Using small scale automation to improve both accessibility and readability of formal proofs in geometry

Pierre Boutry, Julien Narboux, Pascal Schreck, Gabriel Braun

▶ **To cite this version:**

Pierre Boutry, Julien Narboux, Pascal Schreck, Gabriel Braun. Using small scale automation to improve both accessibility and readability of formal proofs in geometry. 2014. hal-00989781v1

HAL Id: hal-00989781

https://inria.hal.science/hal-00989781v1

Preprint submitted on 12 May 2014 (v1), last revised 25 Jun 2014 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Using small scale automation to improve both accessibility and readability of formal proofs in geometry

Pierre Boutry, Julien Narboux, Pascal Schreck, and Gabriel Braun

ICube, UMR 7357 CNRS, University of Strasbourg
Pôle API, Bd Sébastien Brant, BP 10413, 67412 Illkirch, France
{pierre.boutry, narboux, schreck, braun}@unistra.fr

Abstract. This paper describes some techniques to help building formal proofs in geometry and at the same time improving readability. Rather than trying to completely automate the proving process we provide symbolic manipulations which are useful to automate the parts of the formal proof that are usually implicit in a pen and paper proof. We test our framework using some well known theorems about triangles which are taught in high-school. We also highlight the proof steps which are usually overlooked in the informal proof and that we believe should be made explicit. Our framework is based on Tarski's geometry within the Coq proof assistant, but most of the ideas presented in this paper could be applied to other axiomatic systems or proof assistants.

Keywords: formal proof, geometry, automation, Tarksi, Coq

1 Introduction

In this paper, we propose techniques to help building formal proofs in geometry interactively using a proof assistant like Coq. The following research was conducted as part of a larger project which focuses on the use of proof assistants in an educational framework to teach reasoning principles. We believe that geometry is a good subject to study in order to understand the proving process. Our long term goal, consists in the integration of a tool to prove theorems in geometry with a dynamic geometry software like GeoGebra. Previously, the second author and Tuan-Minh Pham have proposed prototypes of graphical user interfaces (GUIs) to build proofs interactively [25,28]. Unfortunately, for several reasons that we will described in this paper, obtaining a formal proof in geometry is not as easy as it should be and these prototypes can not be used in a high-school classroom. While building a formal proof we are facing two main problems: first the formal proving language is difficult to write and read and second the amount of details which has to be given to the interactive prover is too big. In this paper we focus on the second problem. But also, by reducing the amount of details provided to the system we also improve readability of the proof to some extend. Indeed even if we had an elegant proof language which

has a nice output in natural language with beautiful notations, if the proofs contains too much boring details it becomes difficult to read. We do not focus in this paper on the proof language nor the notations.

There are of course several previous works about readability of formal proofs: for instance Y. Coscoy *et al.* in [14] translate low-level proof into readable text while A. Holland-Minkley *et al.* in [18] use built-in tactics in Nuprl [1] translated into English sentences to structure the text of the proof. S. Stojanović *et al.* in [30] aim at producing readable proof within a fully automatic prover based on coherent logic. Some of these works also focus on the problem of hiding unimportant details. One of our goals is then to help the student obtaining the proof, hence we can not use the previous approaches by *a posteriori* transformation of the proof.

Despite the fact the title of this paper is inspired by language SSReflect [17], our work aims at very different goals. SSReflect is a powerful language to help a specialist describing complex proofs, accessibility and readability is not their main motivation.

Geometry is a successful area of the field of automated theorem proving. Several efficient methods have been developed. The most popular ones are the area method [12], the Gröbner basis method [9] and Wu's method [34,10]. It is to be noticed that a decision procedure for the theory we are using was given by Tarski [31]. We formalized some of these methods in Coq [23,19,16]. But in this paper, as we aim at applications in the education, we are not interested in obtaining the most powerful prover as possible which automate the whole proof, we want to automate only the proof steps which are usually implicit in a pen and paper proof.

We designed and implemented some tools integrated into a proof assistant such that a student can produce a proof more easily. Then, the result, which is the proof script is more readable than what we had before. Of course, in order to produce a proof script in natural language, in a similar fashion to [18], some work remains to be done. Here we do not focus on proof syntax but more on small scale automation to hide administrative proof steps. We identify some key points which make geometric proofs difficult to perform and to read. Then, we design and implement some simple yet powerful symbolic manipulations to handle these difficulties.

As a case study, we consider in this paper euclidean geometry. We use an axiomatic system with good meta-theoretical properties, namely Tarski's geometry. Our work is based on the set of axioms as given by W. Schwabhauser, W. Szmielew and A. Tarski in [29] and the formal development in Coq described previously [26,8] which provides the main definitions and hundreds of lemmas.

The rest of the paper is organized as follows. Section 2 presents a plain example in elementary geometry. Section 3 presents issues coming from incidence properties. In section 4, describes a way to take degenerate condition into account. Section 5, presents tactics to deal with allowed permutations of arguments of a proposition. We give in section 6 some examples to illustrate our approach.

Finally, section 7 concludes this paper by summarizing our work and by giving some ideas for future work.

2 A plain theorem

For the sake of clarity, we use throughout the next sections, the same plain example that illustrates most of the notions we want to describes in this paper. The statement of this theorem is the following.

Theorem 1. *In a triangle ABC where P is the midpoint of BC and Q the midpoint of AC then the lines AB and PQ are parallel.*

First we give the informal proof which is often given in class.

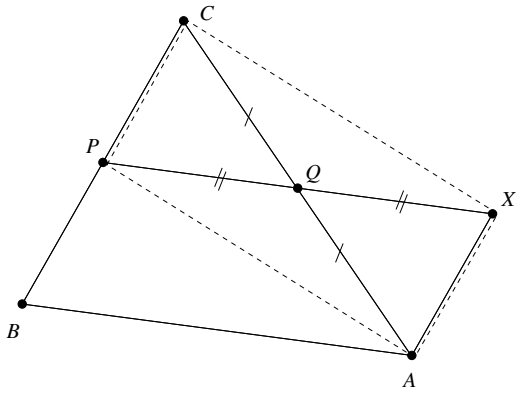

Fig. 1. The midpoints theorem.

Proof. We first construct point X as the symmetrical point of P with respect to Q. Point Q is therefore the midpoint of segment PX. From the assumptions we know that Q is also the midpoint of AC. Thus, the diagonals of the quadrilateral $APCX$ bisect in their midpoint and hence $APCX$ is a parallelogram. Now according to the fact that the opposite sides of a parallelogram are parallel and have the same length, we have that AX and CP (or BP) are parallel and $AX \equiv CP$. As we know that P is the midpoint of BC we also have $AX \equiv PB$. The quadrilateral $AXPB$ has two opposite sides which are parallel and of the same length, hence it is a parallelogram. Finally the opposite sides of a parallelogram are parallel, thus AB and PQ are parallel. □

We will see in the next sections that formalizing this simple proof is not as trivial as it seems. For a good reason: actually, this proof is not correct because from the fact that the opposite sides of a quadrilateral $ABCD$ are parallel and of

the same length we can only conclude that $ABCD$ or $ABDC$ is a parallelogram, proving that one is a parallelogram rather than the other is not trivial and should not be omitted. Second for some bad reasons: this proof contains implicit proof steps that a proof assistant force us to detail. In this paper we describe tools to help us distinguish between implicit proof steps which are not important and should be hidden from missing argument in the proof which should not be forgotten.

The proof steps which we want to hide are from three categories that will be studied in the next sections:

1. some pseudo transitivity properties of some predicates especially the collinearity predicate
2. some non degeneracy conditions
3. some symmetry properties of geometric predicates

3 A reflexive tactic for dealing with pseudo-transitivity of collinearity

The question of collinearity, or in other words of dependence/independence of vector or points, is striking in geometry. This section presents our approach about this issue.

3.1 The issue

Since there is no primitive type *line* in Tarski's geometry, the concept of incidence is formalized by using a predicate of arity 3 called Col predicate which expresses the fact that the three points belongs to the same line. Note that the points do not need to be distinct. This predicate verifies the following lemmas:

$$\forall XYZT,\ X \neq Y\ \wedge\ Col(X,Y,Z)\ \wedge\ Col(X,Y,T)\ \Rightarrow\ Col(X,Z,T).$$

$$\forall XYZ,\ Col(X,Y,Z) \Leftrightarrow Col(\sigma(X),\sigma(Y),\sigma(Z)) \text{ for every permutation } \sigma.$$

In practice, formal proofs then contain often dozens of applications of that lemmas which are useless for understanding the proof.

For instance, in our example, when one wants to prove that points A, P and C are not collinear by contradiction, one has to suppose that these points are collinear and then prove that points A, B and C are collinear too. To do so, we have to exploit the fact that the midpoint of two points is collinear with these points. Then, let us consider the following hypothesis: $Col(C,P,B)$, $Col(A,P,B)$ and $P \neq B$. To prove that $Col(A,B,C)$, we need to apply lemmas of the second family several times and the first lemma one time. For a human being, it is enough to see that the four points are on the same line. Our formal development contains many other examples where things are worse for instance with a line with ten points on it and a lot of applications of the previous lemmas to do.

In order to simplify the proving process and to improve readability, we defined a tactic which can prove automatically collinearity properties which are

consequence of this pseudo-transitivity. Our first approaches to deal with this problem was to use the automation of Coq (namely to create a base of hints for `eauto`) and then to write an *ad hoc* tactic in the tactic language of Coq. However this approach was not fulfilling our needs as it could not cope with difficult problems in a reasonable time. We therefore opted for a different approach. We chose to implement a reflexive tactic to handle this problem. First introduced in Coq by Samuel Boutin [6], reflexivity consists in replacing a tactic by an algorithm written in the Coq language and proving that the algorithm is correct. The reader interested in learning more about this now standard approach can also read the last chapter of the CoqArt [5]. Using a reflective tactic allows us not only to save the user from doing the tedious work about the pseudo transitivity of collinearity but also it hides these steps from the proof term. The algorithm used by this tactic is described in the following paragraph.

3.2 Algorithm

The algorithm is divided into three parts. The first one consists in the initialization phase: it computes the set of all the sets of points provably collinear and the sets of pairs of points provably distinct. The second part consists in updating our internal data structure to compute the sets of points on each line. Finally we check if three given points are collinear by testing if they belong to a single line. For our algorithm we need a set of sets of positive numbers $\mathcal{L}$ to represent the sets of points provably collinear and a set of pairs of positive numbers $\mathcal{D}$ to represent the points provably different.

The algorithm is as follows:
INPUT: 3 points A B C and the current hypotheses.

1. Initialize $\mathcal{L}$ so that it contains all sets of 3 points that are given collinear by the hypotheses in the context and $\mathcal{D}$ so that it contains all the pairs of points that are given different by the hypotheses in the context.
2. For every l_1 and l_2 in $\mathcal{L}$ such that there exists a pair p in $\mathcal{D}$ such that $p \subseteq l_1 \cap l_2$, replace $\mathcal{L}$ with $((\mathcal{L} \backslash l_1) \backslash l_2) \cup \{l_1 \cup l_2\}$ until there are no such l_1 and l_2.
3. If there is a set l in $\mathcal{L}$ such that $A \in l$, $B \in l$ and $C \in l$ then A, B and C are provably collinear.

Remark. Our tactic only captures basic properties of incidence, and it is complete for only a small theory described below. Indeed, it can happen that some points A, B and C are collinear (if this fact follows from other geometric theorems) and our tactic fails in yielding a set $l \in \mathcal{L}$ such that $A \in l$, $B \in l$ and $C \in l$.

3.3 Implementation

We give now some technical details about the implementation in Coq of our algorithm.

Datastructures We need to represent sets of sets of points. To represent points we need a decidable ordered type, hence we use the type of positive numbers as keys. To represent finite sets we use the module `Msets` of the standard library. We could have used the library `Containers` [21] which is easier to use than `Msets` because it infers automatically the structures needed to build the finite sets but we chose to keep with the standard `Msets` to make our development easier to install. We selected the implementation using ordered lists. Notice that using AVLs is not interesting here since we rarely have more than 30 points.

The tactic First, our tactic follows the first step of our previous algorithm in order to build the sets $\mathcal{L}$ and $\mathcal{D}$ by using an associative list so that the positives in our structures identify points. This initialization phase is implemented using the tactic language of Coq.

The second step is implemented as a Coq function defined using the `Function` package of Coq. To convince Coq that the algorithm terminates we proved the fact the cardinal of $\mathcal{L}$ decreases at each recursive call.

The third step is also implemented as a Coq function which search for triple of points in a same set contained in $\mathcal{L}$.

Proof of the correction of our tactic For the sake of modularity, we created a type class with the minimal set of properties that a theory needs to verify and we did all the proofs within the context of this type class. Our tactic is then applicable to any theory verifying this following four properties:

```
Class Col_theory (CTpoint:Type) (CTCol:CTpoint->CTpoint->CTpoint->Prop):=
{
 CTcol_trivial : forall A B : CTpoint, CTCol A A B;
 CTcol_perm_1 : forall A B C : CTpoint, CTCol A B C -> CTCol B C A;
 CTcol_perm_2 : forall A B C : CTpoint, CTCol A B C -> CTCol A C B;
 CTcol3 : forall X Y A B C : CTpoint,
       X <> Y -> CTCol X Y A -> CTCol X Y B -> CTCol X Y C -> CTCol A B C
}.
```

We want to prove that the tactic produces a $\mathcal{L}$ that verify the property "*any triple of point belonging to a set of* $\mathcal{L}$ *are provably collinear*". To do so we prove that the $\mathcal{L}$ produced by the first step of our algorithm verifies this property and that the the second step of the algorithm preserves this property. The original $\mathcal{L}$ trivially verifies this property by construction. We denote by $\overline{x}$ the positive integer representing x. Now assuming that we have l_1, l_2, p_1 and p_2 verifying $\overline{p_1} \in l_1 \cap l_2$, $\overline{p_2} \in l_1 \cap l_2$ and $(\overline{l_1}, \overline{l_2}) \in \mathcal{D}$. Assuming that any triple of point in l_1 are provably collinear and assuming the same for l_2 then for any p_3 in l_1, $Col(p_1, p_2, p_3)$ holds and for any p_4 in l_2, $Col(p_1, p_2, p_4)$ holds. By the lemma stated previously any triple of point in $l_1 \cup l_2$ are provably collinear. This proves that the second step of our algorithm preserves the property stated above and at the end of the second step we will obtain a $\mathcal{L}$ verifying this property.

3.4 Generalization

This algorithm may seem to be very specific. In fact, it can be generalized to deal with other properties than pseudo transitivity of collinearity. For example, the lemma to express the pseudo-transitivity of the con-cyclic predicate has the same form:

$$\neg Col A\ B\ C \Rightarrow concyclic\ A\ B\ C\ P \Rightarrow concyclic\ A\ B\ C\ Q \Rightarrow concyclic\ A\ B\ P\ Q$$

In fact, it is generalizable to any incidence relationship with algebraic curves or affine varieties.

3.5 Relation with equality of lines and rank functions

If we allow ourself the concept of line (either by defining it with Tarski's language or by using another language for geometry which includes lines such as Hilbert's axioms), then we can rewrite the pseudo transitivity property of Col as an equality properties about lines:

$$A \neq B \wedge A \in l \wedge B \in l \wedge A \in m \wedge B \in m \Rightarrow l = m$$

At first sight this property looks nicer than our properties about Col, but the problem with this formulation is that lines are always defined by pairs of distinct points. In practice using this kind of formulation would imply numerous case distinctions about equality of points.

There is a close link between the concept of rank that we formalized previously [22] and the properties studied in this section. The rank of a subset S of elements of the matroid of points is the maximum size of an independent subset of S. Notice that the sub-modularity property of the rank function is a generalization of the pseudo transitivity of Col:

$$r(l \cup m) + r(l \cap m) \leq r(l) + r(m)$$

Indeed, if l and m are lines then their rank is 2, and if their intersection contains two distinct points then the rank of the intersection is at least 2, hence every triple of points in the union is collinear as:

$$r(l \cup m) \leq 2 + 2 - 2 = 2$$

4 Non degeneracy conditions

It has been highlighted several times in the literature that dealing with non degeneracy conditions (NDC or NDCs in short) is a challenge while developing formal proofs in geometry [15,24]. Chou and Gao[11] have noted that :

> "It is our experience that finding non degenerate conditions for a geometry statement is not easy."

Non degeneracy conditions can be defined as negative assumptions of the form $\neg A = B$ or $\neg ColABC$ or $\neg AB \parallel CD$. They are often overlooked in pen and paper proofs and can be a source of errors. Concretely we can distinguish several kinds of problems related to NDCs which occur during the development of a formal proof in geometry:

1. finding the NDCs needed for the validity of the conjecture we want to prove (as noted by Chou and Gao)
2. finding the optimal definitions for the geometry predicates
3. proving the NDCs needed by the lemma we use during the proof, this involves:
   (a) deriving some NDCs from some other NDCs
   (b) proving some NDCs by case distinction

The first issue has been addressed in the literature previously using algebraic approaches such as Wu's method, it is possible to generate automatically the NDCs of a given statement. But Wu's method is efficient only when the statement is given as a construction, applying the method in our context would require to solve a geometric constrains problem to find automatically the construction.

The first two issues are related to the definition of a *geometric universe* by an expert for whom we propose a methodology and the last one concerns more directly a student using a proof assistant to prove a geometric theorem. The next sections give more details on these issues.

4.1 Finding the NDCs needed for the validity of the conjecture

It is not trivial to find out what are the necessary NDCs of a theorem. Let's consider our example. What are the NDCs needed? we could run an algebraic method to find out the answer (Dongming Wang provides an algorithm to search for the minimal set of NDCs[32]).

But a careful analysis shows that in fact we can derive two different statements, the first one has minimal NDCs assumptions:

```
Lemma triangle_mid_par : forall A B C P Q,
 A <> B ->
 is_midpoint P B C ->
 is_midpoint Q A C ->
 Par A B Q P.
```

Note that $P \neq Q$ follows from the fact that $A \neq B$ and the other assumptions. This statement is very general because it can be applied to a degenerated triangle when the points ABC are collinear and even when the points B and C are equal. However the properties which are asserted are weak because it does not guaranty that the lines AB and CD are distinct. Therefore, in some situations it is better to have a statement with stronger NDCs but which provides also a stronger conclusion:

```
Lemma triangle_mid_par_strict : forall A B C P Q,
 ~ Col A B C ->
 is_midpoint P B C ->
 is_midpoint Q A C ->
 Par_strict A B Q P.
```

In our formalization we need both theorems. Note also that the NDCs of a statement depends on the way the statement is formulated. Stating a theorem by the mean of a ruler and compass construction can lead to spurious NDCs.

Overall, in our development we try to force ourselves to find the minimal NDCs and the most general definition for the lemmas. We also chose to provide statements described using geometric constraints rather than geometric constructions .

4.2 Finding the optimal definitions for the predicates

Another important issue regarding NDCs is the definition of geometric notions. Experience shows that one should try to give the most general definition as well as more specific definitions. For example our development contains two predicates for the concept of parallel lines, a strict one and a broad one which includes the case when the line are equal. Consider as another example the definition of the parallelogram. What should it be? Should we exclude degenerated parallelogram?

In our formal development we have two definitions: first the strict parallelogram which ensures that the four points are not collinear (for this strict version all the standard definitions are equivalent) and second, the parallelogram in the broad sense. The most general definition we could find is the following:

$$ABCD \text{ is a parallelogram} =_{def} \begin{array}{l}(A \neq C \ \vee \ B \neq D) \wedge \\ \exists M, M \text{ is the midpoint of AC and BD}\end{array}$$

4.3 Proving NDCs

Another problem which appears often is to prove the NDCs of the lemmas or theorems we use in the proof. For example, if we want to use the theorem `triangle_mid_par_strict` for some triangle PQR, we have to show that P, Q and R are not collinear. In practice, it is not always provable, it may well be the case that PQR can be collinear, if this is the case we have to perform a *case distinction*[1]. To address this issue we have tactics which try to compute the trivial consequences of the negation of a NDC. For NDCs of the form $\neg Col(A, B, C)$ we have the tactic described in Section 3. For NDCs of the form $A \neq B$ we have the tactic `treat_equalities`. This tactic applies repeatedly theorems which shows the equality of two points until it reaches a fix-point. This often allows to

[1] Note that recent results of Michael Beeson [4] shows that in some constructive variant of Tarski's axiom system these case distinctions can be avoided.

'collapse' the figure in such a way that it appears that some NDCs are contradicted. Table 1 shows the list of theorems currently used. Note that this list is updated when we extend the formal development.

1. $AB \cong CC \Rightarrow A = B$
2. $bet(A, B, A) \Rightarrow A = B$
3. $bet(A, B, C) \Rightarrow bet(B, A, C) \Rightarrow A = B$
4. $is\_midpointI\ A\ A \Rightarrow I = A$
5. $is\_midpointI\ A\ B \Rightarrow is\_midpoint\ I\ A\ B' \Rightarrow B = B'$
6. $is\_midpointI\ A\ B \Rightarrow is\_midpoint\ I\ A'\ B \Rightarrow A = A'$
7. $is\_midpointI\ A\ B \Rightarrow is\_midpoint\ I'\ A\ B \Rightarrow I = I'$
8. $is\_midpointA\ B\ A \Rightarrow A = B$
9. $is\_midpointA\ A\ B \Rightarrow A = B$

Table 1. Theorems used by the tactic `treat_equalities`.

Based on these tactics, we have developed some tactics to prove automatically some NDCs conditions. The tactic `assert_ndc_by_contradiction` use the following algorithm: for every couple of points (A, B) in the context: Assume that $A = B$, deduce other equalities and simplify assumptions using the tactic `treat_equalities`. For every couple of points (P, Q) such that $P \neq Q$ or triple of points (P, Q, R) such that $\neg Col(P, Q, R)$ try do find a contradiction. For finding the contradiction we try to prove that $P = Q$ or $Col(P, Q, R)$ using the tactic described previously. The tactic `assert_all_diffs_by_cases`, shows that *without loss of generality* we can assume that some points are distinct. This tactic uses reasoning by cases on the equality of points: for every couple of points (A, B) in the context it assumes $A = B$, and deduce other equalities using the tactic `treat_equalities` and try to solve the goal using a tactic which can solve some simple goals. Compared to the tactic `wlog` of SSReflect our tactic can be seen as a specialized (it adds only assumptions of the form $A \neq B$ to the context) but automatic version which knows about geometry.

These tactics are based on a sub-layer of simple tactics called `assert-congs`, `assert_cols`, `assert_bet` which put in the context some trivial consequences of the hypotheses which are used by other higher-level tactics.

Overall these heuristics allow to reduce significantly the size of the proof and the energy spent to construct it.

5 Application modulo permutations

It often happens that *administrative* lemmas about permutation properties of some predicates are needed in a formal proof when one has to apply a high level geometric theorem. This makes the proof more difficult to perform and to read. For example, let us consider the triangle midpoints theorem:

```
triangle_mid_par : forall A B C P Q : Tpoint,
  A <> B -> is_midpoint P B C -> is_midpoint Q A C -> Par A B Q P
```

If we know that :

```
H1 : is_midpoint G B D
H2 : is_midpoint I A B
H3 : D<>A
```

and we want to show that :

```
Par G I D A.
```

On paper, the triangle midpoints theorem is applied straightforwardly. But when using a proof assistant, say in backward chaining, we have first to reduce the goal to `Par A D I G` and then to prove that `is_midpoint G B D` implies `is_midpoint G D B` and `D<>A` implies `A<>D`. Without user-defined tactics, this administrative reasoning steps could be formalized using the following proof script:

```
cut (Par A D I G).
intro.
apply par_left_comm.
apply par_right_comm.
apply par_symmetry.
assumption.
apply (triangle_mid_par A D B G I).
assumption.
apply l7_2.
assumption.
assumption.
```

This script is not only unreadable it is also long to obtain it because even if it is simple it takes time to guess what is the needed permutation of the parallel predicate.

We present in this section a small tactic `perm_apply` written in `Ltac` which allows us to apply a lemma modulo some commutativity properties such that one can solve our example by simply using the following command:

```
perm_apply (triangle_mid_par A D B G I).
```

Solving the "apply modulo" problem in general requires implementing an algorithm for unification modulo some theory. This problem has been studied in depth in the literature [3,20]. The unification modulo associativity and commutativity has been studied especially and even been implemented in some proof assistant such as Coq [7] and Isabelle [27]. We cannot directly reuse the tactic developed by T. Braibant and D. Pous for rewriting modulo AC because: first the

tactic can not deal with application of a lemma but only with rewriting and second the tactic can only deal with binary operations whose type is $T \to T \to T$ for some type T. Moreover in our case study we are facing a different problem than matching modulo AC because we have to match modulo some permutations. Our axioms are of the form $f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ where π is a permutation on $\{1, \ldots, n\}$. Jürgen Avenhaus has proved that unifiability-modulo is NP-complete for theories of this form [2] and has proposed efficient algorithms to test equalities modulo permutations.

We do not implement the algorithms proposed in the literature, because our problem is simpler. In our context of Tarski's geometry we do not have any function symbols, hence the depth of the atoms we are manipulating is constant and equal to one. Moreover the number of arguments of our predicates is small (≤ 6). The solution we propose is very simple and consists in pre-computing the permutation group.

For instance for the predicate `Par` have the following lemma:

```
Lemma Par_cases :
  forall A B C D,
  Par A B C D \/ Par B A C D \/ Par A B D C \/ Par B A D C \/
  Par C D A B \/ Par C D B A \/ Par D C A B \/ Par D C B A ->
  Par A B C D.
```

Then, using a simple tactic which introduces the disjunction and try using every branch, we can deduce a tactic to apply a lemma modulo some permutations. Our `perm_apply` tactic succeeds if we have the required permutations of the hypotheses in the context. Currently the tactic can deal with the permutations properties of the following predicates: `Par`, `Par_strict`, `Perp`, `Perp_in`, `Per`, `Col`, `is_midpoint` and `Bet`.

Generalization to other geometries

Note that if we had a framework involving function symbols this would make this tactic harder to implement. For example, in a geometry such as Hilbert's geometry which involves points and lines as primitive objects we could have a predicate `Par_lines` of type `Line -> Line -> Prop` and a function symbol to create a line from two points `C_line` of type `Point -> Point -> Line`[2] and a function symbol to create a parallel line to another line going through a point, `C_par` of type `Point->Line->Line`. In this world, the permutation properties of `Par` can be expressed by the commutativity of the `C_line` and `Par_lines` binary operations. But the function symbols can be arbitrarily nested, hence it is possible to write `Par_lines (C_line A B) (C_par (C_line C D) E)` which states that line AB is parallel to the parallel to CD going through E.

[2] We omit here the issue about the fact that the two points should be distinct.

6 Examples

In this section we provide a comparison of the informal and formal proofs for two examples.

The tactics `assert_paras`, `assert_pars` and `assert_cols` deduce trivial properties which are direct consequences of the assumptions. For example if we now that P is the midpoint of a segment AB we do not want to keep implicit the fact that A, B and P are collinear. In an educational context, the quantity of facts which are used implicitly could vary depending on the curriculum of the country and the age of the students.

The readability of the proof could be further improved by using a declarative proof language such as the Mizar language, Isar [33] or Czar [13] but in this paper we do not focus on this issue.

The text written in bold face is the part of the informal proof which has been suggested by the formal proof, which is often omitted in a pen and paper proof and that we think should not be omitted. Figure 2 depicts the example we have described in Section 2. Figure 3 shows the first version of the formal proof of the midpoint theorem which do not use the tools described in this paper (still this first proof use some automation to deal with equality and basic properties of `Col` and `Par` predicates). Figure 4 consists in the proof that the three medians of a triangle intersect in a single point. Figure 5 provides the sketch.

7 Conclusions and Future Work

We presented in this paper some techniques to ease the construction and improve the readability of formal proofs in geometry. We believe that in order to obtain simple proofs, not only general progress about proof languages should be made but also some *ad hoc* techniques specialized in the domain of study should be used. For instance, in analysis tactics to deal with proof involving epsilons could be used. In this paper we described some heuristics for high-school geometry.

Since we are interested in pedagogical applications of proof assistants, we explored tools aiming at next step guidance and yielding readable proofs. To do that, we used different methods:

- using tactics in order to automatically apply low-level lemmas and hide them
- using reflexive functions to deal with combinatorial explosions when dealing with basic incidence axioms
- using tactics which help to prove dis-equalities when dealing with degenerate cases

Most of the methods presented in this paper can be generalized to other contexts than Tarski's geometry and/or Coq framework.

Our study suggests several lines of research for future work. For instance, we deal here with collinearity as an avatar of point-line incidence relationship, but our approach can be generalized to incidence between points and varieties. Also, the application of a theorem modulo some permutation use a particular

```
Lemma
 triangle_mid_par_strict_cong_simp :
 forall A B C P Q,
 ~ Col A B C ->
 is_midpoint P B C ->
 is_midpoint Q A C ->
 Par_strict A B Q P.
Proof with assert_all.
intros.
Name X the symmetric of P wrt Q...
assert (~ Col A P C)
 by (intro; search_contradiction)...
assert (Parallelogram_strict A P C X)
 by (apply mid_plgs with Q;finish)...
assert_paras.
assert_pars.
assert (Cong A X B P)
 by eCong.
assert (Par A X B P)
 by (apply par_col2_par with P C;
     finish).
assert (HElim :
        Parallelogram A X B P \/
        Parallelogram A X P B)
by (apply par_cong_plg_2; assumption).

induction HElim.

- (* Impossible case. *)
  Name M the intersection of
       the diagonals (A B) and
       (X P) of the parallelogram H25.
  treat_equalities.
  search_contradiction.

- assert_paras.
  assert_pars.
  assert (Par P Q A B)
   by (apply par_col_par_2 with X;
       finish).
  apply
   par_not_col_strict with X;finish.
  intro.
  assert_cols.
  apply H. ColR.
Qed.
```

We first construct point X as the symmetrical point of P with respect to Q. Point Q is therefore the midpoint of segment PX. From the assumptions we know that Q is also the midpoint of AC. **On the other hand, points A, P and C are not collinear because assuming they are we could conclude that A, B and C are collinear which contradicts the assumptions.** Thus, the diagonals of quadrilateral $APCX$ bisect in their midpoint and quadrilateral $APCX$ is a **strict** parallelogram. Now according to the fact that the opposite sides of a parallelogram are parallel and have the same length, we have AX and CP (or BP) are parallel, $AX \equiv CP$. As we know that P is the midpoint of BC we also have $AX \equiv PB$. **The quadrilaterals $AXBP$ and $AXPB$ has then two opposite sides which are parallel and of the same length and have three of their vertices non-collinear. One of them is a strict parallelogram according to a well known property of parallelogram. Now assuming that $AXBP$ is a parallelogram then we can deduce that point Q, midpoint of PX is also the midpoint of AB. Thus, points B and C are equal but this contradicts the fact that ABC is non-flat. Therefore this is $AXPB$ which is a parallelogram.** Finally the opposite sides of a strict parallelogram are strictly parallel, thus AB and PQ are **strictly** parallel.

Fig. 2. A comparison between the formal and informal proofs of the midpoint theorem.

```
Lemma triangle_mid_par_strict_cong_simp :
 forall A B C P Q,
 ~ Col A B C ->
 is_midpoint P B C ->
 is_midpoint Q A C ->
 Par_strict A B Q P.
Proof.
intros.
assert (exists R : Tpoint, is_midpoint Q P R).
apply symmetric_point_construction.
decompose [ex] H2.
assert (~ Col A B C /\ A <> B /\ B <> C /\ A <> C).
apply not_col_distincts; assumption.
clear H; use H4.
assert (~ Col A P C).
intro.
assert (Col A B C).
apply col_permutation_1.
apply col_transitivity_1 with P.
show_distinct C P.
assert (C = B).
apply is_midpoint_id_2; assumption.
apply eq_sym in H4; contradiction.
assumption.
Col.
apply col_permutation_2; apply midpoint_col; assumption.
contradiction.
assert (Parallelogram_strict A P C x).
apply mid_plgs with Q.
assumption.
split; assumption.
assert (Cong A P C x /\ Cong A x P C).
apply plgs_cong; assumption.
use H9.
assert (~ Col A P C /\ A <> P /\ P <> C /\ A <> C).
apply not_col_distincts; assumption.
clear H4.
use H9.
assert (Par A P C x /\ Par A x P C).
apply plg_par.
assumption.
assumption.
apply Parallelogram_strict_Parallelogram; assumption.
use H9.
assert (Cong A x B P).
assert (Cong B P P C).
apply mid_cong; assumption.
apply cong_transitivity with P C; Cong.
assert (Par A x B P).
apply par_col2_par with P C.
assert (P <> B /\ P <> C).
apply mid_diff; assumption.
use H17.
apply not_eq_sym; assumption.
assumption.
apply midpoint_col; Mid.
Col.
assert (Plg A x B P \/ Plg A x P B).
apply par_cong_plg; assumption.
assert (~ Plg A x B P).
intro.
apply plg_to_parallelogram in H19.
assert (exists M : Tpoint, is_midpoint M A B /\ is_midpoint M x P).
apply plg_mid; assumption.
decompose [ex] H20; clear H20.
use H21.
assert (Q = x0).
apply l7_17 with P x; Mid.
assert (Col A B C).
apply col_transitivity_1 with Q.
show_distinct A Q.
assert (A = C).
apply is_midpoint_id.
treat_equalities; assumption.
contradiction.
assumption.
treat_equalities; apply col_permutation_4;
apply midpoint_col; assumption.
apply col_permutation_4;
apply midpoint_col; assumption.
contradiction.
decompose [or] H18.
contradiction.
clear H18; clear H19.
apply plg_to_parallelogram in H20.
assert (Par A x P B /\ Par A B x P).
apply plg_par.
show_distinct A x.
assert (Col A B B) by Col.
contradiction.
assumption.
show_distinct P x.
assert (A = B).
apply l7_9 with Q C; assumption.
contradiction.
apply not_eq_sym; assumption.
assumption.
use H18.
assert (Par P Q A B).
apply par_col_par with x.
show_distinct P Q.
assert (Col A P C).
apply col_permutation_4;
apply midpoint_col; assumption.
contradiction.
assumption.
apply col_permutation_1;
apply midpoint_col; assumption.
Par.
apply par_not_col_strict with x.
Par.
apply midpoint_col; assumption.
intro.
assert (Col A B P).
apply not_strict_par1 with x x.
Par.
Col.
Col.
assert (Col A B C).
apply col_permutation_2.
apply col_transitivity_1 with P.
show_distinct B P.
assert (Col A B B) by Col.
contradiction.
assumption.
apply col_permutation_4.
apply midpoint_col; assumption.
Col.
contradiction.
Qed.
```

Fig. 3. Early version of the formal proof of the midpoint theorem using less automated tactics and notations.

```
Lemma three_medians_intersect:
 forall A B C I J K,
 ~Col A B C ->
 is_midpoint I B C ->
 is_midpoint J A C ->
 is_midpoint K A B ->
 exists G,
 Col G A I /\ Col G B J /\Col G C K.
Proof with assert_all.
intros.
assert_diffs.
Name G the intersection
   of the medians (A I)
   which is a median since H0 and (B J)
   which is a median since H1
   of the non-flat triangle A B C H.
exists G; repeat split; try assumption.
Name D the symmetric of C wrt G.
assert_all.
show_distinct' A D.
permutation_intro_in_hyps.
assert (Par1 :=
   triangle_mid_par A D C G J H13 H14 H1).
show_distinct' B G.
assert (Par G B A D)
  by (perm_apply (par_col_par A D G J B)).
show_distinct' B D.
assert (Par2 :=
  triangle_mid_par B D C G I H103 H14 H0).
show_distinct' A G.
assert (Par G A D B)
  by (perm_apply (par_col_par B D G I A))...
show_distinct' D G.
assert (~ Col G A D)
  by (intro; search_contradiction).
assert (Parallelogram G A D B)
  by (apply (par_2_plg G A D B); finish).
Name Z the intersection
  of the diagonals (G D)
  and (A B) of the parallelogram H17...
ColR.
Qed.
```

Given a triangle ABC, let I, J and K be the midpoints of BC, AC and AB respectively. There exists a point G which belongs to the three medians of the triangle AI, BJ and CK (Fig.5).

Let G be the intersection of AI and BJ. We need to show that G belongs to CK.

Construct D the symmetric of C wrt G.

Using the midpoints theorem we have that GB is parallel to AD and GA is parallel to DB.

Hence $GADB$ is a parallelogram and then its diagonals intersect and their midpoint K and so G, C and K are collinear.

Fig. 4. A comparison of the formal and informal proofs that the medians of a triangle intersect in a single point.

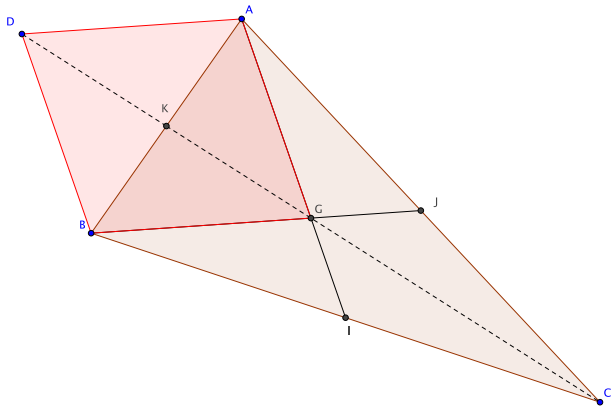

Fig. 5. The medians intersect in a single point.

case of unification modulo a theory. In the future, we plan to implement more general unification algorithm in order to deal with geometries involving function symbols. Finally, our work about NDCs could be enriched by using a well known key feature of Wu's method: its ability to provide automatically NDCs.

Availability

The full Coq development can be found at the following url: `http://dpt-info.u-strasbg.fr/~narboux/tarski.html`

References

1. Stuart F. Allen, Robert L. Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. The nuprl open logical environment. In David A. McAllester, editor, *CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 170–176. Springer, 2000.
2. Jürgen Avenhaus. Efficient Algorithms for Computing Modulo Permutation Theories. In David Basin and Michal Rusinowitch, editors, *Automated Reasoning*, volume 3097 of *Lecture Notes in Computer Science*, page 415429. Springer Berlin Heidelberg, 2004.
3. Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.
4. Michael Beeson. A constructive version of tarski's geometry, 2014.

5. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
6. Samuel Boutin. Using reflection to build efficient and certified decision procedures. In *TACS'97. Springer-Verlag LNCS 1281*, pages 515–529. Springer-Verlag, 1997.
7. Thomas Braibant and Damien Pous. Tactics for reasoning modulo ac in coq. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2011.
8. Gabriel Braun and Julien Narboux. From Tarski to Hilbert. In Tetsuo Ida and Jacques Fleuriot, editors, *Automated Deduction in Geometry 2012*, Edinburgh, United Kingdom, September 2012.
9. Bruno Buchberger and Franz Winkler, editors. *Gröbner Bases and Applications*. Cambridge University Press, 1998.
10. Shang-Ching Chou. *Mechanical Geometry Theorem Proving*. D. Reidel Publishing Company, 1988.
11. Shang-Ching Chou and Xiao-Shan Gao. A class of geometry statements of constructive type and geometry theorem proving. In *Proceeding of CADE 92*, 1992.
12. Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. *Machine Proofs in Geometry*. World Scientific, Singapore, 1994.
13. Pierre Corbineau. A declarative language for the coq proof assistant. In Marino Miculan, Ivan Scagnetto, and Furio Honsell, editors, *Types for Proofs and Programs*, volume 4941 of *Lecture Notes in Computer Science*, pages 69–84. Springer Berlin Heidelberg, 2008.
14. Yann Coscoy, Gilles Kahn, and Laurent Théry. Extracting text from proofs. In Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin, editors, *TLCA*, volume 902 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 1995.
15. Christophe Dehlinger, Jean-François Dufourd, and Pascal Schreck. Higher-order intuitionistic formalization and proofs in Hilbert's elementary geometry. In *Proceedings of Automated Deduction in Geometry (ADG00)*, Lecture Notes in Computer Science, pages 306–324, 2000.
16. Jean-David Génevaux, Julien Narboux, and Pascal Schreck. Formalization of wu's simple method in coq. In *Certified Programs and Proofs (CPP 2011)*, volume 7086 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2011.
17. Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A Small Scale Reflection Extension for the Coq system. Rapport de recherche RR-6455, INRIA, 2008.
18. Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of high-level formal proofs. In Jim Hendler and Devika Subramanian, editors, *AAAI/IAAI*, pages 277–284. AAAI Press / The MIT Press, 1999.
19. Predrag Janičić, Julien Narboux, and Pedro Quaresma. The area method: a recapitulation. *Journal of Automated Reasoning*, 48(4):489–532, 2012.
20. Claude Kirchner and Hélène Kirchner. Rewriting, solving, proving. A preliminary version of a book available at `http://www.loria.fr/~ckirchne/rsp.ps.gz`, 1999.
21. Stéphane Lescuyer. First-class containers in coq. *Stud. Inform. Univ.*, 9(1):87–127, 2011.
22. Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing desargues' theorem in coq using ranks. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 1110–1115. ACM, 2009.
23. Julien Narboux. A decision procedure for geometry in Coq. In Slind Konrad, Bunker Annett, and Gopalakrishnan Ganesh, editors, *Proceedings of TPHOLs'2004*, volume 3223 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

24. Julien Narboux. *Formalisation et automatisation du raisonnement géométrique en Coq*. PhD thesis, Université Paris Sud, September 2006.
25. Julien Narboux. A graphical user interface for formal proofs in geometry. *Journal of Automated Reasoning*, 39(2):161–180, 2007.
26. Julien Narboux. Mechanical theorem proving in Tarski's geometry. In *Proceedings of Automatic Deduction in Geometry 06*, volume 4869 of *Lecture Notes in Artificial Intelligence*, pages 139–156. Springer-Verlag, 2007.
27. T. Nipkow. Equational reasoning in isabelle. *Sci. Comput. Program.*, 12(2):123–149, July 1989.
28. Tuan Minh Pham and Yves Bertot. A combination of a dynamic geometry software with a proof assistant for interactive formal proofs. In *9th International Workshop On User Interfaces for Theorem Provers FLOC'10 Satellite Workshop*, Electronic Notes in Theoretical Computer Science (ENTCS), Edinburgh, Scotland, Royaume-Uni, 2010. Elsevier.
29. Wolfram Schwabhauser, Wanda Szmielew, and Alfred Tarski. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, Berlin, 1983.
30. Sana Stojanović, Vesna Pavlović, and Predrag Janičić. A coherent logic based geometry theorem prover capable of producing formal and readable proofs. In Pascal Schreck, Julien Narboux, and Jürgen Richter-Gebert, editors, *Automated Deduction in Geometry*, volume 6877 of *Lecture Notes in Computer Science*. Springer, 2011.
31. Alfred Tarski. What is elementary geometry? In P. Suppes L. Henkin and A. Tarski, editors, *The axiomatic Method, with special reference to Geometry and Physics*, pages 16–29, Amsterdam, 1959. North-Holland.
32. Dongming Wang. *Elimination Methods*. Springer, 2001.
33. Markus Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Institut f r Informatik, Technische Universit t M nchen, 2002.
34. Wen-Tsün Wu. On the decision problem and the mechanization of theorem proving in elementary geometry. *Scientia Sinica*, 21:157–179, 1978.