An Introduction to Proof Assistants

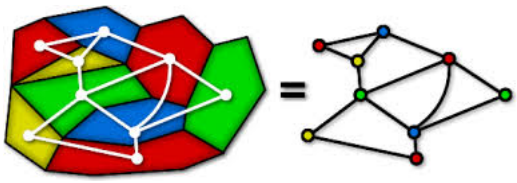
Patrick Schnider

Student Seminar in Combinatorics: Mathematical Software,
ETH Zürich

1 Motivation

The development of proof assistants was motivated by the use of computers in proofs of several mathematical problems. We introduce two of the most famous of those problems, both of which do not yet have a solution without computer assistance.

1.1 The four-colour theorem



Theorem: Every planar graph allows a proper vertex colouring with four colours.

The problem was posed in 1852 by Francis Guthrie to his former Professor Augustus de Morgan. In 1879, Alfred Kempe presented a proof. However, eleven years later, in 1890, Percy Heawood found a mistake in Kempe's argument, but managed to modify the proof in a way that showed that a proper vertex colouring with five colours always exists. In 1976, almost 100 years after Kempe, Kenneth Appel an Wolfgang Haken presented a new proof of the theorem. Their proof relied on a computer checking almost 2000 different cases in more than 1000 hours of computation time. The difficulty of checking whether the calculations of the computer were really correct raised concerns among mathematicians and many were not willing to accept the proof. The easier proof published in 1996 by Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas, which reduced the argument to "only" 633

cases still couldn't convince the sceptics. In 2005, Georges Gonthier finished formalizing the proof in the Proof Assistant Coq. This removed the need to trust complicated programs doing difficult computations, the only program that we need to trust is the Coq proof verifier, a piece of software that is much easier than the programs used in the other proofs.

1.2 Kepler Conjecture

Conjecture: No arrangement of equally sized spheres filling space has a greater average density than that of the cubic close packing and hexagonal close packing arrangements.

This problem was posed by Johannes Kepler in 1611 and remained completely open for more than 200 years. Only in 1831 Carl Friedrich Gauss proofed that the statement was true, provided the centres of the spheres were arranged in a lattice. The problem was included by David Hilbert in his "twenty three unsolved problems of mathematics" in 1900. In 1953, Laszlo Fejes-Toth showed that it was enough to proof the conjecture one a finite (even though large) region. Thus a proof by exhaustion would, in principle, be possible. In fact, the conjecture could be reduced to a problem of maximizing a function in 150 variables, as Tom Hales discovered. In 1998, Hales presented a proof based on this approach. The proof involved solving around 100'000 linear programs. The referees for the Annals of Mathematics said they were "99% certain of correctness", but that they could not certify the correctness of all computer calculations. Tom Hales, aiming to remove these uncertainties, initiated the project "Flyspeck" in 2003, to formalize his proof using the proof assistants HOL light and Isabelle. The project was announced to be completed in August 2014.

2 What are Proof Assistants?

A formal proof is a finite sequence of sentences. Each sentence is either an axiom or follows from the preceding sentences. The last sentence in the sequence is a theorem. Formal proofs can be checked by computers effectively. However, finding formal proofs is in general very hard.
A proof assistant is a software that interacts with the user to find formal proofs. Proof assistants are not to be confused with automatic theorem

provers, the user input is always required. But, depending on the assistant, certain tasks are automated.
There are many proof assistant software used, the most famous being HOL light, Isabelle, Coq and Mizar. While HOL light, Isabelle and Coq all implement higher order logic and offer automated tools, Mizar provides neither of those but instead has the most extensive library of pre-proved theorems.
As mentioned above, proof assistants are not automated theorem provers. On the other hand, they are also suitable to compute solutions for complicated mathematical problems. For these problems there are, depending on the type of problem, lots of software in numerical mathematics, computer algebra and many specialised programs. While these programs allow very interesting mathematics, they give the users no proof. Automated theorem provers give proofs, but, by the complexity of finding proofs, only very seldom interesting mathematics. Proof assistants offer both, but for the price that the user has to do a lot of work.

3 An example: Coq

Coq was developed by INRIA in France. The first release was in 1989. Coq is written in OCaml, a dialect of ML also developed at INRIA. The user communicates with Coq in a language called Gallina. The logical formalism used is the calculus of inductive constructions (CIC). CIC only uses five inference rules, which allows a relatively simple implementation of a proof checker.
Coq is available for all major platforms and also includes two different graphical interfaces (depending on the platform), CoqIDE and ProofGeneral. Coq also has many automated proof tactics and libraries with pre-proved theorems.

3.1 Using Coq

Let us take a look at en example of a proof done with Coq. We will prove the rather easy statement that for any type A and for any maps P and Q form A to a proposition we have that

$$(\forall x \in A : P(x)) \vee (\forall y \in A : Q(y)) \Longrightarrow \forall x \in A : P(x) \vee Q(x).$$

We start by stating the theorem:

```
Theorem thm_name:  forall A: Type, forall P Q: A -> Prop,
(forall x, P x)\/(forall y, Q y) -> forall x, P x \/ Q x.
```

Coq answers with:

```
1 subgoals
______________________________(1/1)
forall (A : Type) (P Q : A -> Prop),
(forall x :  A, P x) \/ (forall y :  A, Q y) -> forall x :  A, P x \/ Q x
```

We start the proof by writing

```
Proof.
```

We then introduce the variables A, P, Q and the hypothesis H:

```
intros A P Q H.
```

Coq answers with:

```
1 subgoals
A : Type
P : A -> Prop
Q : A -> Prop
H : (forall x :  A, P x) \/ (forall y :  A, Q y)
______________________________(1/1)
forall x :  A, P x \/ Q x
```

We want to distinguish the cases where the left side or the right side of the hypothesis hold. This is done as follows:

```
destruct H as [H1 | H2].
```

Coq answers with:

```
2 subgoals
A : Type
P : A -> Prop
Q : A -> Prop
H1 :  forall x :  A, P x
______________________________(1/2)
forall x :  A, P x \/ Q x
______________________________(2/2)
forall x :  A, P x \/ Q x
```

We only want to prove the left side of the first goal:

```
left.
```

Coq answers with:

```
2 subgoals
A : Type
P : A -> Prop
Q : A -> Prop
H1 :  forall x :  A, P x
x :  A
______________________________(1/2)
P x
______________________________(2/2)
forall x :  A, P x \/ Q x
```

The first goal is now just the hypothesis:

```
apply H1.
```

Coq answers with:

```
1 subgoals
A : Type
P : A -> Prop
Q : A -> Prop
H2 :  forall y :  A, Q y
______________________________(1/1)
forall x :  A, P x  Q x
```

We can now just proof the right side of the goal in the exact same way:

```
right.
apply H2.
```

Coq answers with:

```
No more subgoals.
```

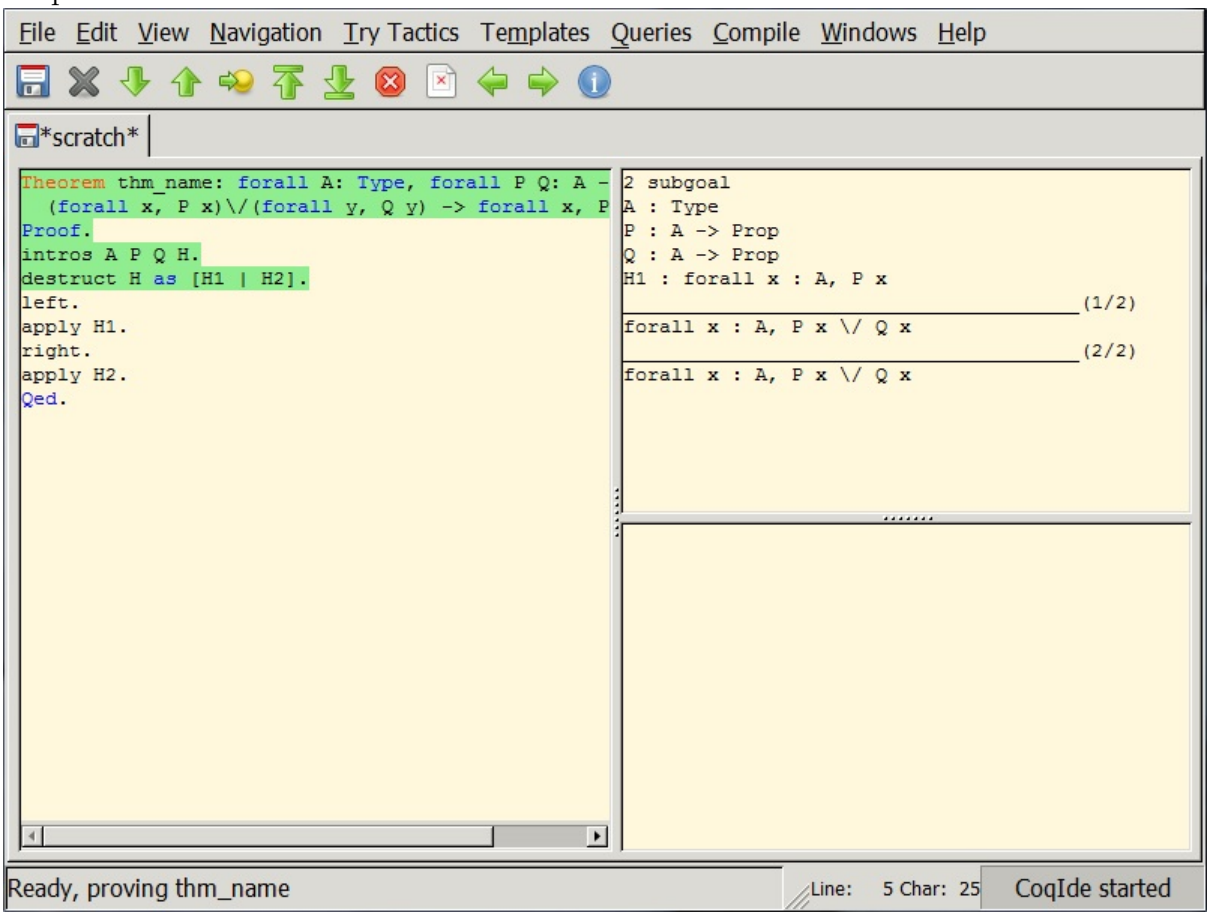
We save the proof by writing

```
Qed.
```

Coq answers with:

```
thm_name is defined
```

The following image shows how this proof looks in the graphical interface CoqIDE:



The proof can be significantly shortened by calling an automated theorem proof tactic for recognizing first-order logic tautologies called `firstorder`:

```
Proof.
firstorder.
Qed.
```

3.2 Implementation

The implementation of Coq is based on 8 parts:

Part	**Function**
1. The logical framework	Meta-language for terms of CIC
2. The language of constructions	language for CIC
3. The type-checker (Kernel)	checks formal proofs
4. The proof engine	interactive proof construction
5. The language of tactics	library of pre-implemented tactics
6. The vernacular interpreter	Interpreter of Gallina inputs
7. The parser and pretty-printer	Translation strings $\leftrightarrow$ formulas
8. The standard library	pre-implemented modules

A few remarks about the Kernel and the language of tactics can be found in the next two sections.

3.2.1 The Kernel

The kernel is the most important part concerning the trustworthiness of a proof assistant. The kernel checks a formal proof for its validity, thus trusting a proof found by a proof assistant effectively means trusting its proof checking engine. In fact, any mistakes done in other parts of the program do not matter at all, if the formal proof in the end is correct. This idea is recorded in the following criterion:

Definition: A proof assistant satisfies the de Bruijn criterion if it generates proofs that can be checked (independently of the system that created it) using a simple program (that a skeptical user can write him/herself).

In Coq, the Kernel is independent of the rest of the system and relatively small. The same also holds for HOL light and Isabelle.

3.2.2 The language of tactics

Tactics are automated tools to prove or simplify statements. Coq offers a wide variety of tactics, including tautology recognition in first-order logic, deciding equality for inductive types, different simplifying procedures and many more. Tactics can also be programmed or extended by user. Tactics can also call other tactics. Tactics that call other tactics are called defined

tactics, those that do not call other tactics are called primitive tactics. Primitive tactics include introducing variables or changing terms into equivalent terms.
To get an idea how such tactics can be implemented, let us take a look at a part of the source code of the tactic `tauto`, which recognizes tautologies in quantifier-free logical formulas:

```
<:tactic<
  $t_not_dep_intros;
  repeat
     (match reverse goal with
      | id: ?X1 |- _ => $t_is_conj; elim id; do 2 intro; clear id
      | id: (Coq.Init.Logic.iff _ _) |- _ => elim id; do 2 intro; clear id
      | id: (Coq.Init.Logic.not _) |- _ => red in id
      | id: ?X1 |- _ => $t_is_disj; elim id; intro; clear id
      | id0: (forall (_: ?X1), ?X2), id1: ?X1|- _ =>
         (* generalize (id0 id1); intro; clear id0 does not work
            (see Marco Maggiesi's bug PR#301)
         so we instead use Assert and exact. *)
         assert X2; [exact (id0 id1) | clear id0]
      | id: forall (_ : ?X1), ?X2|- _ =>
        $t_is_unit_or_eq; cut X2;
          [ intro; clear id
          | (* id : forall (_: ?X1), ?X2 |- ?X2 *)
            cut X1; [exact id| $c1; fail]
          ]
      | id: forall (_ : ?X1), ?X2|- _ =>
        $t_flatten_contravariant_conj
        (* moved from "id:(?A/\?B)->?X2|-" to "?A->?B->?X2|-" *)
      | id: forall (_: Coq.Init.Logic.iff ?X1 ?X2), ?X3|- _ =>
        assert (forall (_: forall _:X1, X2), forall (_: forall _: X2, X1), X3)
          by (do 2 intro; apply id; split; assumption);
          clear id
      | id: forall (_:?X1), ?X2|- _ =>
        $t_flatten_contravariant_disj
        (* moved from "id:(?A\/?B)->?X2|-" to "?A->?X2,?B->?X2|-" *)
      | |- ?X1 => $t_is_conj; split
      | |- (Coq.Init.Logic.iff _ _) => split
      | |- (Coq.Init.Logic.not _) => red
      end;
      $t_not_dep_intros) >>
```

The whole code has 399 lines, including comments. The tactic basically unfolds the architecture of the formulas and applies appropriate tactics for each logical connective.

4 Criticism

In his paper "Flyspecking Flyspeck" Mark Adams mentions seven concerns about the trustworthiness of proof assistants, that any referee of such a proof must adress in his opinion:

1. **Has a final theorem actually been proved in the assistant?**
   The proof scripts might fail to produce a formal proof when processed altogether in one session. All scripts must be rerun by the independent referee.

2. **Does the final statement really mean what we think it means?**
   There might be a subtle mistake in a definition or statement of an auxiliary lemma which might change the meaning of the final theorem.

3. **Were any axioms added that make the proof assistants theory inconsistent?**

4. **Are the settings for displaying concrete syntax configured in a way that happen to make a statement get misinterpreted?**
   There might be different codes that get displayed as the same symbol, which would allow more than one interpretation of some displayed statements.

5. **Can we trust the proof assistant to correctly record and display all the information required for the review?**)
   This concern is explained in the next section (Pollack-inconsistency).

6. **Is the proof assistant sound?**
   There might be a programming error in the proof checker, which would invalidate the whole result.

7. **Is there a proof script that could make the proof assistant unsound?**
   Could it be somehow possible to reprogram the kernel in a proof script? The soundness must hold before, during and after processing a proof script.

Also, any auditor must assume malicious intent. It is quite improbable that any of the above concerns happens purely by accident, but it is always possible that somebody exploits a vulnerability in the proof assistant, be it because of laziness, time pressure or any other reason. This is especially a concern in projects like Flyspeck where a bounty is attached to every successful proof of an intermediate step.

4.1 Pollack-inconsistency

As any program, a proof assistant includes a parser, which turns input strings into formulas that the computer can work with, and a printer, which translates the computer formulas into strings that can the be displayed as output. In an ideal world, for any formula Φ we would have

$$\text{parse}(\text{print}(\Phi)) = \Phi.$$

In practice however, this sometimes breaks.

We can extend our axioms by the so-called Pollack-axioms: for any formulas Φ_1 and Φ_2 we have

$$(\text{print}(\Phi_1) = \text{print}(\Phi_2)) \Longrightarrow (\Phi_1 \Leftrightarrow \Phi_2)$$

We call a proof assistant Pollack-inconsistent if False is provable from Pollack-axioms.
It turns out that HOL light, Isabelle, Coq and Mizar are all Pollack-inconsistent! Thus, we really need to be sure that we did not use any Pollack-axioms in the proof.

5 References

- Wikipedia:
  - http://en.wikipedia.org/wiki/Four_color_theorem
  - http://en.wikipedia.org/wiki/Kepler_conjecture
  - http://en.wikipedia.org/wiki/Proof_assistant
  - http://en.wikipedia.org/wiki/Formal_proof
  - http://en.wikipedia.org/wiki/Coq
- Coq Homepage: https://coq.inria.fr/
- Coq References:
  - Coq Reference Manual
  - "Coq in a hurry": https://cel.archives-ouvertes.fr/inria-00001173v5/document
  - "Theorem Proving with Coq": http://flint.cs.yale.edu/cs430/sectionNotes/section1/CoqTutorial.pdf

  - Book "Software Foundations", B.Pierce et al: http://www.cis.upenn.edu/ bcpierce/sf/current/index.html

- Presentation by H. Geuvers: http://www.cs.ru.nl/ herman/ictopen.pdf
- Coq Source Code repository: https://gforge.inria.fr/git/coq/coq.git
- Mark Adams: Flyspecking Flyspeck. In: Mathematical Software - ICMS 2014.
- T.C. Hales: Computational Discrete Geometry. In: Mathematical Software - ICMS 2010.
- G. Gonthier: A computer-checked proof of the four colour theorem. http://research.microsoft.com/en-us/um/people/gonthier/4colproof.pdf