A Matroid-based Automatic Prover and Coq Proof Generator for Projective Incidence Geometry

David Braun, Nicolas Magaud, Pascal Schreck

Noname manuscript No.
(will be inserted by the editor)

A Matroid-based Automatic Prover and Coq Proof Generator for Projective Incidence Geometry

David Braun · Nicolas Magaud · Pascal Schreck

Received: date / Accepted: date

Abstract We present an automatic theorem prover for projective incidence geometry. This prover does not consider coordinates. Instead, it follows a combinatorial approach based on the concept of rank. This allows to deal only with sets of points and to capture relations between objects of the projective space (equality, collinearity, coplanarity, etc.) in a homogenous way. Taking advantage of the computational aspect of this approach, we automatically compute by saturation the ranks of all sets of the powerset of the points of the geometric configuration we consider. Upon completion of the saturation phase, our prover then retraces the proof process and generates the corresponding Coq code. This code is then formally checked by the Coq proof assistant, thus ensuring that the proof is actually correct. We use the prover to verify some well-known, non-trivial theorems in projective space geometry, among them: Desargues' theorem and Dandelin-Gallucci's theorem.

Keywords automated theorem proving · Coq · projective geometry · matroid theory · proof validation

1 Introduction

Recent proof developments achieved in our team [27,9,4] show that we need more automated tools, not only to simplify formal proofs carried out in a framework such as GeoCoq [10,32] but also to enhance constraint solving tools [36]. Therefore, to ease the proof process, we chose to develop a prototype theorem prover in the specific context of incidence geometry. Incidence geometry

David Braun
ICube UMR 7357 CNRS Université de Strasbourg, FranceE-mail: dmp.braun@gmail.com

Nicolas Magaud
ICube UMR 7357 CNRS Université de Strasbourg, France E-mail: magaud@unistra.fr

Pascal Schreck
ICube UMR 7357 CNRS Université de Strasbourg, France E-mail: schreck@unistra.fr

is a simple but powerful framework, which allows to state and prove some significant 3D theorems such as Desargues' theorem and Dandelin-Gallucci's theorem. In more advanced ways to deal with geometry, such as Tarski's approach, dealing with collinearity and coplanarity is even more challenging. Although our prover is designed with projective incidence geometry [17] in mind, it has no specific features preventing it from being used in a more general framework, such as affine geometry.

Instead of relying on the usual synthetic axiom system for projective geometry, we choose an approach based on the concept of ranks [29]. Ranks are related to matroid theory [33] and they allow geometric configurations to be described using only sets of points. Using ranks and the underlying matroid properties thus allows the switch to a combinatorial approach which is homogeneous and scalable to any dimension (including dimensions greater than 3). As we show in [12], we can switch rather easily from the synthetic description of projective geometry to a computational one based on ranks. Switching from the synthetic description to the combinatorial one boils down to changing from logic reasoning to computing. In [12], we also formally prove using Coq that this change of reasoning paradigm is sound and complete; it can thus be used in both directions. In this article, we mainly exploit the translation from the usual synthetic description to the combinatorial matroid-based one and build an automated prover relying on the matroid properties of ranks.

In order to develop an effective automated tool to prove geometry statements, we assume that we work in a *closed world*, meaning that the automated prover only deals with existing points and does not create any new points. Creating appropriate points to formally prove a statement is known to be difficult and creating inappropriate points may lead to a combinatorial explosion. If new points are required, then the user is in charge of adding such points explicitly in the context (e.g. by using some intersection existence theorems in the context of projective geometry).

The automated prover, named Bip for *matroid **B**ased **I**ncidence **P**rover*, is designed to prove equality between ranks of various sets of points. It is based on rank interval computations. For each subset of the powerset of the geometric configuration, we define the minimum and the maximum rank (in the worst case, when no information is known, the rank of each non-empty subset is between 1 and 4 for 3D configurations). We then use the matroid properties which are enforced by the rank function and reformulate them as rewrite rules to incrementally reduce the size of the interval for each subset. This is achieved using a saturation algorithm, which is run on a valuated graph implementing the inclusion lattice of the point powerset, labeled by the minimum and maximum rank. The saturation algorithm aims at computing the rank of some given subsets of the initial configuration. This can be achieved by computation, with an at-least exponential complexity. Using ranks hinders intuition and readability for the user. That is one of the reasons why we choose a formal approach to make sure the statement is correctly proved. Once the saturation graph is built, it is traversed to build a Coq [6,16] proof script which actually proves the statement at stake. Technical details about the implemen-

tation can be found in [11] and the current implementation of the prover as well as several examples of applications are available in the git repository: `https://github.com/pascalschreck/MatroidIncidenceProver`.

From a practical point of view, we start with a proof outline in a geometric setting (it can actually be a simple diagram, conjecturing a specific geometric statement holds). We then translate it into a formal description using ranks. This configuration (a list of subsets and their ranks) is fed to the automatic prover, which returns a Coq proof script (a simple `.v` file). The Coq file is then type-checked by Coq to make sure the proof is correct. The approach allows us to prove automatically some emblematic theorems of 3D, e.g. Desargues' theorem or Dandelin-Gallucci's theorem [13]. Typically our prover can handle geometric statements with an initial configuration of about 20 points (17 for Dandelin-Gallucci's theorem and 21 for 4D-Desargues' theorem). One of the original features of our work that it is based on a two-level approach, in which one independent program performs the proof search (using a saturation process) and then Coq verifies that the produced proof is actually correct and solve the goal.

Related Work Our work fits in the scope of automated theorem proving, especially its application to geometry. In geometry, most approaches rely on algebraic methods such as Wu's method [38]. More generally, the TPTP project [37] aims at providing a framework to test and evaluate automated theorem provers (ATP), *to help ensure performance results accurately reflect capabilities of the ATP systems*[1]. Among the most powerful Satisfiability Modulo Theories (SMT) provers, we can cite Paradox [15], Vampire [24], Z3 [31] or CVC5 [3]. They are powerful enough to automatically prove some non-trivial geometric theorems, expressed using first-order logic. However they only produce a decision: *satisfiable/unsatisfiable* and do not necessarily produces a proof trace, which allows to check independently that the proof is correct. A recent trend of research consists in adapting ATP to make them generate proof traces which can be verified by an external tool [2, 1, 20]. Tools such as SMTCoq [19] or CoqHammer [18] (inspired by Isabelle/HOL sledgehammer [7]) allow to connect directly an ATP to an interactive theorem prover: the ATP produces the proof and a trace of it, which can then be checked by the interactive theorem prover. Some of the most advanced results connect HOL-Light and CVC Lite [28] or the SMT-solver VeriT [8] and Coq [1].

Outline The article is organized as follows. Section 2 introduces two axiom systems for 3D projective incidence geometry: the first one in a synthetic way and the second one using ranks and matroids properties. Section 3 summarizes the main features of the algorithm onto which our prover is built. Section 4 presents some details of the implementation of our prover as well as a simple example. Section 5 explains how the prover generates some ready-to-be-checked Coq proof scripts. Section 6 presents some significant examples of geometric

[1] https://www.tptp.org/

theorems which have been proved automatically. In Section 7, we present some concluding remarks and draw some promising perspectives.

2 Projective Incidence Geometry and its Description using Matroids and Ranks

2.1 A Synthetic Axiom System for Projective Incidence Geometry

In the context of 3D, we provide an axiom system for projective space geometry (see Table 1 for a mathematical description and Fig. 1 for a more visual description). The system contains five axioms. The axiom (A1P3) expresses that, given 2 points, there exists a line that goes through these two points. *Pasch*'s axiom (A2P3) assumes that two coplanar lines always meet. The axiom (A3P3) expresses the unicity property. The axiom (A4P3) states that there is at least three points per line. Furthermore, we add the axiom *Lower-Dimension* (A5P3) to capture projective geometry in an at least 3-dimensional space (denoted by $\geq$3D). This axiom states that there exist two lines which do not meet.

Pasch's axiom does not limit the upper dimension, that is why, so far, our axiom system only captures an at-least three-dimensional geometry ($\geq$3D). It is possible to limit this spatial geometry by adding the optional axiom (A6P3) to constrain the dimension to be exactly 3. This axiom specifies that there is always one line intersecting three other lines.

(A1P3) Line-Existence : $\forall$ A B : Point, $\exists$ l : Line, A $\in$ l $\wedge$ B $\in$ l

(A2P3) Pasch : $\forall$ A B C D : Point, $\forall$ l_{AB} l_{CD} l_{AC} l_{BD} : Line,
A $\neq$ B $\wedge$ A $\neq$ C $\wedge$ A $\neq$ D $\wedge$ B $\neq$ C $\wedge$ B $\neq$ D $\wedge$ C $\neq$ D $\wedge$
A $\in$ l_{AB} $\wedge$ B $\in$ l_{AB} $\wedge$ C $\in$ l_{CD} $\wedge$ D $\in$ l_{CD} $\wedge$
A $\in$ l_{AC} $\wedge$ C $\in$ l_{AC} $\wedge$ B $\in$ l_{BD} $\wedge$ D $\in$ l_{BD} $\wedge$
($\exists$ I : Point, I $\in$ l_{AB} $\wedge$ I $\in$ l_{CD}) $\Rightarrow$
($\exists$ J : Point, J $\in$ l_{AD} $\wedge$ J $\in$ l_{BC})

(A3P3) Uniqueness : $\forall$ A B : Point, $\forall$ l m : Line,
A $\in$ l $\wedge$ B $\in$ l $\wedge$ A $\in$ m $\wedge$ B $\in$ m $\Rightarrow$ A = B $\vee$ l = m

(A4P3) Three-Points : $\forall$ l : Line, $\exists$ A B C : Point,
A $\neq$ B $\wedge$ B $\neq$ C $\wedge$ A $\neq$ C $\wedge$ A $\in$ l $\wedge$ B $\in$ l $\wedge$ C $\in$ l

(A5P3) Lower-Dimension : $\exists$ l m : Line, $\forall$ p : Point, p $\notin$ l $\vee$ p $\notin$ m

---

(A6P3) Upper-Dimension : $\forall$ l_1 l_2 l_3 : Line, $l_1 \neq l_2 \wedge l_1 \neq l_3 \wedge l_2 \neq l_3 \Rightarrow$
$\exists$ l_4 : Line, $\exists$ P_1 P_2 P_3 : Point, $P_1 \in l_1 \wedge P_1 \in l_4 \wedge$
$P_2 \in l_2 \wedge P_2 \in l_4 \wedge P_3 \in l_3 \wedge P_3 \in l_4$

Table 1 Standard axiom system for projective space geometry

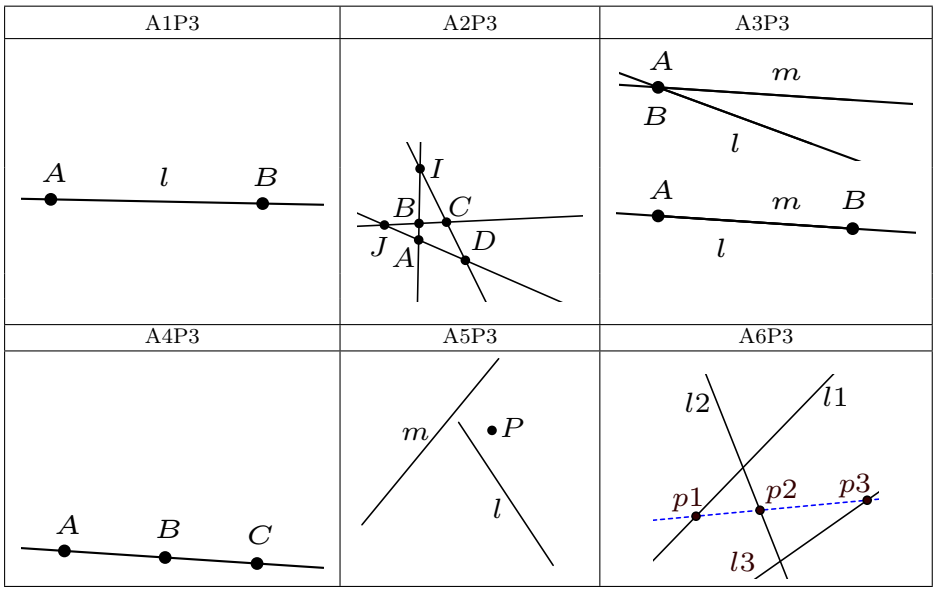

Fig. 1 Illustrations of the standard axiom system for projective space geometry

2.2 Matroids and Ranks

Using the standard axiomatization for projective space geometry of the previous section as a reference, we propose an alternative axiom system based on the concept of rank.

2.2.1 Matroid Properties

The concept of rank comes from matroid theory [33]. Matroids were introduced by Whitney in 1935 to abstractly capture the essence of dependence. Whitney's definition embraces a surprising diversity of combinatorial structures. Matroids apply to a large class of objects, ranging from graph theory or fields to greedy algorithms.

In our setting, matroids allow us to capture and generalize the main set properties of linear dependence in vector space. When combined with a finite set of points, it captures incidence (collinearity, coplanarity, ...) between these points. We propose to use the notion of rank, which respects the matroid properties to formalize the theory of projective incidence geometry. Using ranks allows us to deal only with points. This makes proof automation much easier because we do not handle directly lines or planes. Formally, an integer function *rk* on a finite set E is the rank function of a matroid if and only if conditions of Table 2 are satisfied.

(A1R3) non-negative and subcardinal : $\forall$ X $\subseteq$ E, 0 $\leq$ *rk* (X) $\leq$ |X|

(A2R3) non-decreasing : $\forall$ X $\subseteq$ Y, *rk* (X) $\leq$ *rk* (Y)

(A3R3) submodular : $\forall$ X,Y $\subseteq$ E, *rk* (X$\cup$Y) + *rk* (X$\cap$Y) $\leq$ *rk* (X) + *rk* (Y)

Table 2 Matroid properties of the rank function

2.2.2 Rank to Describe Projective Incidence Geometry

In the context of projective geometry, the rank function on finite sets of points is specified by the three conditions shown in Table 2. We specify the notions of closure and flat which are alternative axiomatizations of the matroids. Let M be a matroid on a finite set E with the rank function *rk* as above. The closure *cl* of a subset A of E is the set: $cl(A) = \{x \in E \mid rk(A) = rk(A \cup \{x\})\}$. A set whose closure equals itself is said to be closed or a flat [14]. A set is a flat if it is maximal for its rank, meaning that either the addition of any other element to the set would increase the rank, or the flat is the whole incidence space itself. In other words, the rank of a flat E is the cardinal of a smallest set generating E. Table 3 provides some examples of subsets and their ranks.

rk({A,B}) = 1	A = B
rk({A,B}) = 2	A $\neq$ B
rk({A,B,C}) = 2	A,B,C are collinear with at least two of them distinct
rk({A,B,C}) $\leq$ 2	A,B,C are collinear
rk({A,B,C}) = 3	A,B,C are not collinear
rk({A,B,C,D}) = 3	A,B,C,D are coplanar, not all collinear
rk({A,B,C,D}) = 4	A,B,C,D are not coplanar

Table 3 Some rank statements and their geometric interpretations

Using this definition, it can be shown that every projective space has a matroid structure, but the converse is not true. To capture projective geometry, we need to introduce some additional axioms to the matroid ones.

2.2.3 3D Rank-based Axiom System

In Table 4, we define a rank-based axiom system to describe projective space. Axioms (A1P3) *Line-Existence* and (A3P3) *Uniqueness* are subsumed by the three basic matroid properties of the rank function (presented in Table2). Axioms (A4R3) *Rk-Singleton* and (A5R3) *Rk-Couple* are added to scale the range of the rank function. Axioms (A6R3) *Rk-Pasch* and (A7R3) *Rk-Three-Points* are straightforward translations of the corresponding axioms of Table 1. (A8R3) *Rk-Lower-Dimension* and (A9R3) *Rk-Upper-Dimension* capture the dimension of the space we consider. As our prover Bip does not handle existential quantifications automatically, but rather leaves the user to deal with

these quantifications explicitly, simply setting bounds to the dimension, as do axioms (A8R3) and (A9R3), is sufficient to capture that the dimension of the space is exactly 3.

(A4R3) Rk-Singleton : $\forall$ P : Point, rk({P}) = 1

(A5R3) Rk-Couple : $\forall$ P Q : Point, P $\neq$ Q $\Rightarrow$ rk({P, Q}) = 2

(A6R3) Rk-Pasch : $\forall$ A B C D : Point, rk({A, B, C, D}) $\leq$ 3 $\Rightarrow$ $\exists$ J : Point, rk({A, B, J}) = rk({C, D, J}) = 2

(A7R3) Rk-Three-Points :
$\forall$ A B : Point, $\exists$ C, rk({A, B, C}) = rk({B, C}) = rk({A, C}) = 2

(A8R3) Rk-Lower-Dimension : $\exists$ A B C D : Point, rk({A, B, C, D}) $\geq$ 4

(A9R3) Rk-Upper-Dimension : $\forall$ A B C D : Point, rk({A, B, C, D}) $\leq$ 4

Table 4 Rank-based axiom system for projective space geometry

Both the synthetic and the rank-based axiom systems for projective incidence geometry are formalized in Coq. Their implementation is rather straightforward and relies on *type classes* to increase modularity of the code depending on the dimension. Once formalized in Coq, these two axiom systems are shown to be equivalent[12]. This equivalence allows back-and-forth translations between the synthetic description of projective geometry and its rank-based one. From now on, we shall only consider the rank-based axiom system and show how it can be used to build an automated prover for incidence geometry.

3 An algorithm based on interval reduction and saturation

Our algorithm relies on the following idea. The rank of a set of points is always bounded by a minimum value (at least 1 for a non-empty set) and a maximum value (the minimum of the cardinal of the considered set and of the rank of the whole space - 4 in the case of 3D configurations). The hypotheses of a geometric configuration may also narrow such intervals for some specific subsets. We then use the inequalities derived from the matroid properties to reduce the interval $[r_{min}; r_{max}]$ for each subset of points of the geometric configuration, in the same way as people do in interval arithmetic [30].

We start by introducing the main principles of our algorithm: transforming the input into an initial graph, propagating the constraints, saturating the problem and recording a trace, reconstructing the proof and finally using Coq to check whether the produced proof is correct. We then show that our algorithm behaves as expected and actually terminates.

3.1 Principle

Let us assume that we have a geometric statement expressed using rank equalities. Such a statement can either be written by hand or automatically derived from a synthetic statement. The conclusion of the statement and its hypotheses are translated into an inclusion lattice assigning already known data to the minimum rank and the maximum rank of all subsets of the initial geometric configuration. We then apply a saturation algorithm to carry out all the deductions which are possible in this lattice by using all the properties of Table 2.

In order to achieve that, these properties are transformed into the rules of Table 6 which can reduce the size of the intervals denoting the rank. The automated prover applies these rules, one after another, on all the subsets of the geometric configuration to tighten the interval between the minimum and maximum possible values for the rank. Each time a new deduction is made, the system updates the lattice accordingly, then propagating the new constraints to other subsets of points. The prover keeps computing new (smaller) intervals until the expected result (the conclusion of the geometric statement) is obtained or until there is no further deduction to make. In parallel, the prover records the application of each rule in order to rebuild the trace of the deductions leading to the computation of a new rank for a given set of points. This path in the graph is then extracted as a proof script and fed to the Coq proof assistant which verifies and validates it. The whole process is summarized in Fig. 2. Note that the saturation phase and the recording of the path are performed simultaneously.

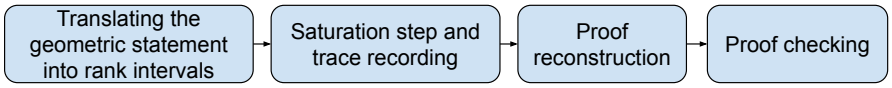

Fig. 2 The prover runs in four separate successive phases.

The prover can handle configurations in both projective and affine geometry. Indeed, it is only based on the matroid properties, which are independent of the axioms capturing the projective aspect of the geometric context. Thus deductions made by the prover are based on *pure* incidence geometry and the results hold in both projective and affine geometry. Dealing with projective geometry requires dealing with existential quantifications (e.g. for intersections of lines) and this difficult task is left to the user.

3.2 Rewrite rules

All the deductions made by the prover are performed using the basic matroid properties of the rank function rk, presented in Table 2. The other axioms (those of Table 4) are simply used to create new objects or to bound the

dimension. We thus remove from the saturation process all rules which feature an existential quantifier, e.g. (A6R3) or (A7R3). All points required to solve the problem need to be created before running the algorithm, so that the algorithm can use them. Axioms related to dimension may be used to tighten the interval $[r_{min}; r_{max}]$ for some subsets.

The matroid property (A1R3) is used to initialize the rank inteval for each subset. We transform the matroid properties (A2R3) and (A3R3) of ranks into eight properties, replacing the rank function by the min and max functions `rkMin` and `rkMax`. Let us consider X and Y two arbitrary subsets of points of the set of all points E. The sets X, Y, $X \cup Y$ and $X \cap Y$ must verify the properties listed in Table 5.

(PS1) $X \subseteq Y \subseteq E$, $rkMin(Y) \geq rkMin(X)$
(PS2) $X \subseteq Y \subseteq E$, $rkMax(X) \leq rkMax(Y)$
(PS3) $X, Y \subseteq E$, $rkMax(X \cup Y) \leq rkMax(X) + rkMax(Y) - rkMin(X \cap Y)$
(PS4) $X, Y \subseteq E$, $rkMin(X) \geq rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(Y)$

Table 5 Properties of rkMin and rkMax used until saturation.

In order to use the properties of Table 5 in the algorithm, we transform all of them into the (computable) rewrite rules of Table 6 which enable the update of the minimal rank and maximal ranks for some subsets of E. In the actual implementation, the rules RS1 to RS4 are duplicated to re-establish the symmetry between the variables denoting sets X and Y, thus making the implementation easier and the code more efficient.

(RS1) **if** $X \subseteq Y$ and $rkMin(X) > rkMin(Y)$ **then** $rkMin(Y) \leftarrow rkMin(X)$
(RS2) **if** $X \subseteq Y$ and $rkMax(Y) < rkMax(X)$ **then** $rkMax(X) \leftarrow rkMax(Y)$
(RS3) **if** $rkMax(X) + rkMax(Y) - rkMin(X \cap Y) < rkMax(X \cup Y)$
then $rkMax(X \cup Y) \leftarrow (rkMax(X) + rkMax(Y) - rkMin(X \cap Y))$
(RS4) **if** $rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(Y) > rkMin(X)$
then $rkMin(X) \leftarrow (rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(Y))$

Table 6 Rewrite rules used to reduce the interval of acceptable values for the rkMin and rkMax functions.

3.3 Termination, Correctness and Validation

Our automatic prover terminates by construction. But on the one hand, as mentioned above, it can fail, even if the property to be proved is valid. This is the case, for instance if some auxiliary points needed in the proof are not provided by the user. Our prover is therefore not complete. On the other hand, when it succeeds, there is no formal guarantee that it performed a correct

deduction. That is why it generates a trace which is then proof-checked using the Coq proof assistant. Correct proofs are only those accepted during the proof-checking process carried out in Coq.

4 Implementation

In this section, we present the concrete implementation of the prover in C[2]. We start with the data-structure we use and then present the algorithm in details. Finally, we summarize how the Coq script is generated. Due to the complexity in time and memory of the saturation algorithm, we choose to implement it as an independent program rather than embedding it into the Coq proof assistant.

4.1 Initialization of the algorithm

When starting the algorithm, the only input is the hand-written description of the geometric configuration together with the overall number of points involved. We start by building the powerset of this set of points. Each part represents a subset of points of the initial geometric configuration.

We encode each sub-set of points as well as the values of the minimal and the maximal rank as a 64-bit word. Rank values range from 1 to $\texttt{dim} + 1$ and subsets are represented using a boolean indicator function. Using an indicator function makes computing usual operations on ranks such as unions and intersections easy. By default, we bound the rank of each subset X of E with the minimum rank: 1 and the maximum rank: $min\{|X|, 4\}$ thanks to axioms (A1R3) and (A9R3). We then refine these ranks using the hypotheses of the geometric statement. In the following, we note **(RS0)** the application of this rule to initialize the ranks of a set of points, when working in 3D:

$$(\textbf{RS0}) \qquad \forall X, \texttt{non\_empty}(X) \Rightarrow 1 \leq rk(X) \leq min\{|X|, 4\}$$

4.2 Saturation loop

The saturation step consists in applying all rules to the powerset of the initial set of points. The overall idea is to make sure the extremal values for minimum rank and maximum rank are coherent with respect to the rules of our system. For instance, let us consider $X = \{A, B\}$ and $Y = \{A, B, C\}$ with the assumption $rkMin(X) \geq 2$. The initialization step yields that $rkMin(Y) \geq 1$ and $rkMax(Y) \leq 3$. However the property **(PS1)** is not verified by the sets X and Y. Indeed, the minimum rank of Y is smaller than the minimum rank

[2] The source code as well as some significant examples of use can be retrieved from the git repository : `https://github.com/pascalschreck/MatroidIncidenceProver`.

of X whereas $X \subseteq Y$. Therefore the minimum rank of Y is updated to express that it is greater than the one of X using the rule **(RS1)**.

To complete the saturation step, the algorithm 1 applies all the rewrite rules to each couple of (sub-)sets of points. If a rule is breached, the algorithm updates the subset whose minimum or maximum rank is incorrect. To ensure the correctness of the algorithm, we restrict the possible updates to two cases: we increase the minimum rank or we decrease the maximum rank while maintaining the invariant $RkMin < RkMax$. When the minimum and the maximum ranks are equal, we have reached the actual rank for this (sub-)set and it will not be modified any more. If the value of $RkMin$ happens to be strictly greater than the value of $RkMax$, then we face an incoherent context with contradictory hypotheses and the algorithm fails. Otherwise, the algorithm keeps applying the rewrite rules to all (sub-)sets until no more update has been carried out during the last iteration of the algorithm. Once the saturation step is completed, all possible deductions from the matroid properties have been made.

4.3 Recording the performed deductions

In parallel to the saturation step, the solver records all the deductions made in order to retrace a path from the conclusion to the hypotheses and to rebuild a formal proof to be fed to Coq in the end. Most deductions are not relevant to reach the expected conclusion from the hypotheses of the geometric statement. The proof path depends on the order used to traverse all subsets X and Y and on the order in which the rewrite rules are applied.

We build a directed acyclic graph (DAG) [3], named deduction graph (DG) in the following. Each node of the graph represents a subset together with its current ranks $RkMin$ and $RkMax$ as well as the rule applied to build this node. At the beginning, we build a node for each subset with ranks initialized using the rule **(RS0)**. When a rule improves the rank of a subset, a new node is added to the deduction graph (DG). This node contains the considered subset and its new ranks updated according to a rule of Table 6. To connect this node to the deduction graph (DG), we use the context of the rule that we just applied. The subset is thus connected by oriented edges to all the parts which occur in the premisses of the rule. This allows to trace the evolution of the interval between the minimum and the maximum ranks of a given subset by visiting its parent nodes.

The algorithm is summarized in 1. One may note that it consists in three main loops (one on all subsets X of E, one on all subsets Y of E, and one on all rewrite rules RSi). The subsets are traversed based on their indicator functions, represented as binary numbers, following the usual order on natural

[3] In theory, this graph is an hypergraph where edges are oriented and tagged by the applied rewrite rules.

numbers. Rules RSi are duplicated to maintain the symmetry of the computations, which is broken due to the scopes of the loop variables X and Y. In the actual implementation, there are 8 rewrite rules (RS1 to RS4 and their symmetric ones). In the pseudo-code, the properties (PSi) and the rules (RSi) are thus numbered from 1 to 8. Altogether its complexity is at least $2^n \times 2^n \times 8$, i.e. $2^{(2n+3)}$.

Algorithm 1: Saturation and construction of the deduction graph.

Input(s) : initialized deduction graph
Output(s): updated deduction graph

```
1  While something was modified in the previous iteration Do
2      ForEach subset X of E Do
3          ForEach subset Y of E such that X ≠ Y Do
4              ForEach rewrite rule (RSi) i ∈ [1;8] Do
5                  If the property (PSi with i ∈ [1;8]) is not verified Then
6                      Build a node in the deduction graph which records
7                      the application of the rewrite rule (PSi) and its
8                      arguments to update the minimum and maximum
9                      ranks of X and Y to ensure RkMin < RkMax
10                 End If
11             End ForEach
12         End ForEach
13     End ForEach
14 End While
```

4.4 A detailled small example

To illustrate the inner workings of our algorithm and show some intermediate states of the prover, we choose a very simple example of statement in 2D we may want to prove automatically.

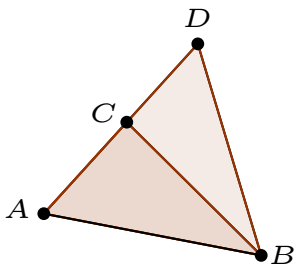

Fig. 3 The geometric configuration of Table 7.

```
(* An example of a 4-points lemma to be saturated *)
Lemma example : forall A B C D : Point,
rk(A::B::D::nil) = 3 ->
rk(A::C::D::nil) = 2 ->
rk(A::C::nil) = 2 ->
rk(C::D::nil) = 2 ->
rk(A:: B:: C::nil) = 3.
```

Table 7 An example of a geometric statement we want to saturate (using Coq syntax).

```
points A B C D
hypotheses
  A B D : 3
  A C D : 3
  A C : 2
  C D : 2
conclusion
  A B C : 3
```

Table 8 An example of a geometric statement we want to saturate (using the input language of our prover).

Let us consider the plane ABD. We build a new point C distinct from A and D lying on the line AD. In this case, we can deduce that the set of points ABC forms a plane (i.e. has rank 3). The initial Coq statement is presented in Table 7 and a geometric interpretation is provided in Fig. 3. In Table 8, our statement is described using the input language of our automatic prover. The input is structured using the keywords `points`, `hypotheses`, and `conclusion`. We first provide the list of all the points involved. Then the hypotheses, which are all of the form $\texttt{rk}\{A, B, C\} = 3$ are simply written as `A B C : 3`. Finally, one or more conclusion statements appear right after the keyword `conclusion`. The generated Coq proof script is available in Appendix A.

Before saturating the geometric statement, we first build the deduction graph (DG) which represents the initial configuration (Fig. 4). As the statement features 4 points, the initial layer of the deduction graph has $2^4 - 1$ nodes where each node represents a (non-empty) subset with its maximum rank on the left and its minumum rank on the right.

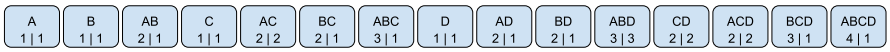

Fig. 4 The initialized deduction graph for the geometric configuration of Table 7.

We then run the saturation process once on all subsets of the initial set by applying all the rules in a specific order, say: RS1 RS3 RS2 RS4 RS5 RS7 RS6 RS8.We obtain a new improved but partial deduction graph (Fig. 5). Nodes with new information are printed in orange. The red digits denote the order in which the deductions were made by the algorithm.

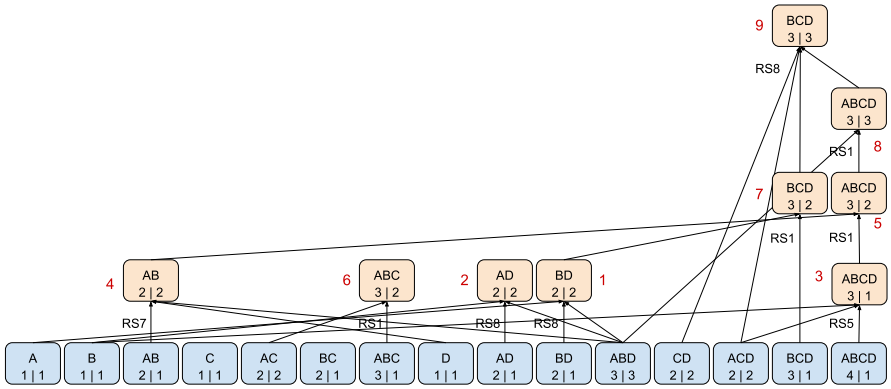

Fig. 5 The partially saturated deduction graph of the geometric configuration of Table 7.

Finally, the algorithm runs a second saturation loop. New nodes (in red in Fig. 6) are computed and we obtain the expected result: the rank of the sub-set of points ABC is exactly 3. We then complete the saturation process which leads to an additionnal result, which appears in the node 11. In this example, the minimum and maximum ranks are equal for all subsets of points. This may not always be the case depending on how well-constrained the problem is. If it is under-constrained, the algorithm can not find the exact rank of each subset. It can also be the case when it is well-constrained or over-constrained (the range of the ranks of some specific sub-sets can be larger than 1).

Each time, in a node, the minimal and the maximal rank get equal, a Coq statement and the associated proof script can be generated. Overall, while building the lemma (and proof) that $rk(\{A, B, C\}) = 2$, we obtain a by-product lemma (and proof) which states that $rk(\{A, B, C, D\}) = 3$.

5 Extracting the proof

Once the proof search phase is completed, we generate a proof trace for Coq. It consists in the statement of the expected theorem, followed by a sequence of tactics retracing the actions of the automatic prover. This sequence can be replayed by Coq and thus allows to validate the proposed proof.

5.1 Proof script generation and tagging of nodes

The third step consists in building a proof script acceptable by Coq which proves the expected results using the deduction graph. This is achieved by a recursive postfix traversal starting from the node whose rank was searched for. Indeed, we need for each node of the graph to build its children nodes before generating the piece of script corresponding to this node. Fig. 7 shows the postfix traversal in our running example, reconstructing the proof that

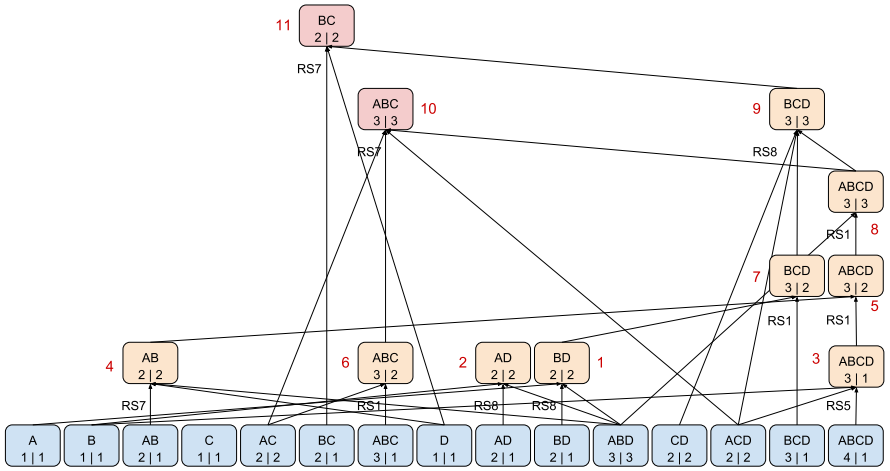

Fig. 6 The deduction graph fully saturated of the geometric configuration of Table 7.

the subset ABC represents exactly a plane. Nodes printed in green are those involved in the postfix traversal and they are annotated in blue with the order in which the proofs of the nodes are built to be used in the subsequent deductions.

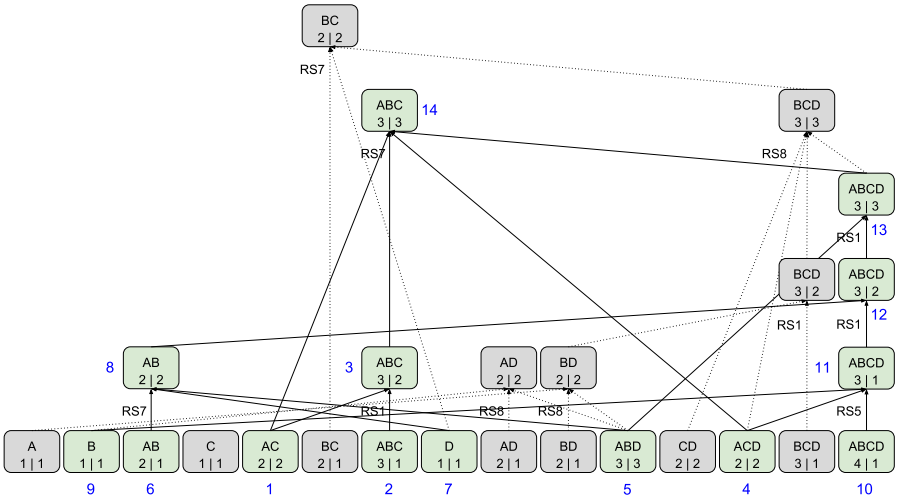

Fig. 7 Reconstruction path in the deduction graph correspondong to the geometric configuration of Table 7.

Not all nodes are useful to build the proof script. Some nodes may however be traversed several times. We add a tagging mechanism to classify the nodes and thus avoid unnecessary traversals. The tag can have five different values:

- 0: the node does not belong to the reconstruction path of the proof.
- 1: the node belongs to the reconstruction path and has not been handled yet.
- 2: the node has already been handled but subproofs brought by its children node are still expected.
- 3: the proof of this node has already been reconstructed in the currrent proof being processed.
- 4: the proof of this node has already been reconstructed in another lemma.

```
(* previous block *)
...

(* block of the result we search *)
assert(Hx : rk(P1 :: P2 :: P3 :: nil) >= 3).
{
        (* checking the hypotheses for soundness *)
        ...
        (* simplifying the intersection and union of subsets *)
        ...
        (* applying the appropriate rule *)
        assert(HT := rule_2 ...);apply HT. (* applying RS7 *)
}

(* next block *)
...
```

Table 9 A block corresponding to the application of rule RS7.

Tag 0 automatically removes the node from the reconstruction process for the considered lemma. Tag 1 is used during the preprocessing step to traverse all nodes needed to reconstruct the proof. This preprocessing step is unavoidable when the theorem is decomposed into intermediate lemmas. Tag 2 states that the proof of this node is about to be reconstructed. Tag 3 states that the node has already been traversed and reconstructed in the proof currently being reconstructed. Tag 4 is used to distinguish between nodes which have been previously reconstructed in other proofs. These proofs now form the intermediate lemmas which can be reused in other reconstruction steps.

Then, to complete the reconstruction of the proof, the algorithm translates each rewrite rule into a piece of code in Coq. This is embedded into a block containing all the proof steps required to the minimum and maximum ranks of the reconstructed node. In the block, we check that the required hypotheses are available in the context and then prove the expected result. A simplified view of a block is shown in Table 9. The recursive algorithm 2 summarizes the

reconstruction step for a node. The initial tag 1 or 4 (for already reconstructed lemmas) is set during the preprocessing phase in the main function.

Algorithm 2: Reconstruction step.

Input(s) : Node to be reconstructed
Output(s): A text file with the corresponding Coq code

```
1  If the node has some children Then
2      ForEach child Do
3          If child.tag == 1 Then
4              child.tag ← 2
5              Algorithm 2 (child)
6          End If
7      End ForEach
8  End If
9  The node is tagged with 3
10 Generation of the Coq code associated to the application of the rule
     stored in the node
```

5.2 Validation by the Coq proof assistant

The generated proof block is then easily exported into some *Gallina* code[4] which contains the statement and its hypotheses, together with the formal Coq script proving it. All this is ultimately checked correct by the Coq proof assistant. In our Gallina output, we choose to implement sets of points as lists of points. The reconstruction process respects some naming conventions to make the Coq proof script generation simpler: name spaces for local and global hypotheses are different. Names of intermediate lemmas are automatically determined, thus being easier to retrieve and reuse in subsequent proof steps. Likewise, tactics are chosen to be as simple as possible, so that the performance of the proof checking step, both with respect to time and memory usage, remains acceptable even if the proof is huge.

Structuring the proofs using blocks allows the handling of hypotheses with the same scope rules as in many programmming languages and avoids cluttering the system with meaningless or inrelevant hypotheses. All local hypotheses introduced in a block are discharged as soon as the expected result for the block is obtained. The only remaining hypothesis is the global one declared right before the beginning of the block. This means names can be reused in local hypotheses. For global hypotheses, they are named according to the points at stake together with the minumum and maximum ranks, e.g `HP3P6P9m2` (resp. `HP4P8P9M3`) states that $rk\{P3, P6, P9\} \leq 2$ (resp. $rk\{P4, P8, P9\} \geq 3$). Symbols `m` and `M` stand for minimum and maximum respectively.

To improve the proof checking step, we need to decompose the proof into several intermediate lemmas, which must be stored and included as global

[4] *Gallina* is the specification language of Coq: a full description of it can be found in Coq reference manual [16].

hypotheses whenever the system requires them (this corresponds to lemmas whose node is tagged with 4). This decomposition reduces the size of the proof and the number of hypotheses to be dealt with. Overall this leads to smaller proof terms which once proven can be freely reused in other proofs.

Finally we carefully ensure that all proof steps in Coq are as efficient as possible. We avoid using generic automated deduction procedures (e.g. `tauto`, `intuition` or `lia`) and rather use explicit, straight-to-the-point, immediate tactics. In most cases, these tactics simply consist in applying a specific lemma to the current goal, thus being completely deterministic and do not require any (costly) proof search steps. Witness generation and the like is handled by the C program generating the proof script.

6 Some results

Incidence geometry theorems are stated using a geometric rank function. Proving them can be seen as solving a constraint problem where the unknown is the rank function itself. The valuation is only known for a few specific sets of points (which correspond to the hypotheses of the statement). After the first toy example presented in the previous section, we now consider more challenging configurations to illustrate the ways our solver can be used.

6.1 Instructions for use and first example

Whereas the first version of Bip required examples to be hard-coded in a C file and the solver to be re-compiled each time a new problem was submitted, the current version offers a friendlier *modus operandi*. The statement is written in a text file describing the context (for instance the dimension of the considered incidence space) and the list of the ranks which are imposed. The solver is then launched with this file as an argument and it produces two files. The first one is a text file with the rank of each subset, or at least the best interval rank found by Bip. It is by essence a large file and so far we simply use it for verification and debugging purposes. However, it could also be used for discovering new theorems (derivable from the context). The second one is a Coq file which contains the proof of the theorem in the case where the Bip prover terminates successfully.

Let us try it on a simple but significant example. We want to prove that in a 3D projective space two different planes intersect along a line. As discussed above, the question of the existence of some points is not considered by Bip and this is actually false in dimension 5. The characterization of the dimension is out of the scope of Bip in terms of existence of intersection points. Instead, we just prove that, in 3D, if there are two points, say M and N in the intersection of two planes, then the line defined by points M and N is the intersection of the two planes (See Figure 8).

We first prove that the intersection is included in the line spanned by M and N (in short line (MN)). The two planes are each defined by three points

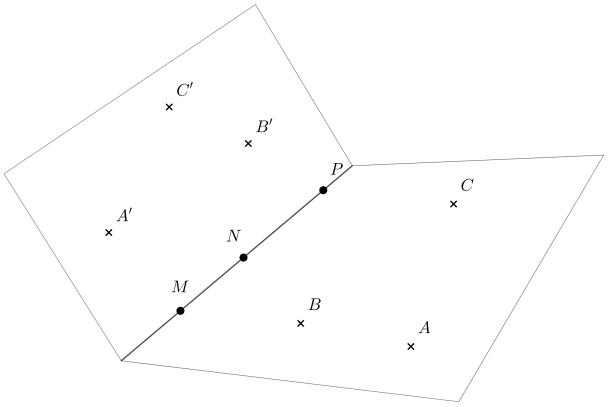

Fig. 8 Intersection of two planes

```
points
  A B C Ap Bp Cp M N P    # Ap is for A'
hypotheses
  A B C : 3               # A B C define a plane
  Ap Bp Cp : 3            # Ap Bp Cp define a plane
  A B C Ap Bp Cp : 4      # the planes are different
  M A B C : 3             # M belongs to (ABC)
  N A B C : 3
  P A B C : 3
  M Ap Bp Cp : 3          # M belongs to (A'B'C')
  N Ap Bp Cp : 3
  P Ap Bp Cp : 3
  M N : 2                 # points M and N are different
conclusion
  M N P : 2               # points M N and P are collinear
```

Table 10 In 3D, the intersection of two planes is included into a line

—A, B and C for the first one, and A', B' and C' for the second one— such that $rk\{A, B, C\} = 3$ and $rk\{A', B', C'\} = 3$. Note that $rk\{A, B, C\} = 3$ implies that all three points are different. The property that two planes are different is translated into $rk\{A, B, C, A', B', C'\} = 4$, i.e. the six points span the whole space. Then, $rk\{A, B, C, M\} = 3$ and $rk\{A, B, C, N\} = 3$ express that points M and N are in the plane (ABC). We do the same with the plane (A'B'C'). We finally obtain the formal description shown in Table 10, which will be the input of our prover.

The size of the output files is dependent of the number of points in the statement. As the statement is quite small and only contains a few points (9), the size of the output file with the whole rank function is only about 58 Kb, and the size of the proof in Coq is about 43 Kb (and a bit less than 1000 lines including 500 lines of preamble). The corresponding Coq statement is shown

```
 Lemma LMNP : forall A B C Ap Bp Cp M N P ,
rk(A :: B :: C ::  nil) = 3 ->
rk(Ap :: Bp :: Cp ::  nil) = 3 ->
rk(A :: B :: C :: Ap :: Bp :: Cp ::  nil) = 4 ->
rk(A :: B :: C :: M ::  nil) = 3 ->
rk(Ap :: Bp :: Cp :: M ::  nil) = 3 ->
rk(A :: B :: C :: N ::  nil) = 3 ->
rk(Ap :: Bp :: Cp :: N ::  nil) = 3 ->
rk(M :: N ::  nil) = 2 ->
rk(A :: B :: C :: P ::  nil) = 3 ->
rk(Ap :: Bp :: Cp :: P ::  nil) = 3 ->
rk(M :: N :: P ::  nil) = 2.
```

Table 11 Lemma expressing that the intersection of two planes is included into a line

```
points
  A B C Ap Bp Cp M N P
hypotheses
  A B C : 3              # A B C define a plane
  Ap Bp Cp : 3           # A' B' C' define a plane
  A B C Ap Bp Cp : 4     # the planes are different
  M A B C : 3            # M belongs to (ABC)
  N A B C : 3
  M Ap Bp Cp : 3         # M belongs to (A'B'C')
  N Ap Bp Cp : 3
  M N : 2                # points M and N are different
  M N P : 2              # M N and P are colinear
conclusion               # now, there are two terms into the conclusion
  A B C P : 3            # P belongs to (ABC)
  Ap Bp Cp P : 3         # P belongs to (A'B'C')
```

Table 12 In 3D, the intersection of two planes is included into a line

in Table 11. Generating the proof takes less than 1s and the verification by Coq only few seconds.

Let us then prove that $(MN) \subset (ABC) \cap (A'B'C')$. The statement is very similar (See Table 12) except that the conclusion is now the conjunction of two equalities. In this case, Bip produces a Coq proof which includes, among others, two lemmas, one for each statement of the conjunction. The size of the produced files is close to the size of the previous theorem, even if we consider a double conclusion.

6.2 Desargues' theorem

Desargues's theorem is very important in projective incidence geometry. It is a plane theorem whose proof requires to consider a three-dimensional setting: more precisely, this is a theorem about a plane figure embedded in a 3D space. Desargues statement in 2D deals with couples of triangles (embedded in a single plane) whereas the 3D version deals with couples of tetrahedrons. The proof of Desargues in 2D only works when the plane which contains both

triangles can be embedded into a 3D space. It is false in a strict 2D setting. For instance, Moulton's plane is a famous example of a 2D model of incidence geometry where the property of Desargues does not hold [26]. The proof of the 2D property relies on a proof of an auxiliary lemma (carried out in a 3D setting, considering a tetrahedron traversed by a plane). We call such a configuration a 2.5D configuration of Desargues (i.e. a 2D configuration embedded into a 3D space) to avoid the confusion with the usual 3D configuration.

Theorem 1 (Desargues' theorem) *Let E be a 3D projective space and P, Q, R, P', Q', R' be points of this space. Let PQR and $P'Q'R'$ be 2 non-degenerated triangles. If lines (PP'), (QQ') and (RR') intersect in a single point O, then points $\alpha = (PR) \cap (P'R')$, $\beta = (QR) \cap (Q'R')$ and $\gamma = (PQ) \cap (P'Q')$ are on the same line.*

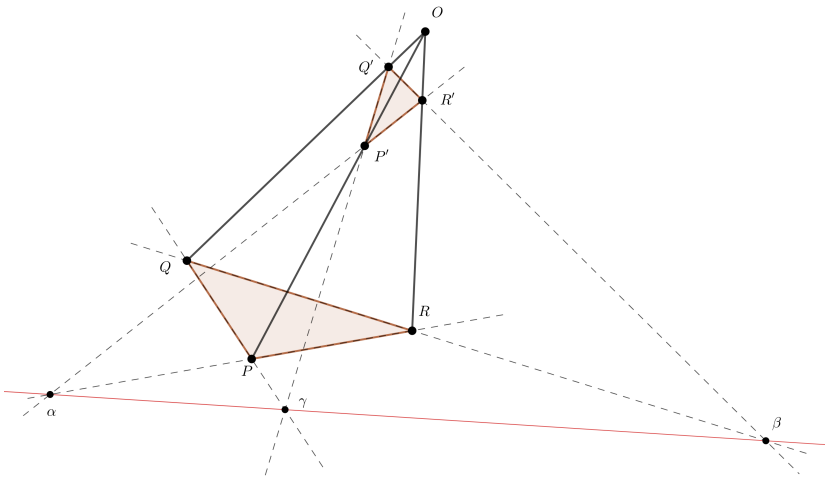

Fig. 9 Desargues' theorem 1, in its 2.5D configuration.

This theorem was proven interactively using Coq a few years ago [23]. It proceeds as follows: we prove the property in the case the two triangles are not coplanar. This configuration is called Desargues 2.5D (as shown in Fig. 9). We then derive a proof that it holds in 2D when the 2 triangles are coplanar (as shown in Fig. 10).

6.2.1 Case where triangles PQR and $P'Q'R'$ are not coplanar

This is the easy part. The main feature of the proof is that the intersection of two planes PQR and $P'Q'R'$ is a line which contains points α, β and γ. But, taking a deeper look to the statement, it can be observed that implicitly

points P and P' must be different as well as Q and Q', and, R and R'. It is less obvious to observe that the point O has to be different to at least one point among P, Q, R, P', Q', R'. We face here a well known problem about the non-degeneracy conditions which can be numerous but still difficult to find. We choose here to describe the general case where the point O neither belongs to the plane (PQR) nor to the plane $(P'Q'R')$. The corresponding statement is given at Table 13

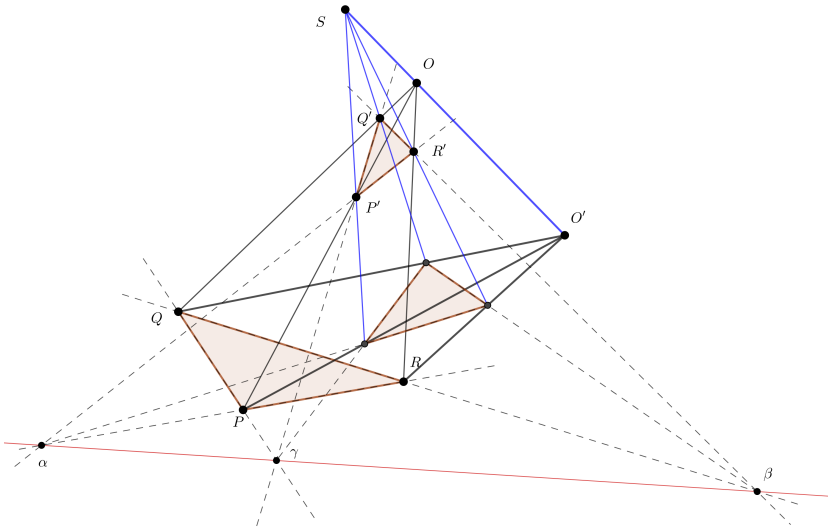

Fig. 10 Desargues' theorem 1, in its 2D configuration together with the extrusion of triangle P'Q'R' to 3D

The sizes of the output files are respectively 118 Kb for the rank function —the size approximately doubles each time a point is added— and about 380 Kb for the Coq proof, with around 4 900 lines.

6.2.2 Case where both triangles PQR, $P'Q'R'$ and the perspective point O' lie in the same plane Π

In this situation, one cannot use the fact that two planes intersect each other along a line because the figure lies in an unique plane Π. The key is to go back to the previous case by extruding the figure using a point outside plane Π:

- first choose a point, say S, outside of the plane Π and consider the tetrahedron $SP'Q'R'$,
- second choose a point, say O, on the line (SO') and different from S and O',
- lines OP and (SP') are coplanar (they belong to the plane $(PO'S)$), then they meet in a point we call P'',

```
 points P Q R Pp Qp Rp Oo alpha beta gamma
    # O means zero in Coq, we use Oo instead
hypotheses
    P Pp : 2
    Q Qp : 2
    R Rp : 2
    P Pp Oo : 2
    Q Qp Oo : 2
    R Rp Oo : 2
    P Q R Oo : 4            # Oo is not in plane PQR
    Pp Qp Rp Oo : 4
    P Q R : 3
    Pp Qp Rp : 3
    P Q R Pp Qp Rp : 4      # non coplanarity
    P Q beta : 2            # alpha, beta, gamma definitions
    Pp Qp beta : 2
    P R alpha : 2
    Pp Rp alpha : 2
    Q R gamma : 2
    Qp Rp gamma : 2
conclusion
    alpha beta gamma : 2
```

Table 13 Statement for the 2.5D version of Desargues's theorem

– construct Q'' and R'' in the same way.

We thus have two tetrahedrons, one with apex S and the second with apex O, with a common triangle $P''Q''R''$, where we can apply the previous reasoning. The construction is more intricate and there are more non-degeneracy conditions to consider. Actually, in the statement given in Table 14, we examine all the triples in the plane Π:

– some have a rank 2, for instance points P, P' and O' have to be collinear,
– some have rank 3, for instance the fact P, Q and P' cannot be collinear because this implies that lines (PQ) and $(P'Q')$ are equal and then the point γ cannot be defined,
– and finally, some can have rank 2 or 3 for instance the set (P, Q', R'): for such triples imposing a rank entails the generality of the theorem.

With 15 points, it takes less than 2 minutes to solve this statement. The file which contains the rank function has a size of 4 Mb (a bit more than 32 × 118 Kb) and the Coq file is about 1.0 Mb and a bit less than 13 000 lines.

6.2.3 One step beyond

Desargues's theorem has a very combinatorial nature and it has in fact a version in any dimension greater than 2. For instance in dimension 3, it states that given two tetrahedrons T and T' which are in perspective from a point O, the six points defined by the intersection of the corresponding edges of T and T' are in a plane and form a complete quadrilateral as shown in Fig. 11

```
  points P Q R Pp Qp Rp Ps Qs Rs Op Oo Sc alpha beta gamma
             # O means zero in Coq, thus O -> Oo
             # ... the same for S, thus S -> Sc
             # moreover Pp stands for P' and Ps stands for P''
  hypotheses
      P Q R : 3       # independence relations headed by point P
      P Q Pp : 3      # for instance this implies that P Q Pp Qp and Op are collinear
      P Q Qp : 3      # idem
      P Q Op : 3      # idem
      P R Pp : 3      # idem for P Pp R Rp and Op
      P R Rp : 3
      P R Op : 3
      P Pp Qp : 3
      P Pp Rp : 3
      P Pp Op : 2     # collinearity
      P Qp Op : 3
      P Rp Op : 3     # end for point P
      Q R Qp : 3
      Q R Rp : 3
      Q R Op : 3
      Q Pp Qp : 3
      Q Pp Op : 3
      Q Qp Rp : 3
      Q Qp Op : 2      # collinearity
      Q Rp Op : 3      # end for point Q
      R Pp Rp : 3
      R Pp Op : 3
      R Qp Rp : 3
      R Qp Op : 3
      R Rp Op : 2      # collinearity, end for point R
      Pp Qp Rp : 3
      Pp Qp Op : 3
      Pp Rp Op : 3             # end for point Pp
      Qp Rp Op : 3
      P  Q  R Pp Qp Rp Op : 3  # coplanarity of the initial figure
      P  Q  R  Sc : 4          # point Sc is chosen out of plane PQR
      P  Q  R  Oo : 4          # the same for point Oo
      Oo Sc : 2                # Sc and Oo are différent
      Op  Oo  Sc : 2           # and collinear with Op
      Ps P  Oo : 2             # definition of Ps
      Ps Pp  Sc : 2
      Qs Q  Oo : 2             # definition of Qs
      Qs Qp  Sc : 2
      Rs R  Oo : 2             # definition of Rs
      Rs Rp  Sc : 2
      P  R  alpha : 2          # definition of alpha
      Pp Rp  alpha : 2
      Q  R beta : 2            # definition of beta
      Qp Rp  beta : 2
      P  Q gamma : 2           # definition of gamma
      Pp Qp  gamma : 2
  conclusion
      alpha beta gamma : 2
```

Table 14 Statement for Desargues's theorem in 2D

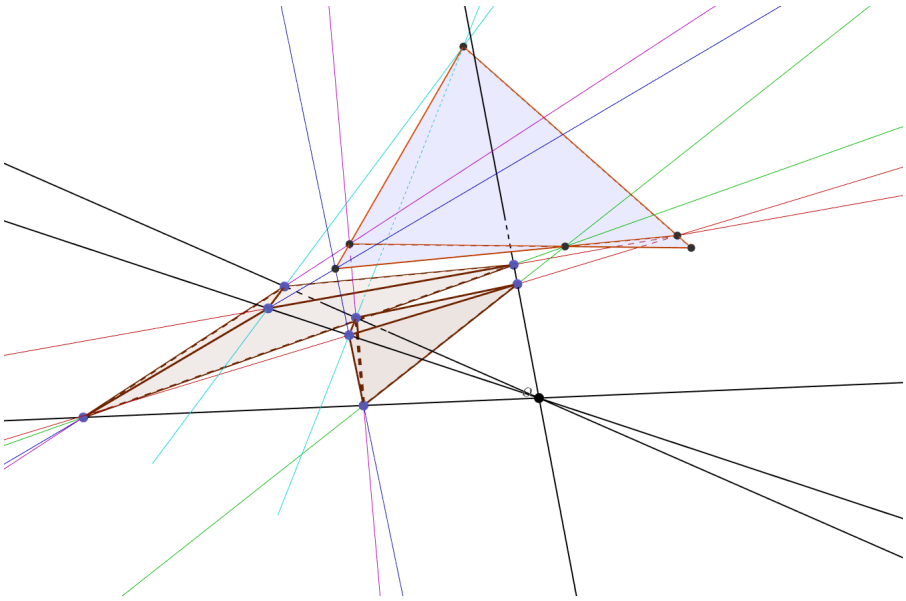

Fig. 11 Desargues's theorem in 3D

[34]. 15 points are involved in this theorem but it is solved in few minutes. The Coq proof is about 31 000 lines long.

We also succeed in proving the 4D version of this theorem: given two pentachores P and P' which are in perspective from a point O, the ten points defined by the intersection of the corresponding edges belong in a 3D space and form a figure which is precisely the one depicted in Figure 9. 21 points are involved in this theorem, and it takes about one week to solve it [35]. The Coq file is 47.4 Mb large. It contains 497 157 lines (with a lot of comments) and features 2517 intermediate lemmas.

In dimension n, let us call a n-complete hyper-tetrahedron the configuration G of $\frac{(n+1)(n+2)}{2}$ points and defined by $G = H \cup H'$ where

- H is a set of $n+1$ points in general position, that is a hypertetrahedron in dimension n,
- H' is the intersection of H by an independent hyperplane P; in other words, H' is a set of $\binom{n+1}{2}$ new points each of them is the intersection of an edge of H with hyperplane P.

A 2-complete hypertetrahedron is a complete quadrilateral and a 3-complete hypertetrahedron is the configuration depicted at Figure 9. We then have the following theorem:

Theorem 2 (Desargues's theorem in dimension $n+1$**)** *Let E an incidence space of dimension $n+1$ and two hypertetrahedrons T and T' in this space in perspective from a point O independent of each tetrahedron, that is,*

O does not belong to any hyperface of T or T'. Then the intersection points of the corresponding edges of T and T' form a n-complete hypertetrahedron.

We do not have yet a formal proof of this theorem and it is out of reach of our prover since it requires a proof by recursion on the dimension of the space. A proof of a more complicated formulation can be found in [5]. Note also that there exist other generalizations of Desargues' theorem [22].

6.3 Dandelin-Gallucci's theorem

In an incidence space with dimension at least 3, Dandelin-Gallucci's property can be informally stated as follows:

Property 1 (Dandelin-Gallucci) Let us assume we have three skew lines a, b and c and three other skew lines e, f and g, such that every line in $\{a, b, c\}$ meets every line in $\{e, f, g\}$. Then, all pairs of lines d and h, such that d meets lines e, f and g, and such that h meets every line a, b and c, are concurrent.

It is important to note that this property is not satisfied by every projective incidence space. In fact it is related to Pappus' property which lives in a 2D plane and can be stated as follows:

Property 2 (Pappus) In a projective incidence plane, let a and e be two distinct lines and let A, B, C, A', B' and C' be six different points with A, B and C belonging to a and A', B' and C' belonging to e. These points define respectively the lines $l_{AB'}, l_{A'B}, l_{AC'}, l_{A'C}, l_{BC'}, l_{B'C}$. The three intersection points $X = l_{AB'} \cap l_{A'B}$ and $Y = l_{AC'} \cap l_{A'C}$ and $Z = l_{BC'} \cap l_{B'C}$ are collinear.

Dandelin-Gallucci's *theorem* then establishes a strong link between an iconic 2D property (Pappus) and a truly 3-dimensional one (Dandelin-Gallucci):

Theorem 3 (Dandelin-Gallucci) *In a projective incidence space whose dimension is greater than or equal to 3, Dandelin-Gallucci's property and Pappus' property are equivalent.*

This proof only uses basic knowledge on incidence geometry. Fig. 12 illustrates the configuration and names some interesting lines. This proof sketch highlights the role of Pappus points X, Y and Z which are not part of the initial configuration and are later used to construct the point R as the intersection of lines (YM) and (ZN). We then show that this new point R is also the intersection of lines d and h. The details of the proof can be found in Horváth's article [21].

6.3.1 From Pappus to Dandelin-Gallucci

As mentioned above, the points X, Y and Z correspond to an instance of Pappus' property. This instance is chosen by hand. There are many possibilities

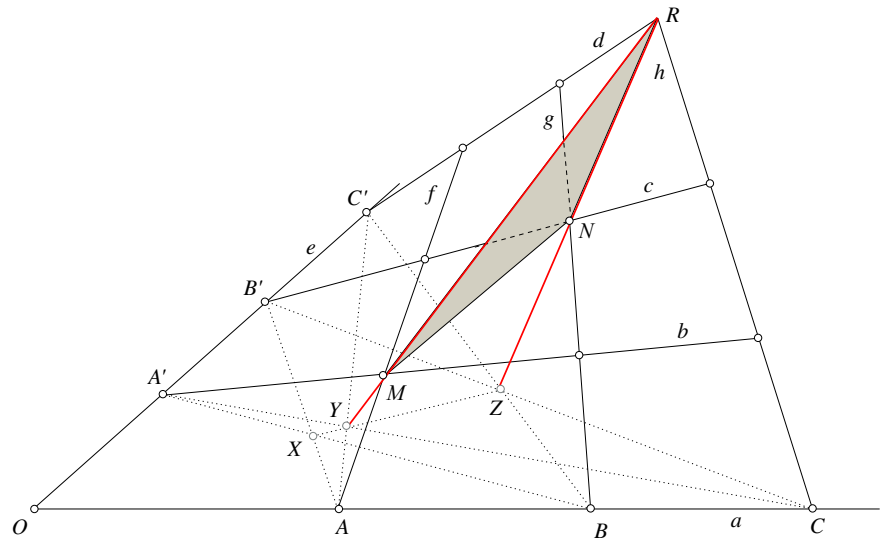

Fig. 12 From Pappus to Dandelin-Gallucci (lines are noted with lower case letters a, b, ...).

among such configurations: 9 pairs of lines can be chosen and for each pair there are 6 possible configurations giving at least 56 Pappus's configurations resulting in 168 points which could be added. Considering the exponential complexity of our naive algorithm, this is impossible to manage. This triple of points being defined, we directly add the collinearity of these points as an hypothesis. Generating automatically this collinearity property could also be achieved by introducing a new rule corresponding to Pappus' property in our system. However, as the points involved in the instance of Pappus' property were already selected by hand, it would have been pointless to let the system search for six points (the six input points of Pappus' property so that it can apply Pappus' property) and thus re-discover the above-mentionned triple of three collinear points.

The point R is then defined as the intersection of lines (YM) and (ZN) (See Figure 12), but the coplanarity of these two lines has to be proved before. This is why there are two parts of the proof. We just give here the second statement which contains all the constraints (all the examples can be found on the git repository already mentioned). The input statement provided to Bip is given in Table 15 and should be easily readable. The introduction of all the points follows a constructive definition of the figure, the names of the lines defined by collinearity are given in comments. The whole figure contains 19 points but only 17 of them are required giving lighter outputs: the size of the rank function file is about 16.4 Mb and the Coq proof about 2 Mb, that is 25 000 lines. The computation time is about half an hour.

```
   points
        Oo A B C Ap Bp Cp X Y Z M N Sp T U V R # points P and Q are not needed
   hypotheses
       Oo A : 2
       Oo B : 2
       Oo C : 2
       A B : 2
       A C : 2
       B C : 2
       Oo Ap : 2
       Oo Bp : 2
       Oo Cp : 2
       Ap Bp : 2
       Ap Cp : 2
       Bp Cp : 2
       Oo A B C : 2              # a
       Ap M Sp : 2               # b
       Oo A B C Ap M Sp : 4      # a and b are not coplanar
       Bp N T : 2                # c
       Oo A B C Bp N T : 4       # a and c are not coplanar
       Ap M Sp Bp N T : 4        # b and c are not coplanar
       Cp U V : 2                # d
       Oo A B C Cp U V : 4       # a and d are not coplanar
       Ap M Sp Cp U V : 4        # b and d are not coplanar
       Bp N T Cp U V : 4         # c and d are not coplanar
       Oo Ap Bp Cp : 2           # e
       Oo A Ap : 3               # a and e are different
       A M U : 2                 # f
       Oo Ap Bp Cp A M U : 4     # e and f are not coplanar
       B N V : 2                 # g
       Oo Ap Bp Cp B N V : 4     # e and g are not coplanar
       A M U B N V : 4           # f and g are not coplanar
       C Sp T : 2                # h
       Oo Ap Bp Cp C Sp T : 4    # e and h are not coplanar
       A M U C Sp T : 4          # f and h are not coplanar
       B N V C Sp T : 4          # g and h are not coplanar
       X A Bp : 2                  # 1st Pappus point
       X Ap B : 2
       Y A Cp : 2                  # 2nd Pappus point
       Y Ap C : 2
       Z C Bp : 2                  # 3rd Pappus point
       Z Cp B : 2
       X Y Z : 2                   # Pappus colinearity
       Y M R : 2                   # addition of R
       Z N R : 2
 conclusion
   C Sp T Cp U V : 3               # lines d and h are coplanar
```

Table 15 Statement for the Pappus to Dandelin-Gallucci direction.

6.3.2 From Dandelin-Gallucci to Pappus

The opposite direction of the proof is similar but a lot of points have to be added by hand: 10 points are involved by the hypotheses and 17 points are needed by the proof. These points all come from the construction of a Dandelin-Gallucci configuration which is done by hand following Horváth's article [21]. With 17 points, the size of the rank function is the same as before, but the Coq proof is a bit longer with 76 000 lines.

7 Conclusion

7.1 Achievements

In this article, we present an automated prover, called Bip. It is designed to carry out proofs of geometric statements in an incidence projective geometry setting. It relies on the concept of rank of a set of points to capture the usual notions of geometry (colinearity, coplanarity, etc.). By dealing only with points, it provides an homogeneous context to carry out proofs by computation. Starting from the hypotheses, which are expressed as rank equalities for some specific sets (e.g. $rk\{A, B, C\} = 3$ expresses that the three points A,B and C are coplanar), it proceeds by saturation of the context to make deductions which allow to prove the conclusion of the theorem at stake. These deduction steps rely on the matroid properties of the rank function. They are recorded and then used to produce a Coq proof script which can then be verified by Coq to ensure the considered statement is actually correct and well-proved. We show that our prover performs well on well-known emblematic theorems of projective incidence geometry such as Desargues' theorem or Dandelin-Gallucci's theorem. We also manage to prove statements (e.g. Desargues' theorem) in spaces of dimension greater to 3.

7.2 Future work

So far, our prover is called outside Coq to prove statements in one go. We aim at having a more integrated system in the Coq proof environment. We designed a prototype interface which allows to embed the prover and call it interactively from Coq [25], allowing for more flexible interaction between automated and interactive theorem proving. The user would only take care of creating new points (if required) and the automated prover would deal with all the other proof details automatically. This would allow to keep the structure and the key steps of the proof visible, and, at the same time, to deal with more technical parts of the proof automatically.

In the near future, we shall carry on proving statements in higher order dimensions, which means studying configurations with more points and larger initial rank intervals. Successfully proving such statements automatically requires to improve the performances of the prover. We may also consider data

mining approaches to discover new theorems of a given configuration, once the saturation process is completed. Once relevant statements are identified, the prover could generate the corresponding Coq proof scripts.

References

1. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In: International Conference on Certified Programs and Proofs, pp. 135–150. Springer (2011)
2. Armand, M., Grégoire, G., Keller, B., Théry, L., Werner, B.: Verifying SAT and SMT in Coq for a Fully Automated Decision Procedure. In: International Workshop on Proof-Search in Axiomatic Theories and Type Theories (PSATTT'11) (2011)
3. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: D. Fisman, G. Rosu (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 13243, pp. 415–442. Springer (2022). DOI 10.1007/978-3-030-99524-9_24. URL `https://doi.org/10.1007/978-3-030-99524-9_24`
4. Beeson, M., Narboux, J., Wiedijk, F.: Proof-checking Euclid. Annals of Mathematics and Artificial Intelligence p. 53 (2019). DOI 10.1007/s10472-018-9606-x. URL `https://hal.archives-ouvertes.fr/hal-01612807`
5. Bell, P.O.: Generalized theorems of desargues for n-dimensional projective space. Proceedings of the American Mathematical Society **6**(5), 675–681 (1955). URL `http://www.jstor.org/stable/2032913`
6. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer Science (2004)
7. Blanchette, J.C., Paulson, L.C.: Hammering away: A user's guide to Sledgehammer for Isabelle/HOL (2011)
8. Bouton, T., Oliveira, D.C.B.D., Déharbe, D., Fontaine, P.: verit: An open, trustable and efficient smt-solver. In: R.A. Schmidt (ed.) Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings, *Lecture Notes in Computer Science*, vol. 5663, pp. 151–156. Springer (2009). DOI 10.1007/978-3-642-02959-2_12. URL `https://doi.org/10.1007/978-3-642-02959-2_12`
9. Boutry, P., Braun, G., Narboux, J.: Formalization of the Arithmetization of Euclidean Plane Geometry and Applications. Journal of Symbolic Computation **90**, 149–168 (2019). DOI 10.1016/j.jsc.2018.04.007. URL `https://hal.inria.fr/hal-01483457`
10. Boutry, P., Gries, C., Narboux, J., Schreck, P.: Parallel postulates and continuity axioms: a mechanized study in intuitionistic logic using Coq. Journal of Automated Reasoning p. 68 (2017). DOI 10.1007/s10817-017-9422-8. URL `https://hal.inria.fr/hal-01178236`. Online first
11. Braun, D.: Approche combinatoire pour l'automatisation en coq des preuves formelles en géométrie d'incidence projective. Ph.D. thesis, Univ. Strasbourg (2019)
12. Braun, D., Magaud, N., Schreck, P.: Two cryptomorphic formalizations of projective incidence geometry. Ann. Math. Artif. Intell. **85**(2-4), 193–212 (2019). DOI 10.1007/s10472-018-9604-z. URL `https://doi.org/10.1007/s10472-018-9604-z`
13. Braun, D., Magaud, N., Schreck, P.: Two new ways to formally prove dandelin-gallucci's theorem. In: F. Chyzak, G. Labahn (eds.) ISSAC '21: International Symposium on Symbolic and Algebraic Computation, Virtual Event, Russia, July 18-23, 2021, pp. 59–66. ACM (2021). DOI 10.1145/3452143.3465550. URL `https://doi.org/10.1145/3452143.3465550`
14. Buekenhout, F. (ed.): Handbook of Incidence Geometry. North Holland (1995)

15. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications, pp. 11–27. Citeseer (2003)
16. Coq development team: The Coq Proof Assistant Reference Manual, Version 8.15. LogiCal Project (2022). URL http://coq.inria.fr
17. Coxeter, H.S.M.: Projective Geometry. Springer Science & Business Media (2003)
18. Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for Dependent Type Theory. pp. 423–453. Springer (2018)
19. Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., Barrett, C.W.: Smtcoq: A plug-in for integrating SMT solvers into coq. In: R. Majumdar, V. Kuncak (eds.) Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 10427, pp. 126–133. Springer (2017). DOI 10.1007/978-3-319-63390-9_7. URL https://doi.org/10.1007/978-3-319-63390-9_7
20. Fontaine, P., Marion, J., Merz, S., Nieto, L.P., Tiu, A.F.: Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In: H. Hermanns, J. Palsberg (eds.) Tools and Algorithms for the Construction and Analysis of Systems, TACAS, Proceedings, *LNCS*, vol. 3920, pp. 167–181. Springer (2006)
21. Horváth, Á.G.: The Theorem of Gallucci revisited. Journal of Geometry and Graphics **23**(2), 167–178 (2019)
22. Kodokostas, D.: Proving and Generalizing Desargues' Two-Triangle Theorem in 3-Dimensional Projective Space. Geometry **2014**, 276108 (2014). DOI 10.1155/2014/276108. URL https://doi.org/10.1155/2014/276108. Publisher: Hindawi Publishing Corporation
23. Kodokostas, D.: Proving and Generalizing Desargues' Two-Triangle Theorem in 3-Dimensional Projective Space. In: Geometry, vol. 2014. Hindawi (2014)
24. Kovács, L., Voronkov, A.: First-order Theorem Proving and Vampire. In: International Conference on Computer Aided Verification, pp. 1–35. Springer (2013)
25. Magaud, N.: Integrating an automated prover for projective geometry as a new tactic in the coq proof assistant. In: C. Keller, M. Fleury (eds.) Proceedings Seventh Workshop on Proof eXchange for Theorem Proving, Pittsburg, USA, 11th July 2021, *Electronic Proceedings in Theoretical Computer Science*, vol. 336, pp. 40–47. Open Publishing Association (2021). DOI 10.4204/EPTCS.336.4
26. Magaud, N., Narboux, J., Schreck, P.: Formalizing Projective Plane Geometry in Coq. In: Automated Deduction in Geometry (ADG'2008), LNAI 6301, pp. 141–162. Springer (2008). URL http://hal.inria.fr/inria-00305998
27. Magaud, N., Narboux, J., Schreck, P.: A Case Study in Formalizing Projective Geometry in Coq: Desargues Theorem. Computational Geometry: Theory and Applications **45**(8), 406–424 (2012). URL http://hal.inria.fr/inria-00432810/en
28. McLaughlin, S., Barrett, C., Ge, Y.: Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. In: Electronic Notes in Theoretical Computer Science, vol. 144, pp. 43–51. Elsevier (2006)
29. Michelucci, D., Schreck, P.: Incidence Constraints: a Combinatorial Approach. Int. Journal of Computational Geometry and Applications **16**(5-6), 443–460 (2006)
30. Moore, R.E.: Interval arithmetic and automatic error analysis in digital computing. Ph.D. thesis, Department of Computer Science, Stanford University (1962)
31. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 4963, pp. 337–340. Springer (2008). URL http://dx.doi.org/10.1007/978-3-540-78800-3_24
32. Narboux, J.: Mechanical Theorem Proving in Tarski's geometry. In: F.B. Eugenio Roanes Lozano (ed.) Automated Deduction in Geometry 2006, *LNCS*, vol. 4869, pp. 139–156. Francisco Botana, Springer, Pontevedra, Spain (2006). DOI 10.1007/978-3-540-77356-6. URL https://hal.inria.fr/inria-00118812
33. Oxley, J.G.: Matroid Theory, vol. 3. Oxford University Press, USA (2006)
34. Roanes-Macías, E., Roanes-Lozano, E.: A maple package for automatic theorem proving and discovery in 3d-geometry. In: F. Botana, T. Recio (eds.) Automated Deduction

in Geometry, 6th International Workshop, ADG 2006, Pontevedra, Spain, August 31-September 2, 2006. Revised Papers, *Lecture Notes in Computer Science*, vol. 4869, pp. 171–188. Springer (2006)
35. Schreck, P., Magaud, N., Braun, D.: Mechanization of incidence projective geometry in higher dimensions, a combinatorial approach. In: Automated Deduction in Geometry (ADG 2021) (2021). URL http://icube-publis.unistra.fr/6-SMB21
36. Schreck, P., Mathis, P.: Using jointly geometry and algebra to determine rc-constructibility. J. Symb. Comput. **90**, 124–148 (2019). DOI 10.1016/j.jsc.2018.04.006. URL https://doi.org/10.1016/j.jsc.2018.04.006
37. Sutcliffe, G.: The TPTP World - Infrastructure for Automated Reasoning. In: Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning, no. 6355 in Lecture Notes in Artificial Intelligence, pp. 1–12. Springer, Dakar, Senegal (2010)
38. Wen-Tsün, W.: Basic principles of mechanical theorem proving in elementary geometries. J. Autom. Reason. **2**(3), 221–252 (1986). DOI 10.1007/BF02328447. URL https://doi.org/10.1007/BF02328447

A Appendix

```
Load "preamble2D.v".

Lemma LABCD : forall A B C D ,
rk(A :: C ::  nil) = 2 -> rk(A :: B :: D ::  nil) = 3 ->
rk(C :: D ::  nil) = 2 -> rk(A :: C :: D ::  nil) = 2 ->
rk(A :: B :: C :: D ::  nil) = 3.
Proof.

intros A B C D
HACeq HABDeq HCDeq HACDeq .

assert(HABCDm2 : rk(A :: B :: C :: D :: nil) >= 2).
{
assert(HACmtmp : rk(A :: C :: nil) >= 2)
       by (solve_hyps_min HACeq HACm2).
assert(Hcomp : 2 <= 2) by (repeat constructor).
assert(Hincl : incl (A :: C :: nil) (A :: B :: C :: D :: nil))
       by (repeat clear_all_rk;my_in0).
apply (rule_5 (A :: C :: nil) (A :: B :: C :: D :: nil) 2 2 HACmtmp Hcomp Hincl).
}


assert(HABCDm3 : rk(A :: B :: C :: D :: nil) >= 3).
{
assert(HABDmtmp : rk(A :: B :: D :: nil) >= 3)
       by (solve_hyps_min HABDeq HABDm3).
assert(Hcomp : 3 <= 3)
       by (repeat constructor).
assert(Hincl : incl (A :: B :: D :: nil) (A :: B :: C :: D :: nil))
       by (repeat clear_all_rk;my_in0).
apply (
  rule_5 (A :: B :: D :: nil) (A :: B :: C :: D :: nil) 3 3 HABDmtmp Hcomp Hincl
      ).
}

assert(HABCDM : rk(A :: B :: C :: D ::  nil) <= 3)
       by (solve_hyps_max HABCDeq HABCDM3).
```

```
assert(HABCDm : rk(A :: B :: C :: D ::  nil) >= 1)
       by (solve_hyps_min HABCDeq HABCDm1).
intuition.
Qed.

Lemma LABC : forall A B C D ,
rk(A :: C ::  nil) = 2 -> rk(A :: B :: D ::  nil) = 3 ->
rk(C :: D ::  nil) = 2 -> rk(A :: C :: D ::  nil) = 2 ->
rk(A :: B :: C ::  nil) = 3.
Proof.

intros A B C D
HACeq HABDeq HCDeq HACDeq .

assert(HABCm2 : rk(A :: B :: C :: nil) >= 2).
{
assert(HACmtmp : rk(A :: C :: nil) >= 2)
      by (solve_hyps_min HACeq HACm2).
assert(Hcomp : 2 <= 2)
     by (repeat constructor).
assert(Hincl : incl (A :: C :: nil) (A :: B :: C :: nil))
     by (repeat clear_all_rk;my_in0).
apply (
  rule_5 (A :: C :: nil) (A :: B :: C :: nil) 2 2 HACmtmp Hcomp Hincl
      ).
}

assert(HABCm3 : rk(A :: B :: C :: nil) >= 3).
{
assert(HACDMtmp : rk(A :: C :: D :: nil) <= 2)
       by (solve_hyps_max HACDeq HACDM2).
assert(HABCDeq : rk(A :: B :: C :: D :: nil) = 3)
       by
       (apply LABCD with (A := A) (B := B) (C := C) (D := D) ; assumption).
assert(HABCDmtmp : rk(A :: B :: C :: D :: nil) >= 3)
       by (solve_hyps_min HABCDeq HABCDm3).
assert(HACmtmp : rk(A :: C :: nil) >= 2)
       by (solve_hyps_min HACeq HACm2).
assert(  Hincl :
         incl (A :: C :: nil)
         (list_inter (A :: B :: C :: nil) (A :: C :: D :: nil)))
       by (repeat clear_all_rk;my_in0).
assert(  HT1 :
         equivlist (A :: B :: C :: D :: nil)
         (A :: B :: C :: A :: C :: D :: nil))
       by (clear_all_rk;my_in0).
assert( HT2 :
        equivlist (A :: B :: C :: A :: C :: D :: nil)
        ((A :: B :: C :: nil) ++ (A :: C :: D :: nil))
     ) by (clear_all_rk;my_in0).
rewrite HT1 in HABCDmtmp;rewrite HT2 in HABCDmtmp.
apply (
  rule_2
     (A :: B :: C :: nil) (A :: C :: D :: nil) (A :: C :: nil)
     3 2 2 HABCDmtmp HACmtmp HACDMtmp Hincl
     ).
}
```

```
assert(HABCM : rk(A :: B :: C ::  nil) <= 3)
       by (solve_hyps_max HABCeq HABCM3).
assert(HABCm : rk(A :: B :: C ::  nil) >= 1)
       by (solve_hyps_min HABCeq HABCm1).
intuition.
Qed.
```