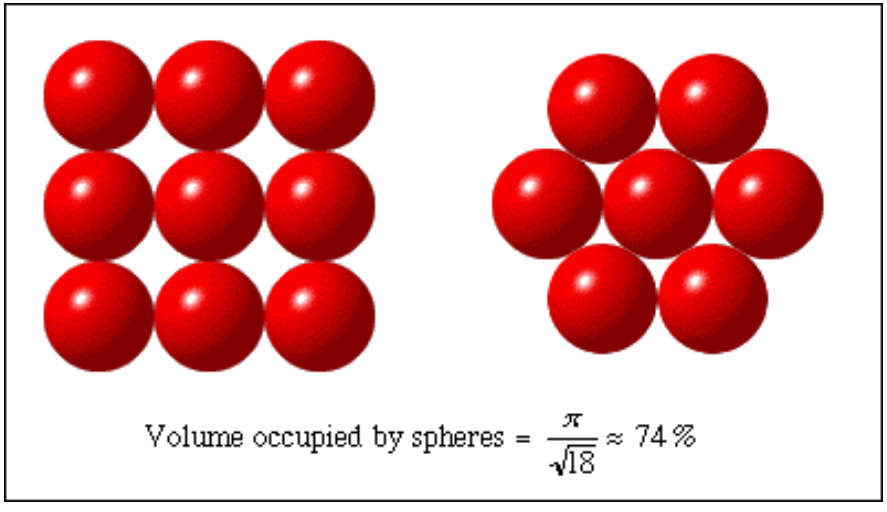
Kepler Conjecture (1611)

Kepler Conjecture (1611)

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

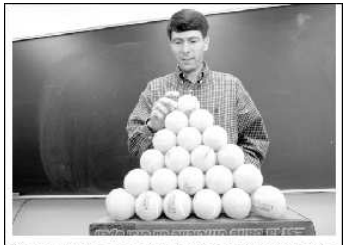
Kepler Conjecture (1611)

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)

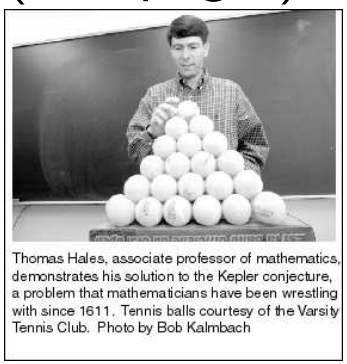


Thomas Hales, associate professor of mathematics, demonstrates his solution to the Kepler conjecture, a problem that mathematicians have been wrestling with since 1611. Tennis balls courtesy of the Varsity Tennis Club. Photo by Bob Kalmbach

- Annals of Mathematics: 99% correct ...

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 1039 inequalities of the shape

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

Hales' proof of the Kepler conjecture

Reduce the problem to 1039 inequalities of the shape

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

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

Some large formalization projects in Computer Science

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

Some large formalization projects in Computer Science

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

Proof Assistants: What needs to be done

Automation

- Formalize all of the Bachelor undergraduate mathematics
- Combination of Theorem Proving and Machine Learning (Urban et al.)
  Use ML to produce a hint databse that can be fed to an Automated Theorem Prover
- Domain Specific Tactics / Automation

Proof Assistants: What needs to be done

Cooperation and Documentation

- PAs cannot cooperate, exchange knowledge: mathematical components
- How to document your development for reuse?
- How to cooperate on a large development?

Proof Assistants: What needs to be done

Cooperation and Documentation

- PAs cannot cooperate, exchange knowledge: mathematical components
- How to document your development for reuse? MathWiki
- How to cooperate on a large development? MathWiki