Formal verification of Zagier's one-sentence proof

Guillaume Dubach[*] & Fabian Mühlböck[†]
IST Austria

It is futile to do with more things that which can be done with fewer.
William of Ockham, *Summa Logicae* i.12

Abstract

We comment on two formal proofs of Fermat's sum of two squares theorem, written using the Mathematical Components libraries of the Coq proof assistant. The first one follows Zagier's celebrated one-sentence proof; the second follows David Christopher's recent new proof relying on partition-theoretic arguments. Both formal proofs rely on a general property of involutions of finite sets, of independent interest. The proof technique consists for the most part of automating recurrent tasks (such as case distinctions and computations on natural numbers) via ad hoc tactics.

Introduction

In his 1990 paper [13], Zagier provided what he advertized as a 'one sentence proof' of the following proposition, first stated by Fermat in a letter to Mersenne:

Theorem 1 (Fermat)**.** *Every prime number p such that $p \equiv 1 \bmod 4$ is a sum of two squares, i.e. there exists $a, b \in \mathbb{N}$ such that $p = a^2 + b^2$.*

Zagier's proof relies on the basic fact that an involution f on a finite set S is such that the number of fixed points of f and the cardinality $|S|$ have the same parity. This principle is used twice, in complementary ways: after defining an appropriate set S, a first involution is invoked to justify that $|S|$ is odd; this implies that *another* involution on S has at least one fixed point; and any such fixed point then yields two integers a and b solutions of the problem. The second involution is a very natural one, whereas the first involution appears at first sight to be extremely involved. The trick, in order to turn this argument into a 'one-sentence proof', is to only state the definitions and major steps as one long sentence, leaving all details and especially all ancillary (but logically necessary) verifications to the reader. This condensed writing style and the mysterious origin of the first involution undoubtedly endowed this proof with rare appeal when it first appeared, and still does some thirty years later.

More recently, another elegant argument was given by David Christopher [4]. His proof involves a few more steps, and the definition of four different finite sets instead of one, corresponding to partitions of the integer p with different properties. The basic mechanism, as for

[*]guillaume.dubach@ist.ac.at [†]fabian.muehlboeck@ist.ac.at

Zagier's proof, is to consider involutions on these finite sets. The whole argument can hardly be stated as only one sentence, but it is safe to say that it takes less time to most readers to be convinced of its validity, as each step left to the reader is truly more elementary.

The purpose of this paper is to present these two proofs in details and to comment on their formalisation using the Coq proof assistant. The current state of the corresponding code can be found on the following github repository:

`https://github.com/gdubach/Zagier_project`.

The code essentially relies on the Mathematical Components libraries, presented in the fundamental book [9], and takes advantage of the SSReflect (*small scale reflection*) syntax, whose general presentation [6] is also included in the Coq reference manual [3]. The tutorial [5] is another valuable reference on this particular syntax.

The general property of involutions over finite sets needed in both proofs is briefly presented in Section 1, and might be of independent interest. Section 2 presents Zagier's original proof together with a recent geometric interpretation of it, and then comments on the code methodology. Section 3 similarly provides a general presentation of David Christopher's proof, and comments on the features of the code that are specific to this alternative argument.

A general aspect of the formal verification of these proofs is that it somehow reverses the usual perspective, in the sense that the difficulties and subtleties concern mostly things that are self-evident for a human mind, and vice-versa: what takes some time to a mathematician, such as checking that Zagier's first map is indeed an involution, is effortless once the computation is automated. However, as far as the logical structure is concerned, the formal proof faithfully reflects the steps of the original proof.

Note that the motivation of this work lies in the structure of the proof only, and not in the result proven. Indeed, Fermat's sum of two squares theorem already appears in the Coq litterature: it has been verified by Laurent Théry in 2016 [12]. Théry's formal proof relies on the properties of the ring $\mathbf{Z}[i]$ of Gauss integers – a very different approach than the combinatorial arguments formalized here. The same approach was used by Chris Hughes [8] to write this formal proof in the Lean Theorem Prover. Establishing all relevant properties of Gauss integers takes more lines of code than following Zagier's legerdemain approach, but it is undoubtedly a more robust and canonical technique.

It was pointed out to the authors that one of the formal proofs of Fermat's sum of two squares theorem written in HOL Light also followed Zagier's approach [2]; the present code was developed independently and without knowledge of this work.

Acknowledgments

We would like to thank Marie Kerjean for her help and advice with the Mathematical Components libraries. Answers and advice from the Coq community on Zulip [14] have been also of great help; special thanks to Cyril Cohen and Christian Doczkal for explaining to us some aspects of the `finmap` library.

G. Dubach gratefully acknowledge funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 754411. F. Mühlböck's research is supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award).

1 Involutions of finite sets

The proofs we formalize rely on a general property of involutions over finite sets:

Lemma 2. *If S is a finite set, an involution $f : S \to S$ is such that*

$$|\{s \in S \ : \ f(s) = s\}| \equiv |S| \bmod 2. \tag{1}$$

In particular, if $|S|$ is odd, every such involution has at least one fixed point.

In professional mathematical papers, this is typically a fact considered so intuitive, and indeed elementary, that a justification would be superfluous. Formalizing it, however, still demands careful consideration. To do so, we work in the context of a variable Type K (which will be $\mathbb{N}^3$ or $\mathbb{N}^4$ in the proofs). It matters to the structure of the proof and especially to the partial automation that K should be a possibly infinite type on which a function f is defined, that is an involution only on a finite subset $E \subset K$. The notion of a *finite* subset of a possibly *infinite* type is given by the *fset* structure from the *finmap* folder, a recent and valuable addition to the Mathematical Components library. So we write, with the appropriate imports:

```
Variable K: choiceType.

Definition involution_on (E: {fset K}) (f:K→ K) :=
(∀ x, x |in E → f x \in E) ∧ (∀ x, x \in E → f (f x) = x).

Definition fixedset (E: {fset K}) (f:K→ K):{fset K} := [fset x in E | x==f x].
```

And the lemma we want to prove is stated as follows:

```
Lemma involution_lemma (f:K→ K): ∀ (E:{fset K}),
involution_on E f → odd #|‘(fixed_fset E f)| = odd #|‘E|.
```

The proof is rather straightforward after a few technical lemmas. We perform a strong induction on the number of the non-fixed points, i.e. on the natural integer

```
n := #|‘E‘\‘(fixedset E f)| : nat
```

i.e. the number of non-fixed points of f. The reason why strong induction is appropriate here is that a non-fixed point $x \neq f(x)$ can always be removed from the set E together with its image $f(x)$, f being still an involution on the smaller set $F_x := E\backslash\{x, f(x)\}$. So the basic induction step reduces the number of non-fixed points from $n + 2$ to n. Writing this as a strong induction also eliminates the need to treat the case $n = 1$ explicitly; the fact that $n = 1$ can never occur is embedded in the structure of the proof.

One technical aspect is that we need the induction hypothesis to state that the conclusion holds for *any finite set* E as well as any f such that the number of non-fixed points of f on E is n. Indeed, we then want to apply it the same function f restricted to the smaller set F_x. To conclude the proof, it is to be verified that

(i) f is still an involution of F_x,

(ii) the number of fixed points did not change,

(iii) the cardinality of the base set (F_x instead of E) has decreased by 2, and

(iv) so has the cardinality of the non-fixed points.

A shorter proof would certainly have been obtained by assuming the Type K to be endowed with a total order (which can easily be done on $\mathbb{N}^3, \mathbb{N}^4$) and considering the subsets of E such that $f(x) < x$, $f(x) = x$ and $f(x) > x$. We chose not to rely on an order structure for this proof, so that the Theorem could be used in a less cumbersome way on any type K.

2 Zagier's one-sentence proof

2.1 The 1990 proof and a recent illustration

Let us first re-state Zagier's proof the way he wrote it.

Zagier's one-sentence proof. The involution on the finite set

$$S_p := \{(x,y,z) \in \mathbb{N}^3 \mid x^2 + 4yz = p\} \tag{2}$$

defined by

$$(x,y,z) \mapsto \begin{cases} (x+2z, z, y-x-z) & \text{if } x < y-z \\ (2y-x, y, x-y+z) & \text{if } y-z < x < 2y \\ (x-2y, x-y+z, y) & \text{if } x > 2y \end{cases} \tag{3}$$

has exactly one fixed point, so $|S_p|$ is odd, and the involution defined by

$$(x,y,z) \mapsto (x,z,y). \tag{4}$$

also has a fixed point. $\square$

Writing the above proof in only one sentence is clearly a bravura piece; let us make a few comments to unpack the argument. It is first implicitly left to the reader to check

(i) that (2) defines a finite set S_p;

(ii) that (3) defines an involution on S_p, which is far from obvious;

(iii) that (3) has exactly one fixed point, which also requires to be worked out.

From this, the elementary Lemma 2 is invoked to deduce that $|S_p|$ is odd, which is the turning point of the proof. It is then, again, left to the reader to check that

(iv) (4) also defines an involution on S_p.

Therefore, by a second appeal to Lemma 2, this involution on a finite set of odd cardinality must have at least one fixed point. By definition, such a fixed point is a triple $(x,y,y) \in S_p$, which implies that $p = x^2 + 4y^2$, a sum of two squares, which concludes the proof.

Concerning the general outline of the argument, Zagier refers to a former proof by Heath-Brown [7] and the works of Liouville. However, Zagier's proof contains a new ingredient, namely the involution defined by (3), which constitutes the most intriguing part of his very short paper. It is not clear at first sight how anyone would come up with the idea of such a map, nor is it clear why this should be an involution on S_p.

In recent years, a geometric interpretation has become popular, and now appears in various forms online. Its origin seems to be a paper by A. Spivak [11]. It is a simple and beautiful idea that makes the involution (3) and its distinction of three cases appear naturally. First, each triple (x,y,z) is associated to a 'windmill', i.e. a square of side x surrounded by four rectangles of dimensions $y \times z$; it is important to specify that y is the length of the side situated *along* the $x \times x$ square. This being posed, the windmill is unique up to symmetry – symmetric windmills being considered equivalent for the sake of the argument[1]. Such windmills can be decomposed in five types, according to the relative size of y compared to x and $x+z$.

[1] For instance, Type 2 and Type 4 in Figure 1 do not have the same orientation, but what matters is the outer shape.

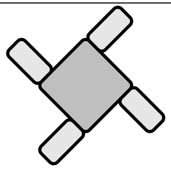
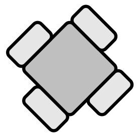
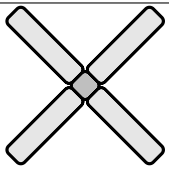
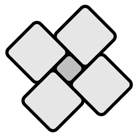
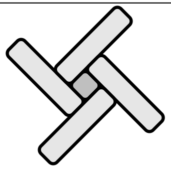
Type 1	Type 2	Type 3	Type 4	Type 5
$y < x/2$	$x/2 < y < x$	$x = y$	$x < y < x + z$	$x + z < y$
				

Figure 1: Exactly five types of windmill that may appear when p is a prime number congruent to 1 mod. 4.

The boundary cases $x = 2y$ and $y = x + z$ do not happen when p is prime, so we need not include them; neither do we need to worry about $x = 0, y = 0$ or $z = 0$. The key observation is that for any given *outer shape* of one of these windmills, there are *either one or two* windmills that fit in it. So to each windmill, a *dual* windmill can be associated – self-dual windmills being those for which only one possible windmill has the same outer shape. This duality defines an involution on the set of windmills of a given area, which action on the dimensions (x, y, z) is precisely given by the set of equations (3). It so happens that windmills of types 2,3 and 4 result in the same equation, which is the reason why Zagier's definition distinguishes three cases and not five.

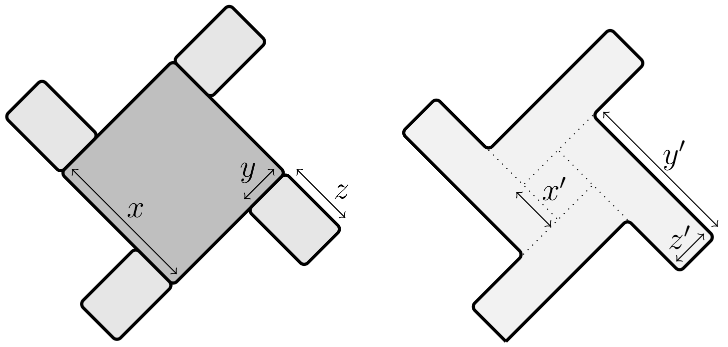


Figure 2: A windmill associated to a triple (x, y, z) and its outer shape. In dotted lines is its image by the Zagier involution, corresponding to the third line of (3).

One can see from Figures 1 and 2 that windmills of type 1 (resp. 2) are paired with windmills of type 5 (resp. 4), whereas windmills of type 3 are self-dual and constitute the fixed point(s) of Zagier's map; it is also quite straightforward, from considerations of area, to prove that such a fixed point occurs and is unique when p is a prime congruent to 1 mod 4. Figures 3, 4, 5 below illustrate this for $p = 17$, where $|S_p| = 5$; windmills with the same outer shape are represented side by side.

This geometric picture should help convince the *bona fide* sceptical reader that Zagier's formulae (3) indeed define an involution on S_p. At the same time, it should be acknowledged that checking this fact *without* Spivak's windmill interpretation is more of a challenge, and requires some time and concentration, even though, admittedly, it only requires to go through a sequence of elementary steps.

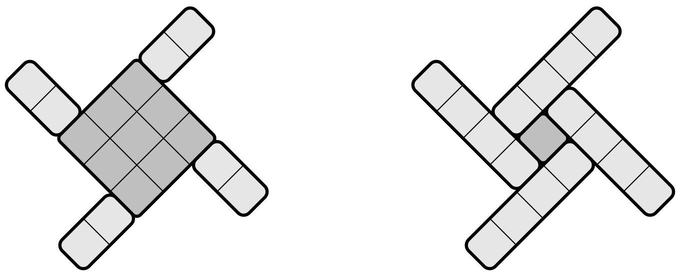


Figure 3: Windmills of type 1 ($2y < x$) are paired with windmills of type 5 ($y > x + z$). Above are $(3, 1, 2)$ and $(1, 4, 1)$ in S_{17}.

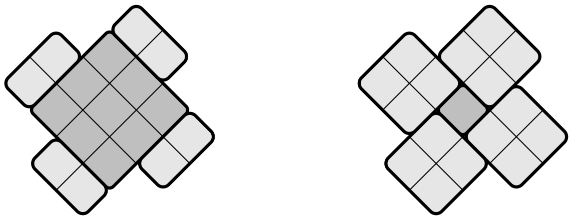


Figure 4: Windmills of type 2 ($x/2 < y < x$) are paired with windmills of type 3 ($x < y < x + z$). Above are $(3, 2, 1)$ and $(1, 2, 2)$ in S_{17}.

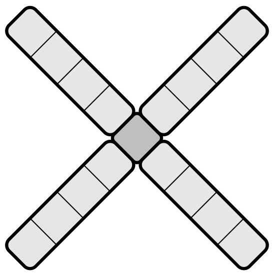


Figure 5: $(1, 1, 4)$ is the fixed point of the Zagier involution on S_{17}.

2.2 Formal verification of Zagier's proof

Although the windmill picture is a very helpful one in order to convince an audience (and therefore a powerful observation as far as the proof as a social fact is concerned), Coq does not need this interpretation; nor does it need this distinction of five cases to perform the necessary computations. Instead, it can simply enumerate the 9 different cases of applying (3) twice to a triple of numbers, and discharge each of them either by proving equality or a contradiction. Doing this kind of enumeration is just as uninteresting on a computer as it is on paper, hence we automated it by writing `Ltac` tactics to do *proof search*. In addition, we wrote some tactics to automate and give names to common tasks. We used `Ltac` because of our existing experience with it; as a result, the way the tactics operate does not necessarily reflect the general philosophy of SSReflect (for example, the tactic `hyp_progress` is a bookkeeping operation on a hypothesis). Nevertheless, this does not seem to conflict with the rest of the system. The following main tactics show up in various places in our proofs:

- `by_contradiction H`: starts a proof by contradiction by doing a case analysis on the current (boolean) goal. In one case, the goal is automatically satisfied and thus discharged. This leaves the contrapositive case, which is now asserted by a hypothesis named `H`. The tactic inverts itself if the goal is a negated boolean, and also has a shortform `by_contra` if the name of the hypothesis should be automatically generated.

- `hyp_progress H`: takes a hypothesis of the form `X -> Y`, introduces a new goal `X` and then modifies the hypothesis `H` to only consist of `Y` in the original goal.
- `destruct_boolhyp H`: automatically destructs the boolean connectives || and && in hypothesis `H` recursively, moving their components back to the goal to be individually named. Also tries to apply distributivity first to reduce the number of new goals.
- `mccontradiction`: discharges a goal whose hypotheses form an obvious contradiction, like $H_1 : a < b, H_2 : b < a$ or $H_1 : X, H_2 : \sim\sim X$.
- `mcnia`/`mclia`: uses another (otherwise unused) tactic `mctocoq` to translate MathComp-style arithmetic expressions in the context and the goal into core Coq arithmetic expressions and then applies the `micromega` [1] tactics `nia`/`lia` included with Coq's standard distribution to discharge the goal. The tactic is limited to the kinds of expressions that we see in our proofs; in the future one would probably want to use mczify [10] – the only reason that we did not was that mczify was not available in the standard Windows distribution of Coq at the time we wrote our proofs.
- `mcsolve`: tries some basic transformations on the hypotheses (e.g. `destruct_boolhyp`) and then tries to apply `auto`, `mcnia`, or `mccontradiction` to discharge the goal.
- `zag_solve`: Our main proof search tactic. Among other things, introduces all hypotheses from the goal, looks for boolean if-then-else blocks and does a case analysis on their condition, destructs elements that are pattern-matched in let-expressions, and splits pair/tuple equality goals into equality goals of their constituting components. After these simplifications, it tries to apply mcsolve to each generated goal.

Most parts of our proof then try to quickly get to a state where one of the proof search tactics, in the most general case `zag_solve`, can take over.

The formal verification of Zagier's proof begins by invoking the general notions about involutions, presented in Section 1, by specifying that `K` is here $\mathbb{N}^3$, as Zagier's set S_p contains triples of integers.

```
Definition N3 : Type := nat * nat * nat.
Definition involutionN3:= (@involution_on [choiceType of N3]).
Definition fixed_fsetN3:=(@fixed_fset [choiceType of N3]).
Definition involutionN3_lemma:=(@involution_lemma [choiceType of N3]).
```

The next lines introduce a generic prime number p and transport the MathComp finite set `'I_p` (that corresponds to integers between 0 and $p-1$) into an object of type `{fset nat}`, and triples of that type.

```
Variable p:nat.
Variable p_prime : prime p.
Definition Ipfset:{fset nat} := [fsetval n in 'I_p].
Definition Ipf3:{fset N3} := (Ipfset`*`Ipfset`*`Ipfset).
```

We can now define Zagier's set S_p as the triples of integers (x, y, z) with $x^2 + 4yz = p$, which we call `area` by reference to the area of a windmill (although the geometric interpretation is of no particular use here).

```
Definition area (t:N3) :nat := (t.1.1)^2+4*(t.1.2)*(t.2).
Definition S:{fset N3} := [fset t:N3 | t \in Ipf3 & (p == area t)].
```

Note that the finite set `Ipf3` is used above in order to ensure that `S` itself is finite; without this bound, it would simply not be possible to define it as an object of type `{fset N3}`.

Obviously, we will also need a lemma that proves that this bound actually follows from the area condition – so that this set S truly is the same as S_p. This lemma is called bound_Sp.

The last things we need to define are the two maps introduced by Zagier from the set S_p to itself. We define them as maps from $\mathbb{N}^3$ to $\mathbb{N}^3$ and will verify that they induce 'involutions on S' in the sense explained in Section 1.

```
Definition zig (t : N3) :N3 := (t.1.1, t.2, t.1.2).

Definition zag (t:N3) :N3 := match t with (x,y,z) =>
    if y ≥ (x+z) then (x+2*z,z,y-(x+z))
else if (2*y) ≥ x then (2*y-x,y,z+x-y)
                     else (x-2*y,z+x-y,y) end.
```

The theorems that follow these definitions establish, one by one, the different requirements of Zagier's proof; that is, everything that was left to the reader, such as the fact that zig and zag are involutions on S. These lemmas, for instance, are called zig_involution and zag_involution. Here is what the proof of the second one – arguably the most challenging claim, in this proof, to check with paper an pen – looks like:

```
Lemma zag_involution: involutionN3 S zag.
Proof.
rewrite /involution_on; split; move => [[x y] z].
 - rewrite !inE /area /zag /Ipfset /= /area /zag => hin.
   destruct_boolhyp hin => hx hy hz /eqP hp.
   have harea_p := area_p hp.
   zag_solve.
 - rewrite !inE /zag => htS; destruct_boolhyp htS => hx hy hz /eqP hp.
   have harea_p := area_p hp.
   zag_solve.
Qed.
```

Other prerequisites are written similarly, that is, with the help of Mathematical Components libraries, the SSReflect syntax, and the above tactics that make part of the analysis automatic. Once this preliminary work has been done, the lines of the final proof can essentially be interlaced with Zagier's original sentence:

```
Theorem Fermat_Zagier : p %% 4 = 1 → ∃ a b :nat, p = a^2 + b^2.
Proof.
move /modulo_ex => [k hk].
```

'The involution on the finite set S defined by zag'

```
have h_zag_invol:=zag_involution.
```

'has exactly one fixed point,'

```
have h_zag_fix_card:(#|`(fixed_fsetN3 S zag)|) = 1.
   - by rewrite (zag_fixed_point hk); first by apply: cardfs1.
```

'so |`S| is odd,'

```
have h_S_odd: odd(#|`S|).
   by rewrite -(involutionN3_lemma h_zag_invol) h_zag_fix_card.
```

'and the involution defined by zig.'

```
have h_zig_invol:= zig_involution.
```

'also has a fixed point.'

```
have [t htzigfix]: ∃ t:N3, t \in (fixed_fsetN3 S zig).
  by apply odd_existence; rewrite (involutionN3_lemma h_zig_invol).
by apply (zig_solution htzigfix).
Qed.
```

3 David Christopher's proof

An alternative proof of Fermat's theorem was recently given by David Christopher [4], which shares with Zagier's proof the appeal to involutions, used in both ways. The sets on which these involutions are defined have a different (arguably more elementary) structure than Zagier's set S_p. We present this method below; first in the classical sense, then our formal verification of it.

3.1 General presentation of the argument

The main set of interest in David Christopher's proof encodes integer partitions of p, which parts take exactly *two* values, a_1 and a_2, with $a_1 > a_2$.

$$\mathscr{P}_2^p := \{(a_1, f_1, a_2, f_2) \in \mathbb{N}^4 \mid a_1 > a_2,\ p = a_1 f_1 + a_2 f_2\}. \tag{5}$$

An element of $\mathscr{P}_2^p$ can be represented by a Young diagram with two steps, as illustrated on Figure 6. The set of such diagrams is stable by conjugation, which induces the following involution on $\mathscr{P}_2^p$:

$$\tau : (a_1, f_1, a_2, f_2) \mapsto (f_1 + f_2, a_2, f_1, a_1 - a_2) \tag{6}$$

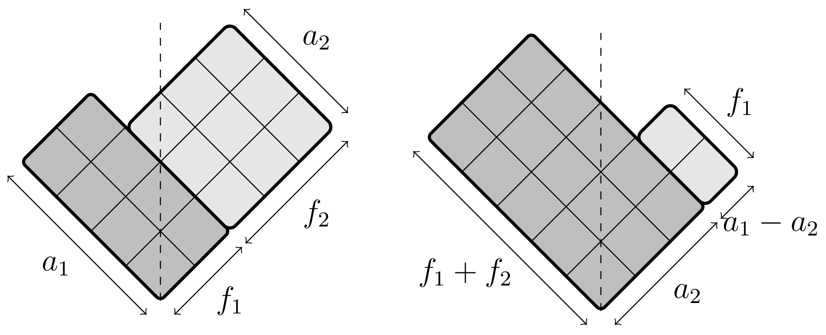


Figure 6: Young diagram corresponding to an integer partition of $p = 17$ with two parts, and its conjugate.

It is straightforward to check that τ has exactly one fixed point. Indeed, the conditions for being a fixed point are

$$a_1 = f_1 + f_2 \ \ \&\ \ f_1 = a_2 \tag{7}$$

so that expressing a_1, a_2 in terms of f_1, f_2 gives $p = f_1(f_1 + 2f_2)$. It follows that $f_1 = 1$, which in turns determines $f_2 = \frac{p-1}{2}$. Note that this part of the argument only relies on p being an *odd* prime number. The conclusion is that $|\mathscr{P}_2^p|$ is odd.

Another natural idea is to swap each pair a_1, f_1 and a_2, f_2, but as such it does not define an involution on $\mathscr{P}_2^p$: for instance, if $f_1 = f_2$ then the outcome is not in $\mathscr{P}_2^p$ as it is not a partition with two parts, but one. However, this idea works on well-defined subsets. We consider the following partition of $\mathscr{P}_2^p$:

$$\mathscr{P}_2^p = \mathscr{P}_{2,>}^p \sqcup \mathscr{P}_{2,=}^p \sqcup \mathscr{P}_{2,<}^p \tag{8}$$

where

$$\mathscr{P}_{2,<}^p := \{(a_1, f_1, a_2, f_2) \in \mathscr{P}_2^p \mid f_1 < f_2\} \tag{9}$$

$$\mathscr{P}_{2,=}^p := \{(a_1, f_1, a_2, f_2) \in \mathscr{P}_2^p \mid f_1 = f_2\} \tag{10}$$

$$\mathscr{P}_{2,>}^p := \{(a_1, f_1, a_2, f_2) \in \mathscr{P}_2^p \mid f_1 > f_2\} \tag{11}$$

and we make a few straightforward observations.
First, concerning (9), we note that the double-swapping

$$(a_1, f_1, a_2, f_2) \mapsto (f_2, a_2, f_1, a_1)$$

defines an involution of $\mathscr{P}^p_{2,<}$ that does not have a fixed point (such a fixed point would imply $p = 2a_1 f_1$, which is absurd). We conclude, invoking Lemma 2, that $|\mathscr{P}^p_{2,<}|$ is even.

The next observation is that the set $\mathscr{P}^p_{2,=}$ defined by (10) can be enumerated directly; as p is prime, an element of $\mathscr{P}^p_{2,=}$ must be such that $f_1 = f_2 = 1$. As there are exactly $(p-1)/2$ ways of writing $p = a_1 + a_2$ with $a_1 > a_2$, it follows that

$$|\mathscr{P}^p_{2,=}| = \frac{p-1}{2},$$

which is even. A partial conclusion at that point is that $|\mathscr{P}^p_{2,>}|$ is odd.

The last observation is that the simple swapping

$$(a_1, f_1, a_2, f_2) \mapsto (f_1, a_1, f_2, a_2)$$

defines an involution on $\mathscr{P}^p_{2,>}$, a finite set of odd cardinality by the previous arguments. By Lemma 2, it must have a fixed point, which implies $a_1 = f_1, a_2 = f_2$, and therefore $p = a_1^2 + a_2^2$, a sum of two squares. This concludes David Christopher's proof.

3.2 Formal verification of David Christopher's proof

A formal verification of this second proof can be found in the third part of the master file `all_Zagier_project.v`, or equivalently `proof_B_partition.v` after compiling the preliminary file `lemmata.v`. It begins with the following transparent definitions; clearly, the partitions of p that are considered involve quadruples of integers, so that `K` is here $\mathbb{N}^4$.

```
Definition N4 : Type := (nat * nat) * (nat * nat).
Definition Ip4:{fset N4} := ((Ip`*`Ip)`*`(Ip`*`Ip)).
Definition involutionN4:= (@involution_on [choiceType of N4]).
Definition fixed_fsetN4:=(@fixed_fset [choiceType of N4]).
Definition involutionN4_lemma:=(@involution_lemma [choiceType of N4]).
```

The quantity $a_1 f_1 + a_2 f_2$ is called `Y_area`, referring to the area of a Young diagram.

```
Definition Y_area (yd : N4) : nat := yd.1.1*yd.1.2+yd.2.1*yd.2.2.
```

Finally, here are the definitions of the sets considered above, properly defined as objects of type `{ fset N4 }` thanks to an *a priori* bound.

```
Definition P2p :=[fset yd:N4 | (yd \in Ip4) && ((Y_area yd ==p)&&(yd.1.1>yd.2.1))].
  Definition P2p_l :=[fset yd:N4 | (yd \in P2p) && (yd.1.2< yd.2.2)].
  Definition P2p_e :=[fset yd:N4 | (yd \in P2p) && (yd.1.2==yd.2.2)].
  Definition P2p_g :=[fset yd:N4 | (yd \in P2p) && (yd.1.2>yd.2.2)].
```

The formal proof then follows the argument explained above in a rather straightforward way. The general lemma on involutions and the ad hoc tactics like `mcnia` and `zag_solve` written for Zagier's proof are used again, to the same effect.

The final proof, written in this way, is longer than the one based on Zagier's argument, and this for two reasons: obviously, it requires to introduce more objects (sets and involutions) and to go through more steps; but also, there is one argument of a different kind needed to enumerate the middle set $\mathscr{P}^p_{2,=}$. From the point of view of formal verification, this is the only difference compared to Zagier's proof, which only relied on involutions. In order to perform this enumeration, we first define an injective function that generates quadruples from natural numbers as follows:

$$q(n) := (p - (n + 1), 1, n + 1, 1)$$

We then prove that $\mathscr{P}_{2,=}$ is equal to the set generated by mapping q over the set of natural numbers less than $\lfloor (p - 1)/2 \rfloor$, which MathComp expresses as `I_((p-1)%/2)`. The cardinality of that set is of course known from the libraries.

Conclusion

We have formally verified two short combinatorial proofs of Fermat's sum of two squares theorem, using the Coq proof assistant. It was to be expected that the final result would not exactly match the length of Zagier's 'one-sentence proof' when presented informally, even together with its geometric illustration. However, we feel that these formal proofs are not outrageously long either, considering the level of exactness they warrant – and they certainly could be further shortened, if needed to make a point. More importantly, the feeling that a formal proof exactly matches the main mental steps required by the original argument, and is not something intrinsically different - but only more precise - is very satisfying.

For both authors, this project was a first encounter with the Mathematical Components libraries, and for one of them, a first serious endeavour with the Coq proof assistant altogether. It would have taken a lot more time and effort without the recurrent help and detailed explanations of experts in the community; especially, one can only advise new users in a similar situation to express their concerns online as soon as they encounter a difficulty. It is also a somewhat dull but necessary task to read the documentation of each library very carefully to avoid issues or resolve them quickly. In particular, one should be careful to not only import the right libraries but also to *open the right scopes in the right order*. For the sake of a (real-life) example: in order to define and use objects of type `{fset K }`, it is necessary to first write

```
Open Scope fset_scope.
```

However, one of the side-effects of this scope is to overwrite the sign `+` usually used for addition, so that it is not possible anymore to write expressions like `a+b` with `a b : nat` right after opening `fset_scope`. One should *re-open* the `nat_scope` first:

```
Open Scope fset_scope.
Open Scope nat_scope.
```

Another important caveat is that the Search function, useful as it is, does not necessarily yield all relationships between concepts, in particular in cases where one concept is defined in terms of another. For example, the divisibility predicate `dvdn d m` is defined using the modulo operation (as `modn m d == 0`). Writing `Search _ modn dvdn.` does not immediately provide access to this information; instead, one needs to look directly at the definition of `dvdn`.

The above aspects are easy enough to work with, once noticed. But subtleties of that sort are perhaps the most time-consuming aspect when discovering the libraries. The sooner they are resolved, the sooner one can actually work on relevant aspects of the code itself.

References

[1] F. Besson, *Fast Reflexive Arithmetic Tactics the Linear Case and Beyond*, Types for Proofs and Programs, 2007, pp. 48–62.

[2] J. Chan, `https://bitbucket.org/jhlchan/project/src/master/fermat/twosq/`.

[3] Coq Development Team, *The Coq Reference Manual, Release 8.13.1* (2021), `https://coq.inria.fr/distrib/current/refman/`.

[4] A. David Christopher, *A partition-theoretic proof of Fermat's Two Squares Theorem*, Discrete Mathematics **339** (2016), no. 4, 1410 - 1411.

[5] G. Gonthier and A. Mahboubi, *An introduction to small scale reflection in Coq*, Journal of formalized reasoning **3** (2010), no. 2, 95–152.

[6] G. Gonthier, A. Mahboubi, and E. Tassi, *A small scale reflection extension for the Coq system*, 2016.

[7] D.R. Heath-Brown, *Fermat's two squares theorem*, Invariant (1984), 3-5.

[8] C. Hughes (2019). `https://github.com/leanprover-community/.../sum_two_squares.lean`.

[9] A. Mahboubi and E. Tassi, *Mathematical components*, 2017.

[10] `https://github.com/math-comp/mczify`.

[11] A. Spivak, *Крылатые квадраты (Winged squares)*, Lecture notes for the mathematical circle at Moscow State University, 15th lecture (2007).

[12] L. Théry (2016), `https://github.com/thery/twoSquare`.

[13] D. Zagier, *A One-Sentence Proof That Every Prime p≡ 1 (mod 4) Is a Sum of Two Squares*, The American Mathematical Monthly **97** (1990), no. 2, 144–144.

[14] `https://coq.zulipchat.com/`.