Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 285 (2012) 43–55

www.elsevier.com/locate/entcs

A Combination of a Dynamic Geometry Software With a Proof Assistant for Interactive Formal Proofs

Tuan Minh Pham[1] Yves Bertot[2]

Inria Sophia Antipolis

Abstract

This paper presents an interface for geometry proving. It is a combination of a dynamic geometry software, Geogebra[11] with a proof assistant, Coq[8]. Thanks to the features of Geogebra, users can create and manipulate geometric constructions, they discover conjectures and interactively build formal proofs with the support of Coq. Our system allows users to construct fully traditional proofs in the same style as the ones in high school. For each step of proving, we provide a set of applicable rules verified in Coq for users, we also provide tactics in Coq by which minor steps of reasoning are solved automatically.

Keywords: interactive geometric proofs, Coq, proof assistant, dynamic geometry

1 Introduction

Nowadays, dynamic geometry software (DGS) plays an important role and has highly influenced mathematics education. Students can use it to construct geometric objects, observe how these objects change when moving free points or applying euclidian transformations, and they can discover conjectures. There are numerous applications for interactive geometry on the market with a variety of features, many of them are really used in classroom in many countries. However, these uses are usually limited at the level of discovering conjectures. Some DGS provide proof feature by combining with automated geometry theorem proving (GTP) and allow users to verify conjectures. They rely on several efficient automatic proof methods, such as Gröbner bases method[13], Wu's method[4], the area method[5] and the full-angles method[6]. The first two methods are algebraic methods which use polynomials to solve problems, they do not give us human-readable proofs. The last two ones can

[1] Email: tuan-minh.pham@sophia.inria.fr

[2] Email: Yves.Bertot@inria.fr

1571-0661
doi:10.1016/j.entcs.2012.06.005

produce human-readable proofs, nevertheless these proofs are not fully traditional proofs, they do not rely on the geometric knowledge as taught in high school.

We aim at constructing a tool for students to prove geometric theorems interactively. Students can reason step-by-step in the style taught in high school. This style may involve backward and forward proof reasoning, it may also involve the construction of auxiliary lines. We present GeoCoq, an interface for interactive geometry proving which is a combination of Geogebra with Coq. Reasoning steps are built by mouse clicks and are sent to Coq. For its part, Coq executes the reasoning steps, the response is sent back to the user to continue their proof. The logical steps and the knowledge are given by a library that was developed with the help of a high-school teacher.

With the support of a proof assistant like Coq, reasoning steps are verified and the correctness of proofs is guaranteed. This is an advantage in using Coq because geometry reasoning usually uses tacit assumptions based on visual evidence without verification.

We also developed automatic tactics that help users solve minor steps of reasoning. Users do not have to delve into details that lead to technical proofs and are not adapted to their level of abstraction. In addition, with the formalization of automatic proof methods as tactics in Coq (such as the area method which is formalized by J.Narboux[14]), users can use these methods to check the fact before proving it, or alternatively use them in steps of constructing proofs.

This paper is organized as follows. In section 2, we give a short description about Geogebra. The Coq system, the communication using Pcoq, and a brief description about the library for geometry are described in section 3. Section 4 presents the interface and provided features for proofs. The next section deals with our implementations. We mention some related work in section 6 and the last section is for our conclusions and perspectives.

2 Geogebra

Geogebra is a free dynamic geometry software for education in schools. It was started by Markus Hohenwarter in 2001 in his master and PhD work[11]. It is implemented in Java and thus available for multiple platforms. It provides a new kind of tool for mathematics by joining geometry, algebra and calculus. It received several international educational software awards and is applied in education at schools in different countries. Like other dynamic geometry systems, Geogebra provides basic geometric objects such as points, vectors, straight lines, circle, and more complex constructions such as midpoint, parallel lines, circumcircle... Users can construct geometric objects and manipulate them. Geogebra also allows students to undo or redo constructions at anytime. By moving free points in figures students can find out conjectures and check theirs correctness, checks are limited to simple relations of 2 objects at the moment.

A strong point of Geogebra which makes it different from other DGSs is the connection of Dynamic Geometry and Computer Algebra System (CAS). Geogebra

can profit both from visualization capabilities of CAS and dynamic changeability of DGS. It encourages students to discover mathematics in a bidirectional experimental way. Students can investigate equations corresponding to drawn objects and they can also investigate the figures corresponding to given equations.

By combining Geogebra and Coq, we would like to provide an interface which allows students to have many points of view on a geometric problem.

3 Coq communication and Geometry library

Coq is a proof development system, which allows users to define functions or predicates, state mathematical theorems and interactively develop formal proofs. Coq is not an automated theorem prover, however it provides a tactic language letting users define their own proof method and it also includes tactics for automatic theorem proving and decision procedures. So Coq is adapted to our needs in interactively constructing proofs.

Some integrated development environments (IDE) can be used to communicate with Coq. Here, we are interested in Pcoq which is a graphical user-interface for Coq[1]. Using Pcoq to communicate with Coq offers some benefits. Pcoq is implemented in Java, the same programming language as Geogebra, so this makes it easy to be integrated in Geogebra. Pcoq manipulates all formulas and commands as tree-like structures (also known as abstract syntax trees) rather than plain text. This makes it easy to attach special mathematical notations to some functions, and to attach sentences in natural language to geometrical statements. Furthermore, this provides an easy access to the structure of information (hypotheses, goals) that we receive from Coq.

In order to be able to reason in Coq, we still need a geometry library, in which geometry basic notions and their properties are formalized, geometry rules that we want to provide for users are verified. An interesting formalization was developed by F.Guilhot for the French high school curriculum[12]. It is based on providing a new axiom system that is more adapted to the knowledge of students. It covers a large portion of basic notions of plane geometry, properties and theorems. Moreover, proofs are traditional, they are formalized in the manner of reasoning suitable to the capacity of high school students. Among the classic theorems which are proved in this library, we can cite Menelaus, Ceva, Desargues, Pythagoras, Simson's line...

4 The interface

Our interface is a combination of Geogebra with Pcoq. We integrate Pcoq as a view of Geogebra (figure 1). Like other views, it can be shown, hidden, dragged around to modify its position, and opened as an external window.

Two main sub windows of Pcoq are the command window and proof window. The command window is used to display all Coq commands which state and prove theorems. The proof window allows users to construct proofs, they can see hypotheses, and goals need to be proved for each step of geometry reasoning in this window.

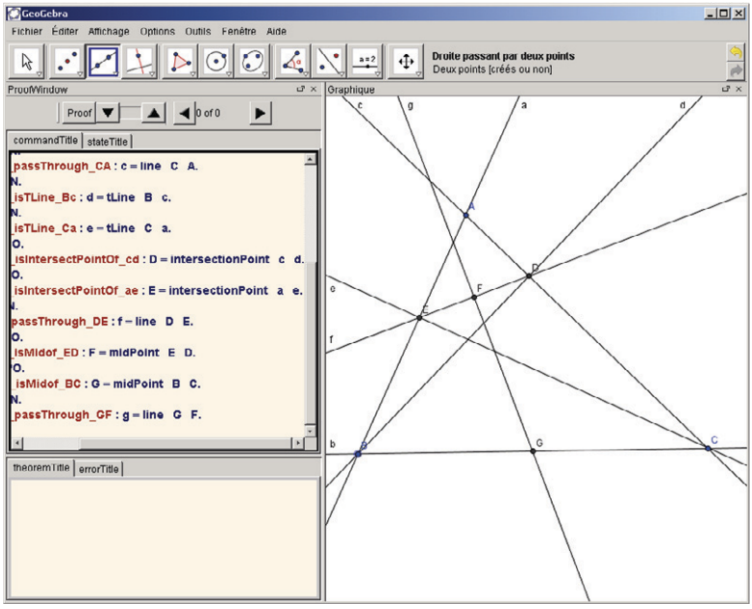

Fig. 1. A screen-shot of GeoCoq GUI. Pcoq is integrated as the windows in the left

In addition, other sub windows are used to display applicable rules for a goal, to apply rules by giving values for their parameters, and to announce errors... We detail the interface and its features by showing how users can state and prove theorems. The following theorem will be used as an example for the rest of this section:

Example 4.1 Let BD and CE be two altitudes of triangle ABC and points G and F be the midpoints of BC and DE respectively. It holds that GF $\perp$ DE.

4.1 Stating theorems

4.1.1 Constructing diagrams

Geogebra provides common geometry construction tools for users to construct diagrams. Geometry objects are created one by one by using appropriate construction tools and selecting existing objects. Users can move free points, and dependent constructions will be updated, this can help users discover conjectures.

To construct a diagram corresponding to the above example, users draw a triangle $\triangle ABC$, construct perpendicular lines d $\perp$ AC such that B $\in$ d and e $\perp$ AB such that C $\in$ e, take intersection points E and D, and complete the figure by taking midpoints G and F of BC and DE respectively and joining them.

Each construction is translated into commands in Coq (figure 1). Adding new geometry objects or removing existing geometry objects implies adding or removing corresponding hypotheses. In most geometry theorem provers, a predicate form is used to describe geometric statement, i.e. hypotheses and conclusion are represented by geometry predicates. Our tool is not an exception, however in the phase of stating theorem, we have geometric statement in constructive form (left column of table 1). In theorem proving phase which will be presented in the next section, users can

obtain the corresponding geometric predicates (right column of table 1) using our tactics in Coq to unroll definitions in hypotheses.

Constructive form	Predicate form
M = midPoint (A, B)	collinear (A B M) $\wedge$ MA = MB
M = intersectPoint (l_1, l_2)	$l_1 \nparallel l_2 \wedge$ M $\in l_1 \wedge$ M $\in l_2$
l = line (A, B)	A $\neq$ B $\wedge$ A $\in$ l $\wedge$ B $\in$ l
l = parallelLine (A, m)	A $\in$ l $\wedge$ l $\parallel$ m
l = perpendicularLine (A, m)	A $\in$ l $\wedge$ l $\perp$ m

Table 1
Constructive and predicate form of some constructions

4.1.2 Discovering conjectures

Once a diagram is completely constructed, users easily see free points that do not depend on other constructions as these points are highlighted using a different color. Users can move these free points by drag-and-drop, dependent constructions are updated, and users can observe relationships between geometry objects, traces of points... If they see that a conjecture seems to be true, they can decide to prove it. We can pre-verify the correctness of conjecture to ensure that users are going to prove a fact. This pre-verification can be performed using Geogebra feature with the support of its CAS, or using an automatic proof method in Coq (such as the area method). For our example, users can move points A, B and C, they find out perpendicularity of GF and DE (GF $\perp$ DE). They select these lines and right click, a dialog appears to confirm this perpendicularity, and ask users to prove it (figure 2). Once users decide to prove the conjecture, we can go to theorem proving phase in the next section.

4.2 Theorem proving

We start this section by discovering geometric predicates from the construction. Our implementation allows users to get geometric predicates automatically or manually. Users can select a geometric object definition in hypotheses and get its properties using mouse-clicks, they can also unroll all definition in hypotheses at the same time. The corresponding tactics are automatically sent to Coq. The following tactics collects the hypotheses that define new points (like M is the intersection point of lines a and b) and creates new hypotheses that enumerate the given properties of the new points (like M $\in$ a, M $\in$ b and a $\nparallel$ b).

```
Ltac unroll_AllDefinition := match goal with
|H:?M = intersectionPoint ?a ?b |- _ =>
let H1 H2 H3 :=fresh "intersectionPointProperty" in
    destruct (@unroll_intersectionPoint M a b)
```

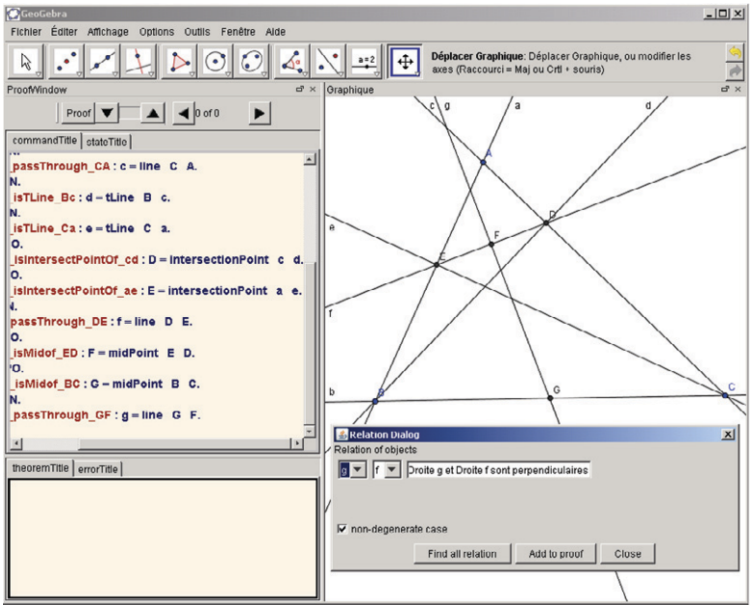

Fig. 2. Users find and add a conclusion

```
      as [H1 [H2 H3] ]; auto;
    revert H; clear H; unroll_AllDefinition; intro H
end.
```

Once this tactic is applied, geometric object definitions are replaced by the corresponding predicates in the second column of table1.

Some predicates such as $l_1 \nparallel l_2$, A $\neq$ B... may be considered as non degeneracy conditions for the existence of objects. With many definitions of geometric objects, we implicitly add non degeneracy conditions in hypotheses. These conditions are very important for reasoning, sometimes resolving a problem in degenerate cases is more complex than resolving the original problem. The same diagram with different way of construction will have different non degeneracy conditions.

4.2.1 Searching and Applying rules

Here, we present a backward-chaining approach for geometry reasoning. This approach progresses from the conclusion to the hypotheses, i.e. we need to prove $\forall GeometricElements, Hyp_1 \wedge \ldots \wedge Hyp_n \rightarrow Goal$. We search in a rule set to find a rule of the following form $\forall GeometricElements, G_1 \wedge \ldots \wedge G_n \rightarrow Goal$. The problem evolves into proving the subgoals $G_1, \ldots, G_n$. This process is repeated for each subgoal until the subgoal is in the hypotheses or is an axiom.

Searching applicable rules in our interface is implemented using Search commands. It is strong enough to find out all rules in a database which lead to a goal given by a pattern. From the proof window of the interface, users can search all applicable rules for the current goal. A list of applicable rules will be displayed, allowing users to select an appropriate rule. After giving values for parameters of this rule which usually are points, lines..., users can update statement of the rule

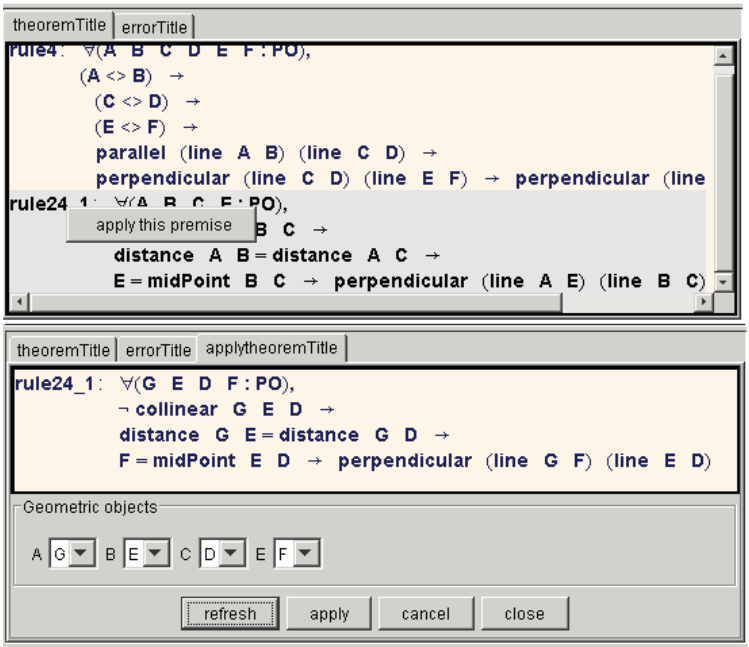

Fig. 3. Applicable rules are displayed at the bottom, users select values to apply the rule

with given values, the system checks correctness by verifying its hypotheses.

Returning to our example 4.1, to prove that GF ⊥ DE, we need to find rules which have conclusion in the form of perpendicularity of 2 lines. A variant of Search command is sent to Coq *SearchPattern (_ _ ⊥ _ _) inside ModuleName*

With a list of applicable rules (the upper figure of figure 3), we can select the following rule to apply.

Lemma isosceles_perp: ∀ A B C M :Point, B ≠ C → A ≠ M → AB = AC → M = MidPoint(B,C) → AM ⊥ BC.

Using the four points G, D, E and F as values for its parameter respectively, we have an instance of this rule (the lower figure of figure 3)

D ≠ E → G ≠ F → GD = GE → F = MidPoint(D,E) → GF ⊥ DE.

By construction of F, we have F = MidPoint(D,E), so applying this rule is reasonable, it is performed by command *apply isosceles_perp with (A:=G)(B:=D)(C:=E)(M:=F)*. Now, we have to prove that GD = GE instead of proving that GF ⊥ DE.

4.2.2 Adding new properties

Generally, we can not use only backward reasoning to solve problems, an alternative way is to use forward-chaining. This process aims at generating new properties from the hypotheses by applying given rules. These properties are used as hypotheses in the next steps of the process. The process finishes when it can generate the goal. If we need to prove that $\forall GeometricElements, Hyp_1 \wedge \ldots \wedge Hyp_n \rightarrow Goal$. And in the base of rules we have $\forall GeometricElements, H_{j_1} \wedge \ldots \wedge H_{j_m} \rightarrow Goal_j$, with $\{H_{j_1}, \ldots, H_{j_m}\}$ are a subset of $\{Hyp_1, \ldots, Hyp_n\}$, so we can add G_j as a hypothesis

of theorem.

Our system provides a variant of forward-chaining, allowing users to add new properties while proving theorems. A property is used as an hypothesis if it is proved. In our example, to prove that GD = GE, users try to prove that GD = GC and GE = GC. For the first, users select 2 segments GD and GC in the figure, with a right click users can find out their equal relation. The system asks users to add this equation to the current proof. To prove GD = GC, we will use the following rule:

Lemma rightTriangle_midPoint: $\forall$ A B C M :Point, A $\neq$ B $\rightarrow$ A $\neq$ C $\rightarrow$ line(A,B) $\perp$ line(A,C) $\rightarrow$ M = MidPoint(B,C) $\rightarrow$ MA = MB $\wedge$ MA = MC.

With the configuration of the 4 points D, B, C and G from our example, we have:

A $\neq$ B $\rightarrow$ A $\neq$ C $\rightarrow$ line(A,B) $\perp$ line(A,C) $\rightarrow$ M = MidPoint(B,C) $\rightarrow$ MA = MB $\wedge$ MA = MC.

It is the same way for the second one GE = GC. It remains to prove that line(B,D) $\perp$ line(C,D) and line(C,E) $\perp$ line(B,E). They are proved thanks to the construction of points D, E and using automatic tactics that we will present later.

4.2.3 Drawing auxiliary lines

Sometime, users need to add lines, ray, segment... to a diagram during their proofs. It is quite important to ensure the existence of these objects. In the theorem proving phase, each created object exists only under some conditions. So when users create a new object, they need to verify its conditions before using this object. The following commands are sent to Coq when users create the intersection point M of 2 lines l_1 and l_2.

```
let H:= fresh "isIntersectionPoint" in
    assert(H: exists M, M = IntersectionPoint l1 l2;
    apply exists\_intersectionPoint.
```

Coq requests to prove $l_1 \nparallel l_2$ to guarantee existence of M. Once this condition is verified, the point M will be displayed in drawing window, and the command *destruct H as [M, H]* automatically is sent to Coq to have M = IntersectionPoint(l_1,l_2)) in hypotheses.

4.2.4 Automatic tactics

Proving geometry with the support of Coq gives a high level of confidence, each reasoning step is verified. However, it also complicates the process of proving. Users not only decide which rule will be applied but also prove many minor goals which lead to tedious proofs and is not suitable for a pedagogical setting. So, to avoid overwhelming users in proof details, we try to provide tactics to automatically solve these problems.

For example, we present a tactic to solve problems of line and points on the line. In our example, we have to prove that line(B,D) $\perp$ line(C,D) while we have c = line(A,C), d = perpendicularLine(B,c), D = IntersectionPoint(c,d). This proof in

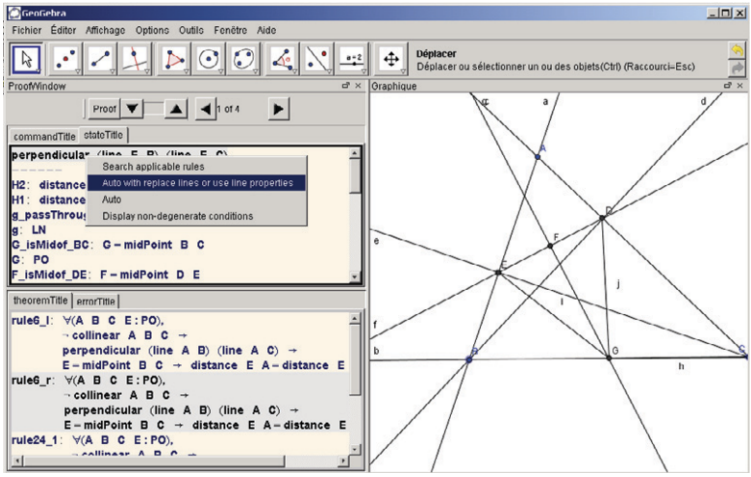

Fig. 4. Users can use automatic tactics in popup menu

Coq is detailed. From c = line(A,C) we have C ∈ c. From D = IntersectionPoint(c,d) we have D ∈ c. So we have line(C,D) = c. We also have line(B,D) = d. From the hypothesis d = perpendicularLine(B,c) we have d ⊥ c, and we get line(B,D) ⊥ line(C,D).

This complexity comes from the distinction of lines in Coq. For example, if we have that a line l pass through points M N P, we easily have line(M,N) = line (N,P) = line (M,P) = l. But these lines can not automatically be replaced each other by the system. We provide the following tactics to solve this problem by discovering all relation of points and lines and replacing all instance of line(M,N) by line l if it finds M ∈ l and N ∈ l in hypotheses.

```
Ltac replace_auto_Lines :=
unroll_AllDefinition;
match goal with
|- context [line ?M ?N] =>
    match goal with | H1:liesOnLine M ?l |- _ =>
      match goal with | H2:liesOnLine M l |- _ =>
      try (replace (line(M,N)) with l in *;auto);
      replaceAllLineInstances
      end
    end
end.
```

For each particular goal, corresponding tactics are provided in a popup menu (figure 4).

5 Implementation

Geogebra is implemented in Java, its architecture is clear and organized in separate layers and modules. In the last version (v3.3.69), each view of Geogebra (such as

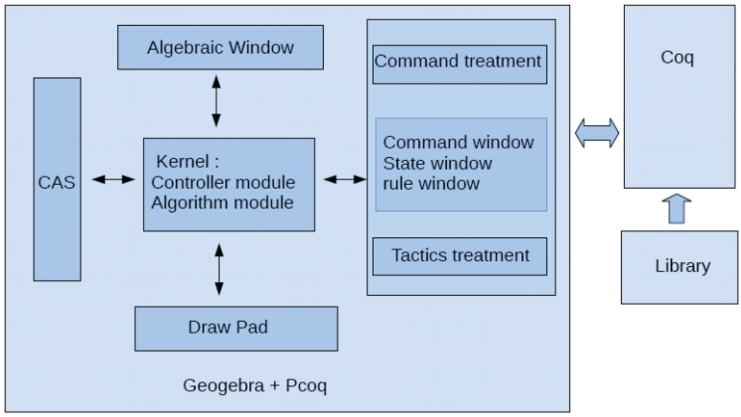

Fig. 5. The system architecture.

drawing pad, algebra view, CAS view, Spreadsheet view) is independently implemented by extending a JPanel class of Java and implementing a View interface. Hence, some features are provided that allow users to show or hide these views, drag around to modify their position, and open them in an external window as well.

```
public interface View{
    public void add (GeoElement geo);
    public void remove (GeoElement geo);
    public void update (GeoElement geo);
    ...
}
```

These views work in synchronized mode thanks to a controller module. This notifies every change of geometric objects to all system view by calling the corresponding functions in each view interface.

Our integration of Pcoq in Geogebra respects this architecture (figure 5. Pcoq is implemented as a view of Geogebra. Once it receives a notification, the corresponding commands are produced and sent to Coq. As we said, communicated information are in tree format, so a module to produce tree format commands from a construction is necessary. The following code lines is an example for adding a midpoint in theorem stating phase:

```
private Tree[] commandMidPoint(GeoElement geo){
//this function is to produce commands for a midpoint
Tree[] contents=null;
AlgoElement parentAlgo = geo.getParentAlgorithm();
GeoElement point1 = parentAlgo.getInput()[0];
GeoElement point2 = parentAlgo.getInput()[1];
contents = new Tree[2];
contents[0] = TreeFormat.addVar(geo.getLabel(),
                                TreeFormat.Point);
contents[1] = TreeFormat.addHypothesis("Hyp_MidPoint",
```

```
    TreeFormat.assignVariable(geo.getLabel(),
        TreeFormat.function(TreeFormat.MidPointAB,
            point1.getLabel(), point2.getLabel())));
return contents;}
```

In our implementation, we continue the notion of "proof by pointing"[3] which has been already approached in Pcoq. It allows users to guide precisely the proof process using the mouse in the user-interface of proof assistants i.e., the gestures of pointing at a subexpression of goals or assumptions is enough to synthesize appropriate commands for the system.

While proof-by-pointing in Pcoq relies on analyzing formulas and understanding the meaning of logical symbols, this notion in our system is realized by the geometric signification of formulas. For each reasoning step, by finding out the signification of subgoal, hypotheses that users selected, the system can provide appropriate tactics in a popup menu to guide users to solve the problem. For example, we determine if the selected goal has the format of (_ = line _ _), (_ = perpendicularline _ _) or (_ = parallelLine _ _), in this case the system provides replace_auto_Lines tactics to solve it.

6 Related Work

DGSs are more and more used in education. Many works have been performed to provide combinations of DGS and GTP. Some of them use automatic methods, allow users to prove geometry theorems. We will cite here several systems that can generate readable proofs.

Geothms[16] with a web interface uses the GCLC prover which is based on the area method. With the support of repository of geometry theorems, users can store theorem statements and their proofs.

MMP/Geometer[10] and GeoExpert[9] are strong systems which implement Wu's method, the area method, the full-angles method (for GeoExpert), and especially the deductive database method[7]. The last method relies on a set of basic rule, and allows to find out traditional proofs. GeoExpert allows users to visually understand each step in generated proof.

Geometry Explorer[17] also implements the full-angles method. However, it can automatically generate novel diagrammatic proofs corresponding with reasoning used by its GTP. It is a good illustration for proofs.

Geoproof[15] is very close to our system. Reasoning in Geoproof is performed by Coq and based on formalization of the area method in this proof assistant[14]. Since Geoproof also uses Coq, so we will have some similar points in stating phase. The main differences lie on theorem proving phase and the way the system connects to Coq. Geoproof only allows users to construct theorem statements, send them to a CoqIDE and check this fact by apply the area method. It does not allow users to construct their own proofs step-by-step and does not provide interactions between DGS and Coq.

Another one which combines a DGS with Coq is GeoView[2], but this tool works in the opposite direction. It produces a diagram in a DGS namely GeoPlan from a theorem statement in Coq.

7 Conclusion and Future Work

In this paper, we present our interface which allows users to interactively construct traditional proofs in Geometry. Reasoning methods at high school level such as forward method, backward method, and drawing auxiliary lines as well are provided. Steps of reasoning are verified by a proof assistant hence the correctness is guaranteed.

Integrating a proof assistant in a dynamic geometry software offers a novel way in learning geometry. It allows users to additionally have a logic view in solving geometric problems. Users know what they have in hypotheses, what they have to prove, which rules they can apply... By which, they can take good decisions and thoroughly understand reasoning steps.

Some features need to be improved to allow users to easily use our tool. Dynamically constructing diagram from statement of rules is an example. With constructed diagram of a rule, users easily find points, lines that their configuration conforms with this diagram. Then users can drag and drop these objects to apply this rule. The second one is how we can organize and manage the set of applicable rules and the set of tactics while adapting to user levels.

References

[1] Amerkad, A. and Y. Bertot and L. Pottier and L. Rideau: 2001, *Mathematics and Proof Presentation in Pcoq*. Workshop Proof Transformation and Presentation and Proof Complexities in connection with IJCAR

[2] Bertot, Y. and F. Guilhot and L. Pottier: 2004, *Visualizing Geometrical Statements with GeoView*. Electronic Notes in Theoretical Computer Science 103:49-65

[3] Bertot, Y. and G. Khan and L. Théry: 1994, *Proof by pointing*. Theoretical Aspects of Computer Software, LNCS 789:141-160

[4] Chou, S.-C.: 1988, *An introduction to Wu's method for mechanical theorem proving in geometry*. Journal of Automated Reasoning 4:237-267

[5] Chou, S.-C. and X.-S. Gao and J.-Z.Zhang: 1994, *Machine proofs in geometry: Automated production of readable proofs for Geometry Theorems* . World Scientific

[6] Chou, S.-C. and X.-S. Gao and J.-Z.Zhang: 1996, *Automated generation of readable proofs with geometric invariants. Theorem proving with full-angles*. Journal of Automated Reasoning 17:349-370

[7] Chou, S.-C. and X.-S. Gao and J.-Z.Zhang: 1996, *A deductive database approach to automated geometry theorem proving and discovering*. Journal of Automated Reasoning 25:219-246

[8] Coq development team, *The Coq proof assitant reference manual*. http://coq.inria.fr/refman/

[9] Gao, X.-S.: 2000, *Geometry Expert, Software Package*. http://www.mmrc.iss.ac.cn/~xgao/gex.html

[10] Gao, X.-S. and Q.Lin: 2002, *MMP/Geometer - a software package for automated geometry reasoning*. Proceedings of Automated Deduction Geometry 44-46

[11] Geogebra development team, *Introduction to GeoGebra*. http://www.geogebra.org/book/intro-en/

[12] Guilhot, F.: 2005, *Formalisation en Coq et visualisation d'un cours de géométrie pour le lycée*. Technique et Science Informatiques 24:1113-1138

[13] Kapur, D.: 1986, *Using Gröbner Bases to reason about geometry problems.* Journal of Symbolic Computation 2:399-408

[14] Narboux, J.: 2004, *A Decision Procedure for Geometry in Coq.* Proceedings of Theorem Proving in Higher-Order Logics,LNCS 3223

[15] Narboux, J.: 2007, *A Graphical User Interface for Formal Proofs in Geometry.* Journal of Automated Reasoning 39:161-180

[16] Quaresma, P. and P. Janičić: 2007, *GeoThms — a Web System for Euclidean Constructive Geometry.* Electronic Notes in Theoretical Computer Science 174:35-48

[17] Wilson, S. and Jacques D. Fleuriot: 2005, *Combining Dynamic Geometry, Automated Geometry Theorem Proving And Diagrammatic Proofs.* Proceedings of the European Joint Conferences on Theory and Practice of Software (ETAPS) Satellite Workshop on User Interfaces for Theorem Provers (UITP)