



HAL
open science

Proof Pearl: Formalizing Spreads and Packings of the Smallest Projective Space $PG(3,2)$ using the Coq Proof Assistant

Nicolas Magaud

► **To cite this version:**

Nicolas Magaud. Proof Pearl: Formalizing Spreads and Packings of the Smallest Projective Space $PG(3,2)$ using the Coq Proof Assistant. The International conference Interactive Theorem Proving (ITP) 2022, Jul 2022, Haïfa, Israel. 10.4230/LIPIcs.ITP.2022.25 . hal-03633318

HAL Id: hal-03633318

<https://hal.science/hal-03633318>

Submitted on 6 Apr 2022

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

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

1 Proof Pearl: Formalizing Spreads and Packings of 2 the Smallest Projective Space $PG(3,2)$ using the 3 Coq Proof Assistant

4 Nicolas Magaud ✉ 🏠 

5 Lab. ICube UMR 7357 CNRS Université de Strasbourg, France

6 — Abstract —

7 We formally implement the smallest three-dimensional projective space $PG(3,2)$ in the Coq proof
8 assistant. This projective space features 15 points and 35 lines, related by an incidence relation. We
9 define points and lines as two plain datatypes (one with 15 constructors for points, and one with 35
10 constructors for lines) and the incidence relation as a boolean function, instead of using the well-
11 known coordinate-based approach relying on $GF(2)^4$. We prove that this implementation actually
12 verifies all the usual properties of three-dimensional projective spaces. We then use an oracle to
13 compute some characteristic subsets of objects of $PG(3,2)$, namely spreads and packings. We formally
14 verify that these computed objects exactly correspond to the spreads and packings of $PG(3,2)$. For
15 spreads, this means identifying 56 specific sets of 5 lines among 360 360 ($= 15 \times 14 \times 13 \times 12 \times 11$)
16 possible ones. We then classify them, showing that the 56 spreads of $PG(3,2)$ are all isomorphic
17 whereas the 240 packings of $PG(3,2)$ can be classified into two distinct classes of 120 elements.
18 Proving these results requires partially automating the generation of some large specification files as
19 well as some even larger proof scripts. Overall, this work can be viewed as an example of a large-scale
20 combination of interactive and automated specifications and proofs. It is also a first step towards
21 formalizing projective spaces of higher dimension, e.g. $PG(4,2)$, or larger order, e.g. $PG(3,3)$.

22 **2012 ACM Subject Classification** [Replace ccsdesc macro with valid one](#)

23 **Keywords and phrases** Coq, projective geometry, finite models, spreads, packings, $PG(3,2)$

24 **Digital Object Identifier** 10.4230/LIPIcs...

25 **1** Introduction

26 Projective incidence geometry [9, 6] is one of the simplest description of geometry, where
27 only points and lines as well as their incidence properties are considered. In addition, in
28 such a setting, we assume that two coplanar lines always meet. There exist some finite and
29 infinite models of projective incidence geometry. Finite projective spaces are usually built
30 from finite (Galois) fields of cardinality n denoted $GF(n)$ via a homogeneous coordinate
31 system. Finite projective spaces arising from $GF(n)$ are denoted by $PG(d, n)$ where d is
32 the dimension of the space and n the order of the underlying field. Several finite models
33 are related to interesting mathematical puzzles and sometimes have practical and enjoyable
34 applications. This is the case for the finite projective plane $PG(2,7)$, which was used to
35 design the card game Dobble¹. In this game, players must identify a symbol which appears
36 on both their card and their opponent's card. As the card desk (almost exactly) implements
37 the projective plane $PG(2,7)$, given two cards (=lines), there always exists a symbol (=point)
38 which belongs to both cards. In a three-dimensional setting, the smallest projective space
39 $PG(3,2)$ can be used to find some solutions to an old combinatorial problem: *Kirkman's*
40 *schoolgirl problem* [7], which is stated as follows: *Fifteen young ladies in a school walk out*
41 *three abreast for seven days in succession: it is required to arrange them daily, so that no*

¹ <https://en.wikipedia.org/wiki/Dobble>

XX:2 Spreads and Packings of PG(3,2) in Coq

	# points	# lines	# points per line
$PG(2, 2)$	7	7	3
$PG(2, 3)$	13	13	4
$PG(2, 5)$	31	31	6
$PG(2, n)$	$n^2 + n + 1$	$n^2 + n + 1$	$n + 1$
$PG(3, 2)$	15	35	3
$PG(3, 3)$	40	130	4
$PG(3, 4)$	85	357	5
$PG(3, q)$	$(q^2 + 1)(q + 1)$	$(q^2 + q + 1)(q^2 + 1)$	$q + 1$

■ **Figure 1** Numbers of points, lines and points per line depending on the dimension and the order of projective planes and spaces

42 *two shall walk twice abreast.* As noted by Hirschfeld in [14, page 75], some solutions to
43 this problem correspond to some packings of $PG(3,2)$, which are one of the substructures of
44 $PG(3,2)$ that we study in this article.

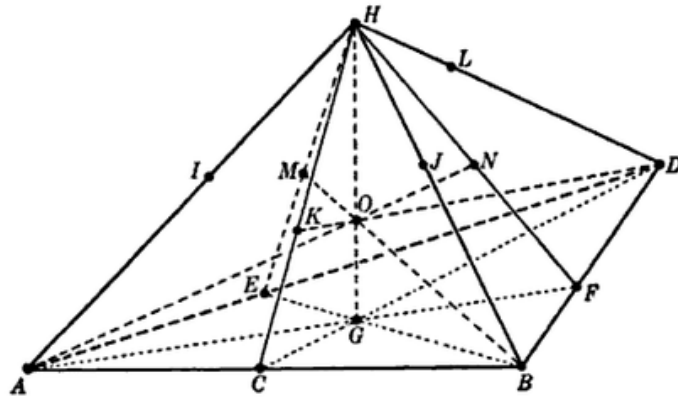
45 Finite projective spaces have been studied extensively from a mathematical point of
46 view (see e.g [14]). Recently [4], we started studying small finite projective planes/spaces
47 from a computer science perspective. We formalized usual projective planes such as $PG(2,2)$,
48 $PG(2,3)$ or $PG(2,5)$ as well as the smallest projective space $PG(3,2)$ using the Coq proof
49 assistant [8, 2]. We especially focused on proving that the synthetic axioms for projective
50 geometry hold in these models. In this paper, we follow up on experiments carried out
51 recently [16] and we formally describe some of the characteristic subsets of $PG(3,2)$, namely
52 spreads of lines and packings of spreads as well as their properties.

53 In a three-dimensional setting, the number of points and lines increase rapidly with the
54 order, as shown in Fig. 1. Thus we need to design extremely efficient proof techniques for
55 $PG(3,2)$ if we want our approach to be scalable to projective spaces of higher dimension
56 or larger order. The whole Coq formalization is available online and can be retrieved at:
57 <https://github.com/magaud/PG3q2>. Pointers to specific parts of the development are given
58 throughout the document. Visual representations of the smallest projective space $PG(3,2)$
59 can be retrieved from <https://demonstrations.wolfram.com/15PointProjectiveSpace/>.
60 For illustration purposes, we reproduce a figure taken from wikipedia³, which presents $PG(3,2)$
61 as a tetrahedron (see Fig. 2).

62 This paper is organized as follows. In Sect. 2, we show how to formally describe $PG(3,2)$
63 in Coq using plain inductive types. In Sect. 3, we define the notions of collineations, spreads
64 and packings in the setting of $PG(3,2)$. In Sect. 4, we characterize all the spreads of $PG(3,2)$
65 and show that they are all isomorphic. In Sect. 5, we characterize all the packings of $PG(3,2)$
66 and then classify them into two distinct classes. In Sect. 6, we present some proof engineering
67 techniques and suggest some additional optimizations to make the proof development smaller
68 and easier to compile. Finally, in Sect. 7, we draw some conclusions and outline how this
69 work can be extended to projective spaces of higher dimension or larger order.

² Be aware that compiling all the .v files of this development requires about 13 hours on a standard PC.

³ [https://en.wikipedia.org/wiki/PG\(3,2\)](https://en.wikipedia.org/wiki/PG(3,2))



■ **Figure 2** The smallest projective space $PG(3,2)$, represented as a tetrahedron.

70 2 Formal Description of the Projective Space $PG(3,2)$ in Coq

71 We first present an abstract interface (a Coq module) to describe what a projective space
 72 is. We also propose an implementation of $PG(3,2)$, relying on plain inductive datatypes for
 73 points and lines. We then show that all axioms of the projective space are verified by this
 74 implementation.

75 2.1 Specification of Projective Spaces

76 A three-dimensional projective space is parameterized by two types `Point` and `Line` as well
 77 as an incidence relation `incid_lp` (see Fig. 3 for the actual specification in Coq). The two
 78 types are equipped with an equality. Axiom `a1_exists` expresses that given two distinct
 79 points, one can always define a line going through these points. Axiom `uniqueness` states
 80 that given 2 points A and B and 2 lines l and m , if A and B are both incident to both l
 81 and m , then either $A = B$ or $l = m$. Axiom `a2`, also known as Pasch axiom, states that two
 82 coplanar lines always intersect. Axiom `a3_1` expresses that given a line, there are always
 83 three distinct points on it. Axiom `a3_2` expresses that there exist two lines which are not
 84 coplanar, thus making the dimension $n > 2$. Finally axiom `a3_3` states that, given 3 lines l_1 ,
 85 l_2 and l_3 , there always exists a fourth line m which intersects these 3 lines. This last axiom
 86 bounds the dimension so that $n \leq 3$.

87 2.2 Points, Lines and the Incidence Relation

88 We choose to use two simple inductive types to represent points and lines of $PG(3,2)$. Points
 89 are represented by an inductive datatype of 15 constructors without arguments. Lines are
 90 represented in the same way using 35 constructors.

```

91 Inductive Point :=
92 | P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9
93 | P10 | P11 | P12 | P13 | P14.
94
95 Inductive Line :=
96 | L0 | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 | L11 | L12
97 | L13 | L14 | L15 | L16 | L17 | L18 | L19 | L20 | L21 | L22 | L23
98 | L24 | L25 | L26 | L27 | L28 | L29 | L30 | L31 | L32 | L33 | L34.
99
100
```

XX:4 Spreads and Packings of PG(3,2) in Coq

```
Parameter Point, Line : Type.

Parameter eqP : Point -> Point -> bool.
Parameter eqL : Line -> Line -> bool.

Parameter incid_lp : Point -> Line -> bool.

Definition Intersect_In (l1 l2 :Line) (P:Point) :=
  incid_lp P l1 && incid_lp P l2.

Definition dist_3p (A B C :Point) : bool :=
  (negb (eqP A B)) && (negb (eqP A C)) && (negb (eqP B C)).

Definition dist_4p (A B C D:Point) : bool :=
  (negb (eqP A B)) && (negb (eqP A C)) && (negb (eqP A D))
  && (negb (eqP B C)) && (negb (eqP B D)) && (negb (eqP C D)).

Definition dist_3l (A B C :Line) : bool :=
  (negb (eqL A B)) && (negb (eqL A C)) && (negb (eqL B C)).

Axiom a1_exists : forall A B : Point,
  {l : Line | incid_lp A l && incid_lp B l}.

Axiom uniqueness : forall (A B :Point)(l1 l2:Line),
  incid_lp A l1 -> incid_lp B l1 ->
  incid_lp A l2 -> incid_lp B l2 -> A = B \ / l1 = l2.

Axiom a3_1 : forall l:Line,
  {A:Point & {B:Point & {C:Point | (dist_3p A B C) &&
  (incid_lp A l && incid_lp B l && incid_lp C l)}}}.

Axiom a2 : forall A B C D:Point, forall lAB lCD lAC lBD :Line,
  dist_4p A B C D ->
  incid_lp A lAB && incid_lp B lAB ->
  incid_lp C lCD && incid_lp D lCD ->
  incid_lp A lAC && incid_lp C lAC ->
  incid_lp B lBD && incid_lp D lBD ->
  (exists I:Point, incid_lp I lAB && incid_lp I lCD) ->
  exists J:Point, incid_lp J lAC && incid_lp J lBD.

Axiom a3_2 : exists l1:Line, exists l2:Line,
  forall p:Point, ~(incid_lp p l1 && incid_lp p l2).

Axiom a3_3 : forall l1 l2 l3:Line,
  dist_3l l1 l2 l3 ->
  exists l4 :Line, exists J1:Point,exists J2:Point,exists J3:Point,
  Intersect_In l1 l4 J1 &&
  Intersect_In l2 l4 J2 &&
  Intersect_In l3 l4 J3.
```

■ **Figure 3** Projective spaces of dimension 3: definitions and properties (pg3x_spec.v)

101 As there are three points per line, the incidence relation `incid_lp`⁴ can be represented in a
 102 compact way using the `match ... with` construct of Coq specification language.

```

103
104 Definition incid_lp (p:Point) (l:Line) : bool :=
105 match l with
106 | L0 => match p with P0 | P1 | P2 => true | _ => false end
107 | L1 => match p with P0 | P3 | P4 => true | _ => false end
108 | L2 => match p with P0 | P5 | P6 => true | _ => false end
109 | L3 => match p with P0 | P7 | P8 => true | _ => false end
110 | L4 => match p with P0 | P10 | P9 => true | _ => false end
111 | [...]
112 end.

```

114 In order to avoid writing too many specifications and proof scripts manually in Coq, we
 115 choose to build an external specification and proofs generator (a simple C program) which
 116 takes as input the number of points, the number of lines as well as the incidence relation
 117 as a plain file (`pg32.txt`⁵) which contains for each line of the projective space, the list of
 118 the points which are incident to it. Given these three elements, the system automatically
 119 builds the inductive datatypes for points and lines as well as the incidence relation. It
 120 also defines an artificial order on points and lines based on the index of the corresponding
 121 points and lines, i.e. $P_0 < P_1 < P_2 < \dots < P_{14}$. The specification generator also builds
 122 some auxiliary functions, which will be useful to prove existential statements of the form
 123 $\forall l_1 l_2 : \text{Line}, \text{exists } P : \text{Point}, \dots$

124 Using plain inductive datatypes may seem naive. An alternative approach to specify
 125 points and lines of $\text{PG}(3,2)$ could be to use finite types I_n of `ssreflect` and the mathematical
 126 components library [12, 17]. However the main drawback is that `ssreflect` is designed for
 127 formal reasoning rather than computing. Thus checking the incidence between a point and a
 128 line is a highly expensive operation, which prevents us from carrying out proofs efficiently.
 129 Using plain inductive types is much more efficient both to check incidence properties and
 130 to perform case analysis. The only drawback is that inductive datatypes and functions are
 131 huge to write, but this is not that important as we manage to generate these specifications
 132 automatically. Overall, our choice is to use the main features of `ssreflect`, especially the
 133 small-scale reflection pattern, but with our own datatypes.

134 2.3 Formal Proofs

135 Once the projective space $\text{PG}(3,2)$ is described, we check whether all the axioms for projective
 136 space geometry hold for this model. This requires proving all axioms of the module defined
 137 in https://github.com/magaud/Pg3q/blob/master/generic/pg3x_spec.v and presented
 138 in Figure 3. This is pretty straightforward and we try and make these proofs as generic and
 139 efficient as possible. We especially focus on writing general-purpose Ltac tactics, which can
 140 be easily reused for other models of projective space such as $\text{PG}(3,3)$.

141 We also rely on our specification generator to enhance producing witnesses for existential
 142 quantification. We use a form of skolemisation to write functions which compute the
 143 existential variable from the other arguments. For instance, to achieve the proof of lemma
 144 `a3_3`, we automatically build a (large) Coq function `f_a3_3` which, given three lines l_1, l_2
 145 and l_3 computes a line l_4 as well as its three intersection points with lines l_1, l_2 and l_3 .

⁴ https://github.com/magaud/Pg3q/blob/master/pg32/pg32_inductive.v

⁵ <https://github.com/magaud/Pg3q/blob/master/pg32/pg32.txt>

XX:6 Spreads and Packings of PG(3,2) in Coq

```
146 f_a3_3 : Line -> Line -> Line -> Line * (Point * Point * Point)
147
```

149 As a consequence, proving the statement `a3_3` boils down to feeding Coq with the correct
150 existential variables for the line and the three intersection points, obtained by applying the
151 function `f_a3_3` instead of trying all possible lines and points (there are 35 possible lines
152 and 15 possible points) for each of the $35^3 = 42\,875$ possible cases for parameters l_1 , l_2 and
153 l_3 of lemma `a3_3`.

154 Once that we checked that our implementation of PG(3,2) verifies all the axioms of
155 projective space, we shall study some specific subsets of points and lines, namely spreads
156 and packings.

157 3 Collineations, Spreads and Packings of PG(3,2)

158 Spreads are sets of lines of a projective space which can be defined when the number of points
159 per line divides the number of points. This is the case for all PG(n,q) whose dimension n is
160 odd. PG(3,2) features 15 points and has 3 points per line. Thus spreads exist in PG(3,2).

161 3.1 Collineations

162 A collineation is a couple of two functions $f_p : \text{Point} \rightarrow \text{Point}$ and $f_l : \text{Line} \rightarrow \text{Line}$ where
163 both f_p and f_l are bijections and respect the incidence relation.

```
164 Definition inj {A:Set} {B:Set} (f:A->B) : Prop :=
165   forall x y:A, f x = f y -> x = y.
166 Definition surj {A:Set} {B:Set} (f:A->B) : Prop :=
167   forall y:B, exists x:A, y=f(x).
168
169 Definition bij {A:Set} {B:Set} (f:A->B) : Prop := (inj f) /\ (surj f).
170
171 Definition is_collineation fp fl :=
172   ((bij fp) /\ ((bij fl) /\
173     (forall x l, incid_lp x l -> incid_lp (fp x) (fl l))))).
174
```

176 Collineations, which are automorphisms of PG(3,2) which respect the incidence relation,
177 shall be useful to establish that two given sets of lines are isomorphic, thus allowing us to
178 classify spreads and packings into equivalence classes.

179 3.2 Spreads

180 3.2.1 Definition and Properties

181 A spread of PG(3,q) is a set of $q^2 + 1$ lines which are pairwise disjoint and thus partition the
182 set of points. In PG(3,2), it corresponds to some sets of 5 lines. As recalled in [3, 7, 15], it is
183 well known that there is only one spread (up to isomorphism) in PG(3,2).

184 3.2.2 Generating all Spreads of PG(3,2)

185 Using our external specification and proofs generating program, we automatically compute
186 all sets of lines of PG(3,2) which are disjoint and cover all the points. As lines contain
187 exactly 3 points, they need to be sets of exactly 5 lines so that all the points of PG(3,2) are
188 accounted for. We generate 56 distinct spreads (modulo permutations of the order of the
189 lines involved). These spreads are defined in Coq as a list of 56 sets of 5 lines, as follows:

```

190
191 Definition S0 := [ L0; L19; L24; L28; L33 ].
192 Definition S1 := [ L0; L19; L26; L29; L32 ].
193 [...]
194 Definition spreads := [ S0 ; S1 ; S2 ; ... ; S54; S55 ].
195

```

We also generate automatically the collineations⁶ which allows to go from the spread S_i to the spread $S_{((i+1) \bmod 56)}$ of the list `spreads`⁷ as shown in the following example for the spreads `S0` and `S1`.

```

199
200 Definition fp0_1 (p:Point) := match p with P0 => P0 | P1 => P1 | P2 =>
201 P2 | P3 => P3 | P4 => P4 | P5 => P5 | P6 => P6 | P7 => P11 | P8 => P12
202 | P9 => P13 | P10 => P14 | P11 => P7 | P12 => P8 | P13 => P9 | P14 =>
203 P10 end.
204
205 Definition fl0_1 (l:Line) := match l with L0 => L0 | L1 => L1 | L2 =>
206 L2 | L3 => L5 | L4 => L6 | L5 => L3 | L6 => L4 | L7 => L7 | L8 => L9 |
207 L9 => L8 | L10 => L11 | L11 => L10 | L12 => L12 | L13 => L14 | L14 =>
208 L13 | L15 => L15 | L16 => L18 | L17 => L17 | L18 => L16 | L19 => L19 |
209 L20 => L20 | L21 => L21 | L22 => L22 | L23 => L25 | L24 => L26 | L25
210 => L23 | L26 => L24 | L27 => L30 | L28 => L29 | L29 => L28 | L30 =>
211 L27 | L31 => L34 | L32 => L33 | L33 => L32 | L34 => L31 end.
212

```

213 3.3 Packings

214 Once that we have built spreads as (disjoint) sets of lines covering all the points, we can
 215 define packings as sets of spreads covering all the lines of $\text{PG}(3,2)$.

216 3.4 Definition and Properties

217 A packing of $\text{PG}(3,q)$ is a set of $q^2 + q + 1$ spreads which are pairwise disjoint and thus
 218 partition the set of lines. In $\text{PG}(3,2)$, it corresponds to some sets of 7 spreads. There are 240
 219 packings, each of them being a list of 7 spreads. As recalled in [3, 7, 15], there are (up to
 220 isomorphism) exactly two distinct classes of packings in $\text{PG}(3,2)$.

221 3.5 Generating all Packings of $\text{PG}(3,2)$

222 We generate all sets of spreads which are disjoint and cover all the lines, and which thus are
 223 packings. As before, these sets of spreads must have 7 elements, as the number of spreads
 224 multiplied by the number of lines in each spread must be equal to the number of lines (35)
 225 of $\text{PG}(3,2)$. As expected (see Theorem 17.5.6 in [14]), we find 240 labelled packings.

```

226
227 Definition PA0 := [ S0; S9; S19; S24; S36; S46; S53 ].
228 Definition PA1 := [ S0; S9; S19; S28; S38; S40; S53 ].
229 Definition PA2 := [ S0; S9; S20; S27; S36; S46; S49 ].
230 [...]
231 Definition packings := [ PA0 ; PA1 ; PA2 ; ... ; PA238 ; PA239 ].
232

```

233 These packings belong to two distinct classes `class0` and `class1`⁸.

⁶ https://github.com/magaud/PG3q/blob/master/pg32/pg32_spreads_collineations.v

⁷ https://github.com/magaud/PG3q/blob/master/pg32/pg32_spreads_packings.v

⁸ https://github.com/magaud/PG3q/blob/master/pg32/pg32_spreads_packings.v

XX:8 Spreads and Packings of PG(3,2) in Coq

```
234
235 Definition class0 := [PA0; PA3; PA5; ... PA237; PA239 ].
236
237 Definition class1 := complement class0 packings.
```

238 As for spreads, we automatically generate the collineations⁹ which allow to go from one
239 packing of the list `class0` (resp. `class1`) to the next packing of `class0` (resp. `class1`).

240 Now that all externally-computed spreads and packings are defined in Coq, we shall
241 formally verify that they actually are spreads and packings of PG(3,2). We shall also check
242 that all the spreads are isomorphic and that the 240 packings can be classified into two
243 distinct classes of 120 elements.

4 Properties of the Spreads of PG(3,2)

4.1 Characterizing all Spreads of PG(3,2)

246 Spreads can be specified using the following definitions: the boolean function `is_partition`
247 computes whether the lists of points p, q, r, s and t partition the set of points and `is_spread5`¹⁰
248 used in conjunction with the function `all_points_of_line` computes whether the lines
249 $l1, l2, l3, l4$ and $l5$ actually constitutes a spread. The boolean function `forall_Point` is a
250 finite universal quantification: this means that `forall_Point (fun t => X t)` stands for
251 $X P0 \ \&\& \ X P1 \ \&\& \ X P2 \ \dots \ \&\& \ X P14$.

```
252
253 Definition is_partition (p q r s t : list Point) : bool :=
254   (forall_Point
255     (fun x => inb x p || inb x q || inb x r || inb x s || inb x t))
256   &&
257   (forall_Point
258     (fun x => negb (inb x p && inb x q && inb x r &&
259                   inb x s && inb x t))).
260
261 Definition is_spread5 (l1 l2 l3 l4 l5 : Line) : bool :=
262   disj_5 l1 l2 l3 l4 l5 &&
263   is_partition (all_points_of_line l1) (all_points_of_line l2)
264                 (all_points_of_line l3) (all_points_of_line l4)
265                 (all_points_of_line l5).
```

267 Once these definitions are set, we prove that the spreads of PG(3,2) are exactly the ones
268 automatically generated by our external program. On the one hand, we easily check that all
269 the computed spreads belonging to the list `spreads` actually verify the property `is_spread5`
270 of being a spread. On the other hand, we prove that all sets of 5 lines verifying the
271 property `is_spread5` belong to the proposed list `spreads`. Due to the size of the proofs
272 and in order to make them accepted by the Coq proof assistant, we need to decompose
273 this part of the proof into 35 specific cases. Each of them corresponds to one of the cases
274 $l1 = L0, l1 = L1, \dots, l1 = L34$. Eventually, using all these auxiliary lemmas, we prove the
275 following property:

```
276
277 Lemma is_spread_descr : forall l1 l2 l3 l4 l5 ,
278   leL l1 l2 && leL l2 l3 && leL l3 l4 && leL l4 l5 ->
279   (is_spread5 l1 l2 l3 l4 l5) <-> In [l1;l2;l3;l4;l5] spreads.
```

⁹ https://github.com/magaud/PG3q/blob/master/pg32/pg32_packings_collineations.v

¹⁰ https://github.com/magaud/PG3q/blob/master/pg32/pg32_spreads.v

281 In the previous statement, `leL` is a total order on the datatype `Line`, which expresses that
 282 $L_0 \leq L_1 \leq \dots \leq L_{34}$ and allows to only consider ordered spreads of lines.

283 4.2 Classifying all Spreads of PG(3,2)

284 The next step consists in proving that all 56 spreads of PG(3,2) are isomorphic. It can be
 285 expressed by stating that there exists a collineation, i.e. an automorphism of PG(3,2) which
 286 respects incidence, between any two spreads of PG(3,2).

```
287 Definition are_isomorphic (s1:list Line) (s2:list Line) : Prop :=
288   exists fp, exists fl, ((is_collineation fp fl) /\ (map fl s1 = s2)).
289
290
```

291 We show that the property `are_isomorphic`¹¹ is reflexive and transitive. Thanks to these
 292 results, we show that proving the equivalence can be achieved by simply proving that there
 293 exists a collineation (we actually build it) from the n -th element of the list to the $(n+1 \bmod 56)$ -
 294 th element of the list `spreads`. Using this transitivity property and the collineations computed
 295 by our external program, we fairly easily prove the following statement:

```
296 Lemma all_isomorphic_lemma : forall t1 t2 : list Line,
297   In t1 spreads -> In t2 spreads -> are_isomorphic t1 t2.
298
299
```

300 Overall, in this section, we formally proved in Coq that there are 56 labelled spreads in
 301 PG(3,2) and that there are all isomorphic.

302 5 Properties of Packings of PG(3,2)

303 In the following, we shall prove that there are 240 labelled packings in PG(3,2) and that
 304 they can be classified into two distinct classes.

305 5.1 Characterizing all Packings of PG(3,2)

306 A packing is defined using the predicate `is_packing`¹² as a set of 7 spreads (each being a
 307 list of lines) which are disjoint and form a partition of the set of lines.

```
308 Definition is_partition7 (p q r s t u v : list Line) : bool :=
309   (forall_Line
310     (fun x => inbL x p || inbL x q || inbL x r ||
311               inbL x s || inbL x t || inbL x u || inbL x v))
312   &&
313   (forall_Line
314     (fun x => negb (inbL x p && inbL x q && inbL x r && inbL x s &&
315                   inbL x t && inbL x u && inbL x v))).
316
317 Definition is_packing7 (s1 s2 s3 s4 s5 s6 s7:list Line) : bool :=
318   disj_7s s1 s2 s3 s4 s5 s6 s7 &&
319   is_partition7 s1 s2 s3 s4 s5 s6 s7.
320
321
```

322 Checking that all computed elements of the list `packings` are actual packings is straightfor-
 323 ward. Proving that all packings of PG(3,2) are in the list `packings` is a lot more challenging,

¹¹https://github.com/magaud/PG3q/blob/master/pg32/pg32_spreads_collineations.v

¹²https://github.com/magaud/PG3q/blob/master/pg32/pg32_packings.v

XX:10 Spreads and Packings of PG(3,2) in Coq

324 especially because of the number of cases to deal with. In order to make it tractable in Coq,
325 we prove several (56) lemmas of the form `statement_packings`¹³ s for some s .

```
326  
327 Definition statement_packings s :=  
328   forall s2 s3 s4 s5 s6 s7 : list Line ,  
329   In s2 spreads -> In s3 spreads -> In s4 spreads ->  
330   In s5 spreads -> In s6 spreads -> In s7 spreads ->  
331   ltS s s2 -> ltS s2 s3 -> ltS s3 s4 ->  
332   ltS s4 s5 -> ltS s5 s6 -> ltS s6 s7 ->  
333   is_packing7 s s2 s3 s4 s5 s6 s7 ->  
334   In [s;s2;s3;s4;s5;s6;s7] packings .  
335
```

336 In each of them, we fix the first spread (e.g. $s=S0$) and then verify that all packings containing
337 s as the first spread actually belong to the list `packings`.

```
338  
339 Lemma aux_S0 : statement_packings S0 .  
340 [...]   
341 Lemma aux_S55 : statement_packings S55 .  
342
```

343 Finally, we aggregate all 56 lemmas to obtain the following property:

```
344  
345 Lemma is_packing_descr : forall s1 s2 s3 s4 s5 s6 s7 : list Line ,  
346   ltS s1 s2 && ltS s2 s3 && ltS s3 s4 &&  
347   ltS s4 s5 && ltS s5 s6 && ltS s6 s7 ->  
348   In s1 spreads -> In s2 spreads -> In s3 spreads -> In s4 spreads ->  
349   In s5 spreads -> In s6 spreads -> In s7 spreads ->  
350   (is_packing7 s1 s2 s3 s4 s5 s6 s7) <->  
351   In [s1;s2;s3;s4;s5;s6;s7] packings .  
352
```

353 In the above statements, `ltS` is an order on spreads, which implements the lexicographic
354 order on spreads using the order on lines `ltL` as its basic order.

355 5.2 Classifying all Packings of PG(3,2)

356 In order to classify the packings of PG(3,2), we shall first prove that there are at most two
357 distinct classes of packings in PG(3,2). This is achieved using the collineations relating
358 packings provided in Sect. 3. Finally, considering two packings (one in each of the conjectured
359 classes), we show that no collineation can transform the first one into the second one.

360 5.2.1 There are at most 2 Classes of Packings in PG(3,2)

361 When considering packings, the relation `are_isomorphic`¹⁴ is a bit more complex as collin-
362 eations may transform a packing into a packing whose spreads are not sorted in increasing
363 order any more. Therefore we enforce that the images of the spreads computed using f_l must
364 be sorted with respect to the relation `ltL`.

```
365  
366 Definition are_isomorphic  
367   (p1:list (list Line)) (p2:list (list Line)) : Prop :=  
368   exists fp, exists fl,  
369   is_collineation fp fl /\   
370   forall s:(list Line), In s spreads -> In s p1 -> In  
371   (sort (map fl s)) spreads /\ In (sort (map fl s)) p2 .
```

¹³https://github.com/magaud/PG3q/blob/master/pg32/pg32_packings.v

¹⁴https://github.com/magaud/PG3q/blob/master/pg32/pg32_packings_collineations.v

373 Once again, we prove that the property `are_isomorphic` is reflexive and transitive. This
 374 allows to prove that all elements of a class are isomorphic by performing a circular permutation,
 375 simply proving that there exists a collineation (which was built explicitly by our external
 376 specification and proofs generating program) from the n -th element of the list `class0` (resp.
 377 `class1`) to the $(n + 1 \bmod 120)$ -th element of the list `class0` (resp. `class1`).

```
378 Lemma all_isomorphic_lemma0 : forall t1 t2 : (list (list Line)),
379   In t1 class0 -> In t2 class0 -> are_isomorphic t1 t2.
380
381 Lemma all_isomorphic_lemma1 : forall t1 t2 : (list (list Line)),
382   In t1 class1 -> In t2 class1 -> are_isomorphic t1 t2.
383
```

385 At this stage, we only proved that there are at most two classes of packings in $PG(3,2)$. The
 386 last step of the proof consists in proving that the two classes `class0` and `class1` are distinct.
 387 To do that, we choose two packings, e.g. `PA0` and `PA1`, one in each of the supposed classes.
 388 We then generate all collineations of $PG(3,2)$ and verify that none of these collineations
 389 allows to go from the packing `PA0` to the packing `PA1`.

390 5.2.2 Characterizing all collineations of $PG(3,2)$

391 So far we defined a collineation as a pair of two bijective functions f_p and f_l and a property
 392 that these functions respect the incidence relation. We shall see that a collineation (f_p, f_l)
 393 can be exactly characterized by simply defining the images of the four following points `P0`,
 394 `P1`, `P3` and `P7`. This relies on the property that there are only three points by line and that
 395 collineations are bijections which respect the incidence relation.

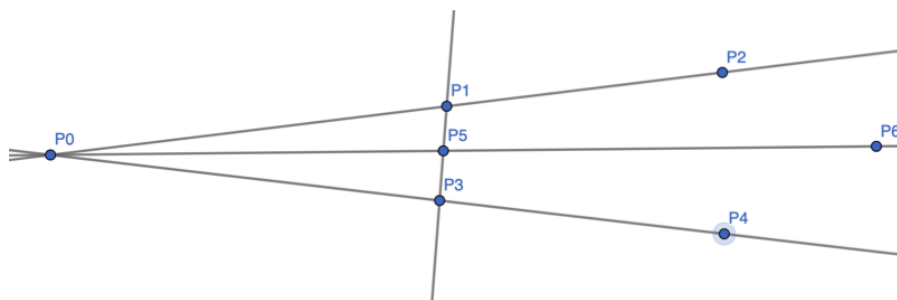
396 Let us start by choosing an image for the first point `P0`. Let us then choose an image for
 397 the second point `P1`. Then the image of the point `P2`, which is on line $L_0 = (P_0P_1)$ is imposed.
 398 It is the third point of the line generated by $f_p P_0$ and $f_p P_1$. Let us choose the image of
 399 the third point `P3`, which lies outside line L_0 . The images of points `P4`, `P5`, `P6` (see Fig. 4
 400 for an visual interpretation of the process) are imposed by the rules of the projective spaces
 401 and the collineation properties. We can then choose a fourth point `P7`. This point is outside
 402 the plane generated by `P0`, `P1` and `P3`. Once these four images $f_p P_0$, $f_p P_1$, $f_p P_3$ and $f_p P_7$
 403 are chosen, the images of all remaining points `P8`, `P9`, `P10`, `P11`, `P12`, `P13`, `P14` are imposed
 404 as being third points of some lines defined by the combination of images of the four initial
 405 points `P0`, `P1`, `P3` and `P7`. In addition, the images of all lines are fully determined as well.
 406 Indeed, the image of the line going through points A and B is the line going through points
 407 $f_p A$ and $f_p B$.

408 From a combinatorial point of view, we can choose the image of `P0` by f_p among 15
 409 points. The image of `P1` by f_p can be chosen among 14 points (all points except `P0`). The
 410 image of `P3` by f_p can be chosen among 12 points (all points except those on line L_0 , which
 411 contains points `P0`, `P1` and `P2`). Finally, the image of the fourth point `P7` by f_p can only
 412 be chosen outside of the images of points `P0`, `P1`, `P2`, `P3`, `P4`, `P5`, and `P6`, thus leaving only
 413 8 options available. Once the images of these four points are chosen, the collineation is
 414 fully characterized because both f_p and f_l must be bijective and that they must respect the
 415 incidence relation. This means that there are $15 \times 14 \times 12 \times 8 = 20\,160$ different ways to
 416 define a collineation of $PG(3,2)$.

417 Our external specification and proofs generating program takes care of computing all
 418 possible collineations of $PG(3,2)$. It generates some very large files: `pg32_automorphisms.v`¹⁵

¹⁵https://github.com/magaud/PG3q/blob/master/pg32/pg32_automorphisms.v

XX:12 Spreads and Packings of PG(3,2) in Coq



■ **Figure 4** Describing a collineation of PG(3,2) can be achieved by simply providing the images of points P0, P1, P3 and P7

419 (where the 20 160 collineations are enumerated), `pg32_automorphisms_inv.v` as well as files
420 `pg32_collineationsX.v` and `pg32_decompX.v`, with X ranging from 0 to 14. The list of all
421 collineations is splitted into smaller lists of size 96 in order to be able to handle the proofs in
422 Coq. Each subset of 96 collineations corresponds to specific collineations whose images of
423 points P0 and P1 are the same.

```
424  
425 Definition all_c0 := [  
426   (fp_0, fl_0); (fp_1, fl_1); ... ; (fp_94, fl_94); (fp_95, fl_95)].  
427 [...]  
428 Definition all_collineations :=  
429   all_c0 ++ all_c1 ++ all_c2 ++ ... ++all_c208 ++ all_c209.  
430
```

431 On the one hand, for each of these subsets, we can check that the given collineations
432 actually verify the property `is_collineation`. On the other hand, we verify that all
433 collineations which verify the following conditions: $f_p P0 = PX$ and $f_p P1 = PY$ actually
434 belong to the corresponding subsets of collineations, namely `all_cZ` where $Z = 14 \times X + Y$.
435 As an example, all collineations which respect the conditions $f_p P0 = P8$ and $f_p P1 = P2$
436 belong to the subset of collineations `all_c114`.

```
437  
438 Lemma is_collineations_descr_B_P8_P2 :  
439   forall fp fl, is_collineation fp fl -> fp P0 = P8 -> fp P1 = P2 ->  
440   In (fp,fl) all_c114.
```

442 Splitting the main statement characterizing all collineations into 210 smaller statements allows
443 to handle the proofs in Coq. Thankfully, all these 210 statements are almost automatically
444 generated and only some minor parts require to be fixed by hand. The last step consists in
445 agregating all these lemmas to obtain the following statement, which explicitly characterize
446 all collineations of PG(3,2).

```
447  
448 Lemma is_collineations_descr : forall fp fl,  
449   is_collineation fp fl <-> In (fp,fl) all_collineations.
```

5.2.3 There are exactly 2 Distinct Classes of Packings in PG(3,2)

452 Now that we have a list of all collineations of PG(3,2) at our disposal, we can traverse it to
453 verify that none of these collineations allow to transform a packing of the class `class0`, say

454 PA0 into a packing not in `class0`, say PA1. The proof simply consists in assuming, for each
 455 collineation that they allow to transform the packing PA0 into the packing PA1 and exhibit a
 456 contradiction. As we must check all collineations, the Coq file has more than 20 160 lines.

```
457
458 Lemma not_iso : ~are_isomorphic PA0 PA1.
459
```

460 The statement `not_iso`¹⁶ shows that the two classes of packings `class0` and `class1` are
 461 distinct. We can conclude that the 240 packings of PG(3,2) belong to two distinct classes,
 462 each of these classes containing exactly 120 elements.

463 6 Discussion

464 The Coq development is quite large. It contains more than 50 files. Thankfully, most for
 465 them are automatically generated. It consists in more than 317 345 lines of specifications and
 466 proofs, among them more than 290 000 are proof steps. Some files have about 20 000 lines,
 467 which makes them difficult (or at least very slow) to handle in an editor for Coq. Compiling
 468 the whole development requires about 13 hours (584 minutes on a Intel (R) Core(TM) i5-4460
 469 CPU @ 3.20GHz with 32GB of memory). Therefore it is important that all proofs are as
 470 concise as possible and the development must be well structured as changes in the structure
 471 may result in several hours of compilation before being able to resume interactive theorem
 472 proving. In the following, we present some proof engineering techniques which proved very
 473 useful in our development. We also propose some possible improvements to our work.

474 6.1 Proof Engineering

475 6.1.1 Using bool instead of Prop

476 As we work with finite types, equality and the other relations that we use are decidable.
 477 We can directly implement such relations as operations producing elements of the boolean
 478 datatype `bool`. This is more convenient than defining them as operations producing elements
 479 of type `Prop` together with a decidability property: $\forall x y, \{x = y\} + \{\neg x = y\}$. This practical
 480 approach is inspired by the `ssreflect` [11] and the mathematical components [17] libraries.
 481 In this setting, logical reasoning (eliminating conjunctions or disjunctions) is a bit more
 482 technical. However this makes most proofs much easier to complete by simply computing a
 483 boolean value and checking that it is equal to `true`.

484 6.1.2 Optimizing proofs

485 We design some optimization techniques for generating and checking proof terms. We focus
 486 on the current goal, applying some sort of locality principle which means that we try to
 487 prove a (sub-)goal the very first time we face it. This means sequences of tactics such as

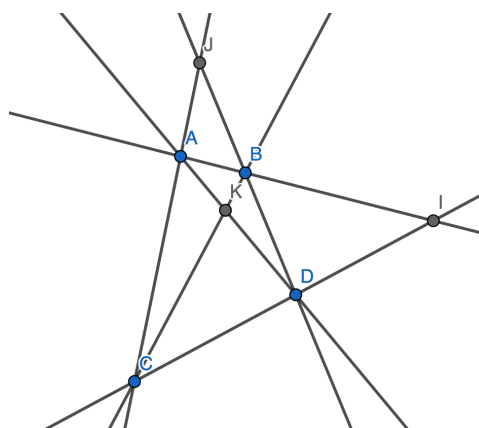
```
488
489 intros a; case a; intros H;
490   try (exact (degen_bool )_ H).
491 solve_goal.
492
```

493 must be replaced by more efficient sequences like

```
494
495 intros a; case a; intros H;
496   solve [(exact (degen_bool )_ H |solve_goal)].
497
```

¹⁶https://github.com/magaud/PG3q/blob/master/pg32/pg32_packings_two_distinct_classes.v

XX:14 Spreads and Packings of PG(3,2) in Coq



■ **Figure 5** An illustration of the new form of Pasch axiom used to deal with symmetries

498 In this simplified example, we try to apply the tactic `(exact (degen_bool)_ H)` for a
499 subgoal and then we switch to the next subgoal. Eventually we solve the remaining subgoals
500 using the `solve_goal` tactic. The idea here is to solve the goal the first time we encounter
501 it. It is achieved by having several possibilities of tactic applications to solve the goal (this
502 corresponds to the `solve [t1|t2|t3]` syntax). The order of the tactics `t1`, `t2` and `t3` can
503 be highly significant as well: we should always call the tactic which is the most successful
504 one on such subgoals first.

505 As we face a huge number of cases, we need to design extremely efficient prototype tactics
506 on some specific subgoals and apply them automatically to all the subgoals at stake. Fine
507 tuning the tactics rapidly is the key to making the proofs faster to complete.

508 Finally, Coq provides some sort of task parallelism in the form of the `par` tactical. It is
509 very useful to deal with all the sub-goals of a proof, once we figure out how to prove the
510 first one. The generic tactic proving the first goal, say `mytactic` can be easily applied to
511 all sub-goals in parallel (in some cases, we have $35 \times 35 = 1225$ or more goals to deal with) by
512 simply writing `par:mytactic`.

513 6.1.3 Without Loss of Generality

514 Most proofs are highly branching. For instance, performing case analysis on all three lines
515 to prove the lemma `a3_3` leads to $35^3 = 42875$ cases. In order to make the proof more
516 tractable, we propose a new tactic named `wlog`¹⁷, which implements the *without loss of*
517 *generality* principle, as it is described in [13]. This allows to reduce the number of cases to
518 solve explicitly. To use it, we build a virtual order on the points and lines, simply mapping
519 point P_i (resp. line L_i) to the value i of its index and then extend statements of the form
520 $\forall l_1, l_2 : \text{Line}, \dots$ to $\forall l_1, l_2 : \text{Line}, l_1 < l_2 \rightarrow \dots$

521 Surprisingly, using the without loss of generality tactic forces us to generalize our statement
522 for Pasch axiom to accommodate all cases, depending on the order in which we consider
523 points A , B , C , and D , as shown in Fig. 5. The usual conclusion of Pasch axiom:

```
524 (exists I:Point, incid_lp I lAB && incid_lp I lCD) ->  
525 {exists J:Point}, incid_lp J lAC && incid_lp J lBD.  
526
```

¹⁷<https://github.com/magaud/PG3q/blob/master/generic/wlog.v>

528 is transformed into a conjunction of two existential properties:

```
529 (exists I:Point, incid_lp I lAB && incid_lp I lCD) ->
530 (exists J:Point, (incid_lp J lAC && incid_lp J lBD)) /\
531 (exists K:Point, (incid_lp K lAD && incid_lp K lBC)).
532
533
```

534 The principles behind the tactic `wlog` were also extremely useful when dealing with
 535 spreads and packings, especially when checking inside Coq which sets of lines are actual
 536 spreads and which sets of spreads are actual packings.

537 6.2 Improvements

538 While carrying out such a proof development, one of the main difficulties is to decide what
 539 a *small* Coq proof is. Our first experiments crashed because we assumed Coq will handle
 540 very large specifications and proofs easily. Instead we needed to scale down our proofs and
 541 decompose them a lot to make sure they can be compiled. The current decomposition is
 542 probably too strong, but it has the advantage of being tractable by Coq.

543 Most definitions and properties used in this development are first order. So it would be
 544 interesting to implement the same formal description in a first-order prover such as Z3 [10].
 545 It can also be of interest to use first-order tools such as [1] provided in Coq.

546 From a specification point of view, as collineations can be simply characterized by the
 547 images of only four points, we shall study how to remove the bijection on lines from the
 548 definition of the collineation and reconstruct it from the bijection on points. This would
 549 make the proof development much smaller and reduces the number of objects we are handling
 550 simultaneously. Finally, most proofs are very similar to one another. In the near future,
 551 we shall study how symmetry arguments could help reduce the number of cases to handle.
 552 We shall also investigate how to carry out circular permutations of the set of points so that
 553 some proofs can be factorized by simply specifying the first point or line at stake and then
 554 rotating the statement to obtain the other cases.

555 7 Conclusion and Future Work

556 In this work, we show how to formalize in Coq the spreads and packings of $PG(3,2)$. Using
 557 an external specifications and proofs generating program, we build automatically all the
 558 spreads and packings, as well as all the collineations of $PG(3,2)$. We then easily verify that
 559 these generated sets of lines (resp. spreads) are actual spreads (resp. packings). We also
 560 successfully prove that they are the only ones. In addition, we classify the spreads and
 561 packings, showing that there is only one class for the 56 spreads and that the 240 packings
 562 are splitted into two classes of 120 elements. Showing that these two classes are distinct
 563 required generating and characterizing in Coq all the 20 160 collineations of $PG(3,2)$.

564 All the proofs carried out in this work are very large. A single case analysis on a point
 565 generates 15 cases, and a single case analysis on a line generates 35 cases. In order to let
 566 Coq deal correctly with all these proof scripts, we had to decompose our statements into
 567 several smaller lemmas, which could each be independently handled by Coq. During this
 568 study, we faced case analysis with a huge number of cases as well as debugging proof script
 569 with thousands of sub-goals. We propose some proof engineering techniques to make Coq
 570 process the files more easily e.g by directly providing witnesses or by pruning the proof tree
 571 by using a *without loss of generality* principle.

572 So far, we only address properties and transformations which remain in the same (pro-
 573 jective) space. We are currently working on generating specifications of projective spaces

574 automatically in order to easily have a formal description of two different projective spaces
575 and thus to be able to formally describe constructions as the Bruck-Bose construction which
576 allows to build translation planes from projective planes [5]. In parallel, we plan to formalize
577 the spreads and packings of PG(3,3) and their properties, as presented in [3]. This would
578 allow to check whether our specification and proofs techniques scale well when shifting for a
579 projective space with 15 points and 35 lines to a much bigger one with 40 points and 130
580 lines.

References

- 582 **1** Mickaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and
583 Benjamin Werner. Verifying SAT and SMT in Coq for a Fully Automated Decision Proced-
584 ure. In *International Workshop on Proof-Search in Axiomatic Theories and Type Theories*
585 (*PSATTT'11*), 2011.
- 586 **2** Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development,*
587 *Coq'Art : The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science,
588 An EATCS Series. Springer-Verlag, Berlin/Heidelberg, May 2004. 469 pages.
- 589 **3** Anton Betten. The packings of pg(3, 3). *Designs, Codes and Cryptography*, 79(3):583–595,
590 2016. doi:[10.1007/s10623-015-0074-6](https://doi.org/10.1007/s10623-015-0074-6).
- 591 **4** David Braun, Nicolas Magaud, and Pascal Schreck. Formalizing Some "Small" Finite Models
592 of Projective Geometry in Coq. In Jacques Fleuriot, Dongming Wang, and Jacques Calmet,
593 editors, *Proceedings of Artificial Intelligence and Symbolic Computation 2018 (AISC'2018)*,
594 number 11110 in LNAI, pages 54–69, Sept. 2018. URL: <https://hal.inria.fr/hal-01835493>.
- 595 **5** R.H Bruck and R.C Bose. The construction of translation planes from projective spaces. *Journal*
596 *of Algebra*, 1(1):85–102, 1964. doi:[https://doi.org/10.1016/0021-8693\(64\)90010-9](https://doi.org/10.1016/0021-8693(64)90010-9).
- 597 **6** Francis Buekenhout, editor. *Handbook of Incidence Geometry.* North Holland, 1995.
- 598 **7** F. N. Cole. Kirkman parades. *Bull. Amer. Math. Soc.*, 28(9):435–437, 12 1922. URL:
599 <https://projecteuclid.org:443/euclid.bams/1183485271>.
- 600 **8** Coq development team. *The Coq Proof Assistant Reference Manual, Version 8.13.2.* INRIA,
601 2021. URL: <http://coq.inria.fr>.
- 602 **9** Harold Scott Macdonald Coxeter. *Projective Geometry.* Springer Science & Business Media,
603 2003.
- 604 **10** Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In
605 *Proceedings of TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008. doi:
606 [10.1007/978-3-540-78800-3_24](https://doi.org/10.1007/978-3-540-78800-3_24).
- 607 **11** Georges Gonthier and Assia Mahboubi. A Small Scale Reflection Extension for the Coq system.
608 Technical Report RR-6455, INRIA, 2008. URL: <http://hal.inria.fr/inria-00258384/>.
- 609 **12** Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A small scale reflection extension
610 for the coq system. Research Report RR-6455, Inria, Saclay Ile de France, 2015. URL:
611 <https://hal.inria.fr/inria-00258384>.
- 612 **13** John Harrison. Without loss of generality. In Stefan Berghofer, Tobias Nipkow, Christian Urban,
613 and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International*
614 *Conference, TPHOLS 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674
615 of *LNCS*, pages 43–59. Springer, 2009. doi:[10.1007/978-3-642-03359-9_3](https://doi.org/10.1007/978-3-642-03359-9_3).
- 616 **14** J. W. P. Hirschfeld. *Finite projective spaces of three dimensions.* Oxford mathematical
617 monographs. Clarendon Press ; New York : Oxford University Press, Oxford, 1985.
- 618 **15** R.H. Jeurissen. Special sets of lines in PG(3, 2). *Linear Algebra and its Applications*, 226-228:617
619 – 638, 1995. Honoring J.J.Seidel. doi:[https://doi.org/10.1016/0024-3795\(95\)00200-B](https://doi.org/10.1016/0024-3795(95)00200-B).
- 620 **16** Nicolas Magaud. Spreads and packings of pg(3,2), formally! *Electronic Proceedings in*
621 *Theoretical Computer Science*, 352:107–115, Dec 2021. doi:[10.4204/eptcs.352.12](https://doi.org/10.4204/eptcs.352.12).
- 622 **17** Assia Mahboubi and Enrico Tassi. *Mathematical Components.* Draft, 2016. URL: <https://math-comp.github.io/mcb/>.
- 623