
**PROJET DE FIN D'ÉTUDES**

pour obtenir le diplôme de

l'**Université Galatasaray**
Spécialité : **Mathématiques**

# Formalisation du théorème de Desargues en Lean

Préparé par **Abdullah Uyu**
Résponsable : **Can Ozan Oğuz**

*6 mai 2024*

Ce document est composé avec `xelatex` dans Linux Biolinum pour le corps du texte, TeX Gyre Pagella Math pour les mathématiques et Iosvmata pour le code source, tous en 10pt.

Le code source de la formalisation, ce document et le schéma de preuve peuvent être trouvés dans https://github.com/oneofvalts.

# Remerciments

Tout d'abord, je tiens à remercier mon professeur, Can Ozan, pour l'attention et le soin qu'il a apportés à mon étude. Il a non seulement suivi les développements de mon étude, mais il a également appris à mes côtés pour mieux me guider. J'étais toujours enthousiaste à l'idée de partager mes résultats avec lui et de voir qu'il l'était aussi. Ses évaluations de mes rapports étaient très détaillées et il m'a toujours aidé à résoudre l'état brumeux de mon esprit induit par de longues périodes d'étude. Il le faisait en posant des questions cruciales qui aidaient également l'étude à reprendre son rythme dans les moments critiques.

Je dois également remercier les utilisateurs du canal Zulip Anand Patel, André Hernández-Espiet, Eric Wieser, Filippo A. E. Nuccio, Jireh Loreaux, Joseph Myers, Kevin Buzzard, Kyle Miller, Matt Diamond, Patrick Massot, Riccardo Brasca, Richard Copley, Ruben Van de Velde, Sabrina Jewson, Sophie Morel et Yaël Dillies pour leur aide dans le cadre de la programmation Lean.

Je voudrais également remercier ma famille qui m'a soutenu par ses vœux et ses prières. En particulier, l'intérêt de mon frère aîné Bayram à écouter le travail une fois terminé est très précieux pour moi.

Enfin, je voudrais remercier ma compagne, Esra, pour sa patience et pour avoir toujours été à mes côtés. Sans sa présence et son aide, je n'aurais jamais commencé à étudier.

*à Esra*

# Introduction

Lean est un assistant de preuve. La transcription d'une preuve en langage humain dans un assistant de preuve est appelée *formalisation*. Lean dispose d'une grande bibliothèque de preuves, y compris de nombreuses preuves du niveau de licence, appelée mathlib4. Dans cette bibliothèque, il y a d'importants théorèmes manquants, et le théorème de Desargues est l'un d'entre eux.

Les deux aspects principaux de ce projet sont l'apprentissage des rudiments de la théorie des géométries projectives, et de la programmation en Lean. Avec ces deux aspects, le but ultime est de formaliser le théorème de Desargues dans Lean.

# Table des matières

# Résultats réquis

## 1 Première rencontre

Pour motiver les géométries projectives, on commence par considérer les droites passant par l'origine dans le plan. On peut représenter la plupart de ces lignes par des points sur l'axe $y = 1$. La seule droite que l'on n'a pas réussi à représenter est l'axe $x$. On notera également que l'on peut bien sûr choisir n'importe quel axe pour la représentation, à l'exception de ceux qui passent par l'origine. Cette impossibilité dans les cas d'exception est clairement visible sur la figure 1.1. Si l'on choisit un tel axe, la droite que l'on ne parviendra pas à représenter sera la droite (passant par l'origine) qui est parallèle à cet axe.

Effectuons la même procédure pour l'espace. On peut représenter la plupart des droites passant par l'origine par des points sur le plan $z = 1$. Maintenant, les seules droites que nous ne pouvons pas représenter sont exactement les droites du plan que nous avons représenté précédemment. La remarque sur le choix de l'axe de représentation s'étend ici comme tout plan ne passant pas par l'origine peut être utilisé. De plus, le plan irreprésentable sera celui (qui passe par l'origine) qui est parallèle au plan

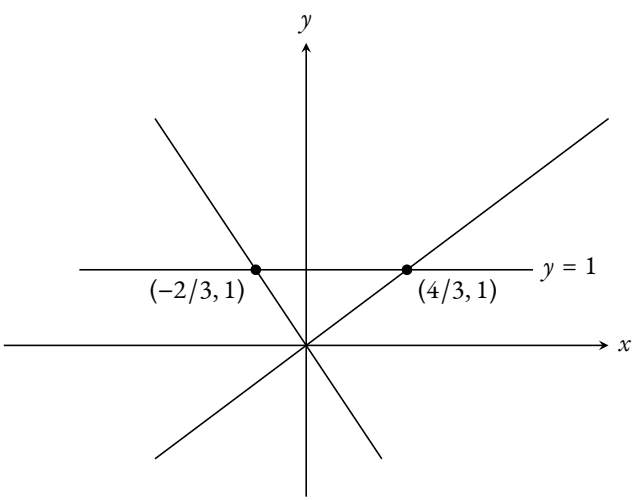


Figure 1.1 – La représentation des droites passant par l'origine dans le plan

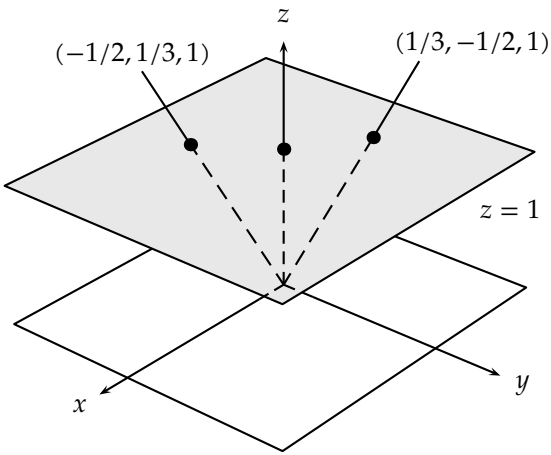


FIGURE 1.2 – La représentation des droites passant par l'origine dans l'espace

de représentation.

Ce processus s'appelle la *projectivisation d'un espace vectoriel*. Écrivons-le en langage d'algèbre linéaire.

## 2 Projectivisation d'un espace vectoriel

**Définition 1** ([FF00, p. 27]). Soit $V$ un espace vectoriel. Sur $V^\bullet := V \setminus \{0\}$, on définit une relation binaire comme suit : $x \sim y$ si et seulement si $x, y$ sont linéairement dépendants. Comme ceci est une relation d'équivalence, l'ensemble quotient $\mathbb{P}(V) := V^\bullet/\sim$ est bien défini. $\mathbb{P}(V)$ est appelée la *projectivisation de l'espace vectoriel* $V$.

Pour simplifier le langage, nous aurons également besoin de la définition de l'union disjointe.

**Définition 2.** Soit $A$ et $B$ deux ensembles. L'union $(A \times \{1\}) \cup (B \times \{0\})$, noté $A \dot\cup B$, est appelée *union disjointe* de $A$ et $B$.

La motivation ci-dessus peut donc être formulée comme suit :

$$\begin{aligned}\mathcal{P}(\mathbf{R}^3) &= \mathcal{P}(\mathbf{R}^2 \times \mathbf{R}) \\ &\cong \mathbf{R}^2 \dot\cup \mathcal{P}(\mathbf{R}^2) \\ &= \mathbf{R}^2 \dot\cup \mathcal{P}(\mathbf{R} \times \mathbf{R}) \\ &\cong \mathbf{R}^2 \dot\cup \mathbf{R} \dot\cup \mathcal{P}(\mathbf{R}).\end{aligned}$$

Cela résume ce que l'on fait mathématiquement lorsque l'on fait des dessins en perspective : On prend un plan ($\mathbf{R}^2$), on choisit un horizon ($\mathbf{R}$) et un point de fuite ($\mathcal{P}(\mathbf{R})$). La proposition suivante n'est qu'une généralisation de ce processus.

**Proposition 1** ([FF00, p. 28]). *Pour un $K$-espace vectoriel $V$, il existe une bijection naturelle*

$$s : \mathcal{P}(V \times K) \to V \dot\cup \mathcal{P}(V)$$

*induite par la fonction $t : (V \times K)^\bullet \to V \dot\cup \mathcal{P}(V)$ définie par $t(x, \xi) = \xi^{-1}x$ si $\xi \neq 0$ et $t(x, 0) = [x]$ pour $x \neq 0$, où $[x]$ désigne le point de $\mathcal{P}(V)$ représenté par $x$.*

**Définition 3** ([FF00, p. 26]). Une *géométrie projective* est un ensemble $G$ accompagné d'une relation ternaire $\ell \subseteq G \times G \times G$ telle que les axiomes suivants sont satisfaits :

($L_1$) $\ell(a, b, a)$ pour tout $a, b \in G$.

($L_2$) $\ell(a, p, q)$, $\ell(b, p, q)$ et $p \neq q \implies \ell(a, b, p)$.

($L_3$) $\ell(p, a, b)$ et $\ell(p, c, d) \implies \ell(q, a, c)$ et $\ell(q, b, d)$ pour un $q \in G$.

Les éléments de $G$ sont appelés les *points* de la géométrie. Et trois points $a, b, c$ sont dits *colinéaires* si $\ell(a, b, c)$.

Notons l'exemple le plus important pour une géométrie projective. Lors d'une discussion sur la chaîne de Zulip de Lean, Joseph Myers a conseillé d'énoncer le théorème sur cet exemple, mais pas sur la définition abstraite d'une géométrie projective. De plus, il est noté que la version du théorème en géométrie euclidienne, qui est probablement même connue de nombreux élèves du lycée, peut être déduite de l'énoncé du théorème sur cet important modèle de géométrie projective.

**Proposition 2** ([FF00, p. 27]). *Pour un espace vectoriel $V$, $\mathcal{P}(V)$ est une géométrie projective si pour tout élément $X, Y, Z \in \mathcal{P}(V)$ on définit : $\ell(X, Y, Z)$ si et seulement si $X, Y, Z$ ont des représentants $x, y, z$ linéairement dépendants.*

## 3 Isomorphismes

Les isomorphismes seront très utiles tout au long du voyage.

**Définition 4** ([FF00, p. 27]). Un *isomorphisme* de géométries projectives est une bijection $g : G_1 \to G_2$ satisfaisant $\ell_1(a,b,c)$ si et seulement si $\ell_2(ga, gb, gc)$. Si $G_1 = G_2$, alors on dit que $g$ est une *collinéation*.

Toutes les applications linéaires bijectives induisent une colinéation.

*Exemple* 1. Soit $T : \mathbf{R}^n \to \mathbf{R}^n$ une application linéaire bijective. L'application $g : \mathcal{P}(\mathbf{R}^n) \to \mathcal{P}(\mathbf{R}^n)$, $[x] \mapsto [T(x)]$ est une isomorphisme de géometries projectives.

L'application $g$ est bien définie : Soit $x, y \in \mathbf{R}^n$ tel que $[x] = [y]$, i.e. $x = ky$ pour un $k \in \mathbf{R}$. On a :

$$\begin{aligned} [T(x)] &= [T(ky)] \quad x \text{ définition} \\ &= [kT(y)] \quad T \text{ linéaire} \\ &= [T(y)] \quad \text{définition 1.} \end{aligned}$$

D'où, $g$ est bien définie.

Montrons que $g$ est injective. Soit $[x], [y] \in \mathcal{P}(\mathbf{R}^n)$ tel que $[T(x)] = [T(y)]$, i.e. $T(x) = kT(y)$ pour un $k \in \mathbf{R}$. Comme $T$ est linéaire, $T(x) = T(ky)$. De plus, $x = ky$ car $T$ est injective. Par la définition de la classe d'équivalence, on obtient $[x] = [y]$. D'où $g$ est injective. Pour la surjectivité, prenons $[x] \in \mathcal{P}(\mathbf{R}^n)$. Comme $T$ est bijective, $T^{-1}$ existe, et $[T^{-1}(x)] \in \mathcal{P}(\mathbf{R}^n)$. Ainsi $g([T^{-1}(x)]) = [T(T^{-1}(x))] = [x]$. D'où, $g$ est surjective. On a montré que $g$ est bijective.

Vérifions la condition d'isomorphisme. Soit $[x], [y], [z] \in \mathcal{P}(\mathbf{R}^n)$. Supposons que $\ell([x], [y], [z])$. On a des équivalences :

$$\begin{aligned} &\iff ax + by + cz = 0 \quad \text{proposition 2} \\ &\iff T(ax + by + cz) = 0 \quad T \text{ linéaire, injective} \\ &\iff aT(x) + bT(y) + cT(z) = 0 \quad T \text{ linéaire} \\ &\iff \ell([T(x)], [T(y)], [T(z)]) \quad \text{proposition 2} \end{aligned}$$

Mais les colinéations sont bien plus que des applications linéaires bijectives. En d'autres termes, il existe des collinéations qui ne sont pas induites par une application linéaire.

*Exemple* 2. L'application $g : \mathcal{P}(\mathbf{R}^2) \to \mathcal{P}(\mathbf{R}^2)$ définie par $[(x_1, x_2)] \mapsto [((x_1/x_2)^3, 1)]$ si $x_2$ est non-nul, et $[(x_1, 0)] \mapsto [(x_1, 0)]$ sinon, est une collineation qui n'est pas induite par une application linéaire.

L'application $g$ est bien définie. Soit $x_1 > 0, x_2 > 0, x'_1, x'_2$ des réels tel que $[(x_1, x_2)] = [(x'_1, x'_2)]$. Alors, $x'_1 = kx_1$ et $x'_2 = kx_2$ pour un $k$ non-nul. On a :

$$\begin{aligned}[((x'_1/x'_2)^3, 1)] &= [((kx_1/kx_2)^3, 1)] \\ &= [((x_1/x_2)^3, 1)]\end{aligned}$$

D'où, $g$ est bien définie.

Montrons que $g$ est injective. Soit $x_1, x_2, x'_1, x'_2$ des réels tel que

$$[((x_1/x_2)^3, 1)] = [((x'_1/x'_2)^3), 1].$$

On va montrer que $[(x_1, x_2)] = [(x'_1, x'_2)]$. Par la définition de la classe d'équivalence, on a

$$((x_1/x_2)^3, 1) = k((x_1/x'_2)^3, 1)$$

pour un $k$ non-nul. Alors, $k = 1$ et on a :

$$x_1/x_2 = x'_1/x'_2.$$

Si $x_1$ est nul, alors $x'_1$ l'est aussi, et donc on a :

$$\begin{aligned}[(0, x_2)] &= [0, \frac{x'_2}{x_2} x_2] \quad \text{classe d'équivalence définition} \\ &= [0, x_2]\end{aligned}$$

Sinon on a $x_1/x'_1 = x_2/x'_2$ et donc :

$$\begin{aligned}[(x_1, x_2)] &= [(\frac{x'_1 x_2}{x'_2}, \frac{x_1 x'_2}{x'_1})] \\ &= [(x'_1, x'_2)] \quad \text{classe d'équivalence définition}\end{aligned}$$

Enfin, on vérifie la reste de l'image. Soit $x_3$ un réel. Il est clair que $[((x_1/x_2)^3, 1)] \neq [(x_3, 0)]$. D'où, $g$ est injective. Pour la surjectivité, prenons des réels $x_1, x_2$. Si $x_2$ est nul, on a $g([(x_1, 0)]) = [(x_1, 0)]$. Sinon on a :

$$\begin{aligned}g([(x_1^{1/3}, x_2^{1/3})]) &= [((x_1^{1/3}/x_2^{1/3})^3, 1)] \quad g \text{ définition} \\ &= [(x_1/x_2, 1)] \\ &= [(x_1, x_2)] \quad \text{classe d'équivalence définition.}\end{aligned}$$

$g$ est automatiquement une collinéation car trois vecteurs quelconques sont toujours linéairement dépendants dans $\mathbf{R}^2$.

Montrons maintenant que $g$ n'est pas induite par une application linéaire. Supposons par l'absurde que $g$ est induite par l'application linéaire $T : \mathbf{R}^2 \to \mathbf{R}^2$. Alors on a $T(1,0) = (k_1, 0)$ et $T(0,1) = (0, k_2)$ pour $k_1, k_2$ non-nuls. Soit $x$ un réel. D'une part, on a $T(x,1) = k_x(x^3, 1)$ pour un $k_x$ non-nul, et d'autre part, on a $T(x,1) = xT(1,0) + T(0,1)$. Donc, on a :

$$k_x(x^3, 1) = x(k_1, 0) + (0, k_2)$$

Par conséquent, on obtient l'égalité des polynômes $k_2 x^3 = k_1 x$ qui entraîne $k_1 = k_2 = 0$. Ceci oblige $T = 0$ qui est absurde.

## 4 Sous-espaces

Cette partie comprend les structures qui sont importantes pour la première grande étape du projet.

**Définition 5.** Un *sous-espace* d'un géométrie projective $G$ est un sous-ensemble $E \subseteq G$ satisfaisant

$$a, b \in E \implies a \star b \subseteq E$$

**Définition 6.** Un *sous-géométrie projective* d'un géométrie projective $G$ est un sous-ensemble $G' \subseteq G$ tel que $G'$ avec la restriction $\ell' := \ell \cap (G' \times G' \times G')$ est aussi un géométrie projective.

On s'intéresse au résultat suivant.

**Proposition 3.** *Tout sous-espace est une sous-géométrie projective.*

On définit également des droites.

**Définition 7.** Une droite de $G$ est un sous-ensemble de la forme $\delta = a \star b$ où $a \neq b \in G$.

**Proposition 4.** *Toute ligne est un sous-espace.*

## Formalisation

Commençons par transcrire la définition de la géométrie projective (définition 3). Il faut d'abord représenter d'une certaine manière les points, c'est-à-dire les éléments d'une géométrie projective, et la relation de colinéarité. Les points de la géométrie projective seront abstraits, c'est-à-dire un *type*. La relation de colinéarité peut être représentée par une fonction qui prend trois points et renvoie vrai ou faux ; après tout, lorsque l'on dit qu'une relation $\ell$ est valable, ce que l'on fait, c'est prendre trois points et se demander s'ils sont colinéaires ou non.

```
class HasCollinear (P : Type) where
ell : P → P → P → Prop

export HasCollinear (ell)

variable {Point : Type} [HasCollinear Point]
```

Ici, la flèche entre les types `P` peut être considérée comme la flèche entre le domaine et le codomaine dans les définitions de fonctions dans le langage habituel des mathématiques, c'est-à-dire le symbole "→".

Comme le point reste abstrait, il a fallu dire à Lean que l'existence d'une interprétation de la colinéarité est assurée, c'est-à-dire que la colinéarité des points est quelque chose que l'on peut décider. Dans le code, cela est satisfait par la dernière ligne : `Point` est un type dont la colinéarité existe.

Maintenant, on peut énoncer les axiomes $L_1$, $L_2$ et $L_3$ dans la définition 3.

```
axiom l1 (a b : Point) : ell a b a
axiom l2 (a b p q : Point) : ell a p q → ell b p q → p ≠ q → ell a b p
axiom l3 (a b c d p: Point) : ell p a b → ell p c d →
                              ∃ q : Point , ell q a c ∧ ell q b d
```

Il y a quelques points à noter ici. Tout d'abord, les flèches utilisées ici sont différentes de celles que nous avons utilisées ci-dessus, même si elles sont représentées par le même caractère typographique : Il s'agit de flèches d'implication, pour lesquelles nous utilisons normalement le symbole "⟹". Deuxièmement, les conditions des axiomes qui contiennent des conjonctions logiques sont converties en implications sérielles. Expliquons cette équivalence logique. Supposons qu'on a une condition de la forme

$(p \wedge q) \implies r$. On a des équivalences suivantes :

$$\begin{aligned}
&\iff \neg(p \wedge q) \vee r \\
&\iff (\neg p \vee \neg q) \vee r \\
&\iff \neg p \vee (\neg q \vee r) \\
&\iff p \implies (\neg q \vee r) \\
&\iff p \implies (q \implies r)
\end{aligned}$$

Grâce à ces axiomes, nous pouvons prouver la proposition suivante.

**Proposition 5.** *Toute relation ternaire $\ell$ qui satisfait les deux axiomes $L_1$ et $L_2$ est symétrique.*

Dans la preuve, on va dériver la colinéarité de toutes les permutations possibles de trois points $a, b, c$ à partir de $\ell(a, b, c)$. Ici, on ne donnera que $\ell(a, c, b)$ et les cinq autres versions seront omises.

```
theorem acb
  (a b c : Point) :
    ell a b c → ell a c b := by
  intro acb_col -- Supposons que $\ell(a,b,c)$.
  obtain rfl | bc_neq := eq_or_ne b c -- Si $b = c$,
  exact acb_col -- le résultat est trivial,
  apply l2 a c b c -- sinon on applique $\text{L}_2$;
  exact acb_col -- première condition de $\text{L}_2$
  apply l1 c b -- deuxième condition de $\text{L}_2$
  exact bc_neq -- troisième condition de $\text{L}_2$.
```

Ce code est la transcription exacte de la phrase “Si $b = c$, le résultat est trivial, et sinon on applique $L_2$ à $\ell(a, b, c)$ et $\ell(c, b, c)$”.

## 1 Regrouper le point et les axiomes

Cette proposition la plus simple peut être considérée comme un premier théorème prouvé avec cette configuration. Des théorèmes plus ambitieux peuvent également être prouvés, mais décrire une géométrie projective en déclarant son point et axiomes les uns après les autres n’est pas la manière la plus élégante. Pour l’instant, ils n’ont qu’une relation sérielle dans le fichier. Ce qui est plus utile, c’est une structure qui contiendrait toutes les informations nécessaires à une géométrie projective. La structure suivante répond à ce besoin :

```
class ProjectiveGeometry
  (point : Type u) where
  ell : point → point → point → Prop
  l1  : ∀ a b, ell a b a
  l2  : ∀ a b p q, ell a p q → ell b p q → p ≠ q → ell a b p
  l3  : ∀ a b c d p, ell p a b → ell p c d →
                   ∃ q : point, ell q a c ∧ ell q b d
```

De cette manière, on a *regroupé* la relation de colinéarité et les axiomes (“bundling”), et le point est resté *dégroupé* (“unbundled”) parce qu’il s’agit d’un paramètre et que l’on veut que différents types de points correspondent à différentes géométries projectives. Pour mieux expliquer le contraste, on ne veut pas que des relations de colinéarité différentes correspondent à des géométries projectives différentes ; on pense à une relation de colinéarité canonique pour obtenir une géométrie projective. Ceci est expliqué en détail ici.

## 2 Instances

Il est maintenant possible de déclarer que la projectivisation d’un espace vectoriel est une géométrie projective.

```
instance : ProjectiveGeometry (ℙ K V) :=
  ⟨
  fun X Y Z => ¬ Independent ![X, Y, Z],
  sorry,
  sorry,
  sorry
  ⟩
```

Expliquons comment cela fonctionne : Tout d’abord, une relation de colinéarité sera fournie. Celle-ci est déjà établie mathématiquement dans la proposition 2. La fonction `Independent` provient de mathlib4. Elle prend trois points de la projectivisation et affirme que leurs représentants sont linéairement dépendants. Sur le plan technique, il prend une famille de points de projectivisation. Une famille est une fonction d’indices dans des points. Heureusement, mathlib4 dispose d’une syntaxe pour convertir une liste en une telle fonction : “`!`”. Si elle n’existait pas, on devrait l’écrire manuellement :

```
def indexer_generator
  (X Y Z : ℙ K V) :
    Fin 3 → ℙ K V :=
  fun i : Fin 3 => match i with
  | 0 => X
  | 1 => Y
  | 2 => Z
```

Ensuite, les axiomes seront vérifiés. Lean remplacera la relation de colinéarité dans les axiomes pour nous. Les “sorry” servent de substitut à des preuves qui n’ont pas encore été écrites.

Ainsi, nous pouvons continuer à étudier le côté purement synthétique et nos résultats s’appliqueront automatiquement à la projectivisation d’un espace vectoriel : Le côté concret. Pour les choses qui sont spécifiques aux projectivisations, nous pouvons bien sûr passer du côté concret et continuer à développer des propriétés pour elles.

## 3 Les yeux fermés

Une chose que j’ai souvent constatée tout au long de mon parcours, c’est que le livre est extrêmement prudent dans ce qu’il fait et ce qu’il dit. L’une de mes dernières expériences a été de réaliser que la fonction de colinéarité devrait en fait être dégroupé. Je m’en suis rendu compte une fois que j’ai essayé de montrer qu’un ensemble ayant deux fonctions de colinéarité différentes peut en fait être deux géométries projectives différentes, mais la fonction de colinéarité étant groupée, il était impossible de l’énoncer en premier lieu. En fait, nous devrions énoncer la classe de géométrie projective comme suit :

```
class ProjectiveGeometry
  (G : Type u)
  (ell : G → G → G → Prop) where
  l1  : ∀ a b , ell a b a
  l2  : ∀ a b p q , ell a p q → ell b p q → p ≠ q → ell a b p
  l3  : ∀ a b c d p, ell p a b → ell p c d → ∃ q, ell q a c ∧ ell q b d
```

Cela correspond en fait à la définition donnée dans le livre : Une géométrie projective est un ensemble $G$ avec une relation de colinéarité $\ell$... J’ai tiré deux leçons de cette expérience : La première est que lors de la formalisation d’un livre, il faut essayer de comprendre les intentions de l’auteur, en particulier lorsqu’une définition de base est donnée. La seconde est que, si

le livre est suffisamment prudent (formel, rigoureux...), il peut être possible de formaliser les yeux fermés. Par les yeux fermés, je veux dire formaliser comme un robot, dans un état pas particulièrement attentif.

Cela a été possible de manière très surprenante dans de nombreux cas. Je ne faisais qu'appliquer syntaxiquement les phrases des preuves les unes après les autres. Bien sûr, j'ai souvent pensé aux mécanismes (tactiques, structures ou autres caractéristiques du langage Lean) nécessaires à l'élaboration de la preuve donnée dans le livre, mais une fois cela fait, le travail consistait simplement à exécuter la preuve avec soin. C'est, encore une fois de manière surprenante, possible parce qu'il y a un vérificateur de preuves qui crie constamment la prochaine chose à montrer. Sans lui, il n'est pas possible d'avancer dans la preuve sans réfléchir.

## 4 Dépendances

J'ai été enthousiasmé de voir que les épreuves données dans le livre fonctionnaient bien. Cela passe par l'efficacité des preuves, la mention claire et explicite des propriétés, remarques, propositions ou théorèmes précédents montrés. Le livre a un sens de l'ordre très précis ; en tant qu'étudiant de licence, j'étais un peu timide à l'approche du livre, mais avec le temps et sa précision, je me suis continuellement familiarisé avec le livre.

Maintenant que j'ai parlé de la nature du livre, je devrais peut-être continuer à l'expliquer en détail. Le livre est un traité de géométrie projective avec morphismes. Il s'agit d'une géométrie projective axiomatique, et non d'une géométrie projective analytique. Et les morphismes sont des fonctions partielles, et non des fonctions à proprement parler. Le livre explique que c'est peut-être la raison pour laquelle il n'existe pas de traitement systématique de la géométrie projective avec morphismes. Le livre ajoute ensuite : "Nous avons l'intention de combler cette lacune. C'est en ce sens que la présente monographie peut être qualifiée de moderne".

Il semble que l'ambition d'utiliser l'algèbre se soit étendue à d'autres parties du livre, car celui-ci utilise également la théorie des treillis et la théorie des catégories pour diverses parties de la géométrie projective. Pour le meilleur ou pour le pire, j'ai maintenant une certaine compréhension de la façon dont différentes abstractions peuvent être utilisées et incorporées lors de la construction d'une théorie. En particulier dans la théorie des treillis, j'ai apprécié l'utilisation de l'opérateur $\vee$ dans la définition de l'hyperplan.

Mais cette généralité a été un obstacle pendant la plus grande partie du voyage. J'essayais constamment de réduire ces notations et ces définitions

parfois riches en formes pures. Par formes pures, j'entends les formes n'impliquant que les axiomes de la géométrie projective et les définitions que j'avais données jusqu'alors. Cette réduction était très difficile, d'une part parce que je devais apprendre une nouvelle théorie à partir de zéro et me familiariser avec ses notations, et d'autre part parce que je n'avais pas le temps de me familiariser avec la géométrie projective. Donnons la définition d'un hyperplan.

**Définition 8** ([FF00, p. 38]). Un *hyperplan* d'une géométrie projective $G$ est un sous-espace $H$ de $G$ qui est maximal parmi les sous-espaces stricts de $G$.

Immédiatement après la définition, nous avons la proposition énonçant les conditions équivalentes pour être un hyperplan.

**Proposition 6.** *Pour un sous-espace $H$ d'une géométrie projective $G$, les conditions suivantes sont équivalentes :*

- *$H$ est un hyperplan,*
- *$H$ est un sous-espace strict et pour tout point $a \in H$ on a $a \vee H = G$,*
- *il existe un point $a \notin H$ tel que $a \vee H = G$,*
- *$H \neq G$ et pour tout point $a \neq b$ on a $(a \star b) \cap H \neq \emptyset$.*

Ici, nous pouvons voir à la fois la forme pure, la forme théorique du treillis et le mélange des deux. L'objectif de cette proposition est en fait de les coller, de sorte que chacune d'entre elles puisse être utilisée librement par la suite. Mais du côté de la formalisation, c'est difficile. Il y a deux options pour les chemins à prendre. La première consiste à formaliser également les définitions et théorèmes de la théorie des treillis nécessaires, ou à les utiliser à partir de mathlib4 s'ils sont déjà formalisés, ce qui n'est pas toujours une opération parfaitement aisée. L'autre consiste à traduire à la main toutes les définitions et théorèmes de la théorie des treillis en formes pures et à les utiliser. Ici, nous avons heureusement une forme pure, mais lorsque le livre utilisera librement les théorèmes de treillis plus tard, nous serons coincés. Il s'agit donc d'un aspect dont il faut être conscient dans le cadre de la formalisation. Je pense pouvoir dire en toute confiance que cela a été la partie la plus sophistiquée, la plus longue et la plus fastidieuse de l'aventure.

## 5 Progression et régression

Maintenant que j'ai expliqué certains thèmes de la formalisation, il est temps de mentionner quelques phénomènes de haut niveau. Lorsque j'ai

commencé à étudier, ma stratégie consistait à acquérir une compréhension générale de la géométrie projective et de la preuve du théorème de Desargues, puis à trouver des solutions en cours de route. Cela n'a pas mal marché, mais il convient de noter que ce n'était pas du tout bien défini. Une compréhension générale est bien sûr nécessaire, et trouver des solutions en cours de route peut sembler pragmatique, mais lorsque je me suis finalement assis et que j'ai essayé de mettre en œuvre, j'ai immédiatement vu que le début est facile, mais que les conséquences seront dures. En effet, j'ai changé mes formulations un nombre incalculable de fois. Plus tard, j'ai réalisé que ce remaniement constant était en fait une partie importante du voyage.

Ensuite, lorsque j'arrive à un point stable, où je suis satisfait de mes formulations et où je n'obtiens aucune erreur du vérificateur de type, je me vois souvent un peu perdu, comme si je devais me demander quelle est la prochaine chose à énoncer ou à prouver, à nouveau. C'est la malédiction de l'approche progressive, et cela m'a incité à attaquer le problème de manière régressive.

L'approche régressive consiste à partir du dernier théorème à prouver et à déterminer ce qui est nécessaire pour le prouver en termes de définitions et de théorèmes, jusqu'à ce que toutes les lacunes soient comblées. Cette approche présente ses propres difficultés. Les dépendances augmentent de façon exponentielle et je me perds à les suivre. Pour faciliter le processus, j'ai décidé, à un moment donné, d'utiliser `Emacs/org-roam`.

Org-roam permet de travailler avec des noeuds. Dans chaque noeud, je conserve une définition ou un théorème. L'intérêt est de pouvoir les relier entre eux. J'obtiens ainsi une toile de définitions et de théorèmes. Ce qui est encore plus intéressant, c'est de pouvoir visualiser ce réseau, de manière totalement interactive. Cette méthode de travail a considérablement développé ma compréhension de la pratique des mathématiques. Je pense que tout mathématicien devrait au moins utiliser une fois cette méthode de construction d'un arbre de dépendance.

Je n'ai pas pu terminer complètement l'arbre de dépendance, mais je pense en avoir réalisé une assez grande partie. Analysons-le. Tout d'abord, notez qu'il s'agit d'une génération automatique de graphiques, c'est un fichier `svg` de haute qualité. Le théorème final se trouve en haut de la figure. Il s'agit de la projectivisation des espaces vectoriels, c'est-à-dire le théorème de Desargues habituel pour les élèves du secondaire. Immédiatement après, nous avons la forme générale du théorème.

La forme générale dépend de deux définitions majeures : La **géométrie arguésienne** et le **plongement de sous-espace**. Ceci est logique et probablement visible par la plupart des lecteurs. Mais en y regardant de

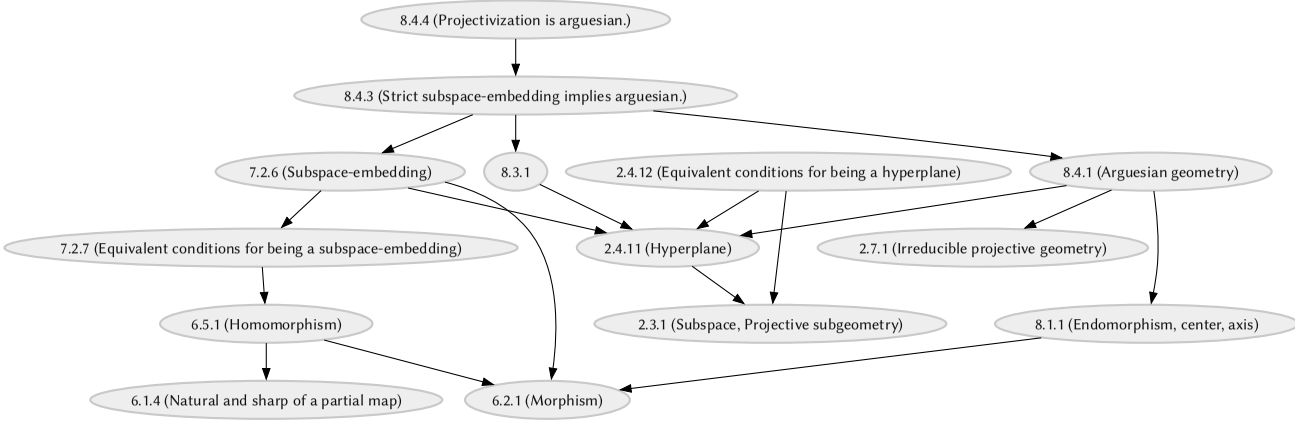


FIGURE 2.1 – L'arbre de dépendance de la preuve du théorème de Desargues.

plus près, on s'aperçoit que les définitions de l'**hyperplan** et des **morphismes** sont respectivement reliées par quatre et trois flèches. En d'autres termes, ils ont le plus grand nombre de liens de retour parmi les nœuds : Ce sont des nœuds centraux. Nous disons que de telles choses sont des concepts centraux. Mais dans le graphique, nous disposons d'une mesure quantifiée de l'importance d'un concept.

Ce fut très instructif, et je pense pouvoir dire que ce fut la partie la plus utile du voyage de formalisation.

Pour conclure mes propos sur les dépendances, j'ajouterai une petite découverte que j'ai faite lorsque je transcrivais le ($P_9$). Jusqu'à ce moment-là, toutes les transcriptions étaient presque mot à mot, phrase à phrase aux details de Lean près. Dans ($P_9$), à la dernière phrase, je lis "...et par ($P_4$) on conclut que...", mais il faut en fait appliquer ($P_4$) deux fois. Dans la preuve de ($P_8$), je lis "...en appliquant deux fois ($P_7$).", donc quand il faut appliquer un théorème deux fois, c'est indiqué dans le livre. Donc, soit on l'a oublié, soit il s'agit d'une omission incohérente.

## 6 Décidabilité

Lors de la définition de l'opérateur $\star$, il faut supposer que l'égalité de deux éléments de $G$ est décidable.

```
variable [DecidableEq G]
```

Voici la définition de l'opérateur $\star$.

```
def star
  [ProjectiveGeometry G ell]
  (a b : G) :
    Set G :=
  {c : G | if a = b then c = a else ell a b c}
```

Lorsque nous omettons l'égalité décidable de $G$, nous obtenons l'erreur :

```
-- failed to synthesize instance
--   Decidable (a = b)
```

## 7 Simplification

Comme la plus grande partie du travail consiste à réécrire l'expression de l'objectif, il est utile de disposer d'une automatisation pour cela. Lean a

cette syntaxe où vous mettez la balise `@[simp]` juste avant une définition ou un théorème qui énonce une égalité, la tactique `simp` applique automatiquement ces définitions et théorèmes balisés à l'objectif. La définition de l'opérateur $\star$ est assez appropriée pour cela. Voici la version balisée de la définition :

```
@[simp]
def star
  [ProjectiveGeometry G ell]
  (a b : G) :
    Set G :=
  {c : G | if a = b then c = a else ell a b c}
```

# Conclusion

Je me souviens avoir été averti par Yaël Dillies que la formalisation de la géométrie est étonnamment trompeuse. Je comprends maintenant que c'est peut-être le cas parce que la preuve de ($P_5$) fait 53 lignes et celle de ($P_9$) 77 lignes de code. Leurs versions informelles sont toutes deux de 5 lignes. Dans les deux, il y a de nombreux cas où je dois considérer à plusieurs reprises les cas où deux points sont distincts ou non. Et dans la preuve informelle, ces cas sont omis, pour de bon. Je pense que pour continuer ce projet, il faut absolument avoir une automatisation qui gère ces analyses de cas creux. Ce besoin d'automatisation apparaît également lorsque je prouve la symétrie de la relation de colinéarité. Les preuves ne sont que des applications de la permutation. Il est possible d'appliquer correctement les permutations ou d'effectuer une recherche par force brute. Dans le cas où nous avons $3! - 1$, il ne s'agit pas vraiment d'une recherche longue.

En dehors de ce goulot d'étranglement technique, les bases sont formalisées. Il est démontré que les droites sont des sous-espaces, et que les sous-espaces sont des sous-géométries projectives. C'est l'étape à laquelle se trouve le projet, à la fin de l'étude.

Pour poursuivre la formalisation, il faut absolument compléter le graphe de dépendance. Ensuite, tous les théorèmes requis devraient être énoncés et assumés. L'objectif est que le théorème de Desargues soit énoncé, mais non prouvé. Les lacunes peuvent être comblées à l'aide du livre, je ne pense pas que ce soit si difficile à ce stade.

# Références

[Avi+14] Jeremy Avigad et al. *Theorem proving in Lean 4*. Oct. 2014. url : https://lean-lang.org/theorem_proving_in_lean4/.

[FF00] C.A. Faure et A. Frölicher. *Modern Projective Geometry*. T. 521. Dordrecht : Kluwer Academic Publishers, 2000.