Université de Strasbourg

Ecole doctorale de Mathématiques, Sciences de l'Information et de l'Ingénieur

Thèse de doctorat

---

presentée publiquement le **23 septembre 2019** pour obtenir le grade de

Docteur de l'université de Strasbourg

par

David Braun

Approche combinatoire pour l'automatisation en Coq des preuves formelles en géométrie d'incidence projective

---

Membres du jury

Rapporteurs

M. Jacques Fleuriot	*Professeur, Université d'Edimbourg*
M. Dominique Michelucci	*Professeur, Université de Bourgogne*

Examinateurs

Mme. Dominique Bechmann	*Professeur, Université de Strasbourg*
M. Laurent Théry	*Chercheur, Inria Sophia Antipolis*

Encadrants

M. Nicolas Magaud	*Maître de conférences, Université de Strasbourg*
M. Pascal Schreck	*Professeur, Université de Strasbourg*

Résumé

Ce travail de thèse s'inscrit dans le domaine de la preuve assistée par ordinateur et se place d'un point de vue méthodologique. L'objectif premier des assistants de preuves est de vérifier qu'une preuve écrite à la main est correcte ; la question ici est de savoir comment à l'intérieur d'un tel système, il est possible d'aider un utilisateur à fabriquer une preuve formelle du résultat auquel il s'intéresse. Ces questions autour de la vérification de preuves, en particulier en certification du logiciel, et au delà de leur traçabilité et de leur lisibilité sont en effet devenues prégnantes avec l'importance qu'ont prise les algorithmes dans notre société.

Bien évidemment, répondre à la question de l'aide à la preuve dans toute sa généralité dépasse largement le cadre de cette thèse. C'est pourquoi nous focalisons nos travaux sur la preuve en mathématiques dans un cadre particulier qui est bien connu dans notre équipe : la géométrie et sa formalisation dans le système Coq. Dans ce domaine, nous mettons premièrement en évidence les niveaux auxquels on peut œuvrer à savoir le contexte scientifique à travers les méthodes de formalisation mais aussi le contexte méthodologique et technique au sein de l'assistant de preuve Coq. Dans un second temps, nous essayons de montrer comment nos méthodes et nos idées sont généralisables à d'autres disciplines.

Nous mettons ainsi en place dans nos travaux les premiers jalons pour une aide à la preuve efficace dans un contexte géométrique simple mais omniprésent. À travers une approche classique fondée sur la géométrie synthétique et une approche combinatoire complémentaire utilisant le concept de rang issu de la théorie des matroïdes, nous fournissons à l'utilisateur des principes généraux et des outils facilitant l'élaboration de preuves formelles. Dans ce sens, nous comparons les capacités d'automatisation de ces deux approches dans le contexte précis des géométries finies avant de finalement construire un prouveur automatique de configuration géométrique d'incidence.

Remerciements

Je souhaite débuter ce manuscrit en exprimant ma gratitude la plus sincère à toutes les personnes qui d'une manière ou d'une autre ont contribué à la réussite de ce projet doctoral.

Avant tout, je souhaiterai remercier mes encadrants Nicolas Magaud et Pascal Schreck pour leur support continu et leur supervision. Je remercie tout particulièrement Pascal pour son dévouement et son implication, ainsi que pour les innombrables discussions sur la recherche et parfois sur la vie, durant lesquels j'ai énormément appris. Quant à Nicolas, je lui exprime ma reconnaissance pour sa disponibilité, sa patience et ses conseils avisés qui m'ont guidé tout au long de cette thèse.

J'adresse la plus vive des gratitudes à l'ensemble de mon jury et tout particulièrement à mes rapporteurs pour avoir accepté d'évaluer mes travaux : Jacques Fleuriot, Dominique Michelucci, Dominique Bechmann ainsi que Laurent Théry.

Mes remerciements vont également à Julien, Pierre, Gabriel et Robin pour leur participation active à ce projet de recherche. Ce fut un plaisir de travailler à leur côté. Cette thèse ne serait pas la même sans leur contribution. Plus généralement, je voudrais remercier toute l'équipe IGG, et plus spécialement Dominique Bechmann, pour m'avoir accueilli dans ce cadre de travail agréable et épanouissant.

L'occasion m'est donnée ici de remercier également mes collègues doctorants actuels et passés en tentant de n'oublier personne : Alexandre, Cédric, Florian, Geoffrey, Katia, Mélinda, Nicolas, Noura, Pascal, Pierre et Sabah. Un grand merci à Cédric avec qui je partage mon bureau depuis trois ans pour sa joie de vivre et toutes les situations comiques que nous avons pu partager.

Finalement, je tiens à remercier mes parents, ma soeur et mon oncle pour leur générosité, leur discernement et leur soutien inconditionnel. Je leur dédie mes réussites passées et à venir.

Table des matières

Introduction

De la preuve mathématique à la preuve interactive en informatique

Selon le dictionnaire *Larousse*, une preuve est "un raisonnement qui permet, à partir d'axiomes, d'établir la validité d'une assertion". De tels raisonnement utilisent la logique mais incluent aussi habituellement des éléments du langage naturel en évitant autant que possible d'introduire des ambiguïtés. Ajoutons qu'une preuve joue deux rôles : premièrement elle convainc le lecteur que l'énoncé formulé est prouvable ; deuxièmement elle explique en quoi cet énoncé est correct.

En mathématiques, une preuve est irréfutable ; la correction de cette dernière peut être établie par n'importe qui. Une telle démonstration peut se réduire à une série de très petites étapes dont chacune est vérifiable simplement et indiscutablement. Ces étapes sont si petites qu'aucun mathématicien ne fait cela dans la pratique. Cependant, il est généralement admis que les preuves que nous trouvons dans les livres et les articles sont décomposables de cette manière. Il arrive parfois qu'une démonstration se révèle être incorrecte [1]. Dans ce cas, toutes les étapes n'ont pas été soigneusement détaillées et vérifiées.

Sans entrer dans les détails, les mathématiciens s'accordent sur la validité des étapes de preuves élémentaires et sur les méthodes qui permettent de les combiner pour former des preuves plus grandes. Le problème de la vérification manuelle d'une démonstration est ainsi lié à la traduction de cette dernière vers un formalisme plus précis. On pourrait penser que cela n'est pas très important puisque les mathématiciens décrivent déjà leur résultats avec des formules et un « jargon » technique relativement restreint. Cependant, il existe encore un écart considérable, en particulier au niveau des détails, entre les démonstrations manuscrites et les preuves qui sont comprises par un ordinateur.

Étant donné que le rôle premier d'une preuve est de garantir la véracité d'un énoncé mathématique, l'utilisation de systèmes informatiques pour concevoir et vérifier des démonstrations s'avère parfaitement adaptés. L'idée de mécaniser les preuves de théorèmes grâce aux capacités des ordinateurs est le fruit d'une longue histoire tant en mathématiques qu'en informatique. Le premier environnement logique reconnu permettant une mécanisation des preuves, appelé *Automath*, a été conçu en 1967 [NGdV94] par de Bruijn. Depuis *Automath*, une multitude d'assistants de preuves [Wie06] et de prouveurs automatiques [MMZ+01, NOT06] ont vu le jour. Grâce à ces outils, la machine et l'utilisateur humain peuvent travailler ensemble pour produire une preuve formelle sachant qu'il existe un large spectre des possibilités d'interactions. À un extrême, l'ordinateur ne sert que de vérificateur pour les preuves produites par l'humain, de l'autre nous disposons de prouveurs automatiques puissants. Malgré les succès du raisonnement automatique actuel [Jan11, Vor03], les résultats sont encore loin de reproduire le savoir-faire humain. En effet, l'espace de recherche pour les problèmes complexes explose très rapidement, ce qui les rend soit trop longs, soit inutiles dans de nombreuses tâches. Au vu des limitations de l'automatisation

1. Exemple célèbre de la démonstration du théorème de Fermat par Andrew Wiles qui incluait une faille dans la démonstration initiale.

pure, il semble aujourd'hui que la preuve interactive soit le meilleur moyen de démontrer la plupart des théorèmes non triviaux ou la correction des systèmes informatiques.

Cependant, cette approche nécessite beaucoup d'efforts de la part de l'utilisateur humain. Non seulement chaque preuve exige une bonne vision de sa structure, mais elle requiert aussi le traitement méticuleux d'une multitude de détails de bas niveau pouvant nuire à l'élaboration de la preuve principale. Pour ces raisons, il existe un intérêt croissant pour la démonstration semi-automatique de théorèmes, où un système intègre des techniques d'automatisation dans un environnement de démonstration interactif. De cette manière, nous conservons le meilleur des deux mondes : l'utilisateur peut guider manuellement la preuve lorsque l'automatisation des fragments simples de la théorie a échoué.

Le système auquel nous nous intéressons est l'assistant de preuve Coq [BC04, Coq02], qui possède une vaste bibliothèque de théories et des méthodes de preuves automatiques (tactiques et procédures de décision) combinant simplification et raisonnement classique. Cet outil fondé sur le langage puissant *Gallina* associe à la fois une logique d'ordre supérieur [2] et un langage de programmation fonctionnel fortement typé. Ce langage permet notamment : de définir des fonctions et des prédicats, d'énoncer des théories mathématiques et des spécifications pour les logiciels, de développer des preuves formelles interactives, de vérifier ces preuves grâce à un « noyau » de certification et enfin d'extraire des programmes certifiés dans plusieurs langages de programmations (Caml [LW93], Haskell [Jon03], ...). Cet assistant fournit aussi des algorithmes de décisions [Cré17, DM01] et un langage de tactique *Ltac* [Del00] offrant la possibilité de définir ses propres méthodes de construction de preuves. Ajoutons qu'il existe des connexions logiques entre cet assistant et différents prouveurs automatiques de théorèmes [AFG+11a, EMT+17, FMM+06]. Cet assistant de preuve sera notre terrain d'expérimentation pour étudier dans cette thèse la mise en place de la mécanisation plus ou moins complète des preuves en mathématiques.

Formalisation des mathématiques et certification logicielle

Ces vingt dernières années, plusieurs propriétés ont pu obtenir le statut de théorème grâce à la capacité des assistants de preuves à formaliser des démonstrations toujours plus complexes. Ces preuves sont si longues et si compliquées que, pour être reconnues par la communauté, leur formalisation dans des assistants de preuves semblait inévitable. Le résultat le plus célèbre que l'on peut citer est probablement le théorème des quatres couleurs [AH89]. Ce théorème indique que tout planisphère peut être colorié en utilisant uniquement quatre couleurs sans que deux régions adjacentes soient de la même couleur. Cette propriété initialement prouvée à l'aide d'un programme informatique par Appel et Haken énumérait toutes les combinaisons de cartes planes possibles. Elle n'a été officiellement acceptée par les mathématiciens lorsque Gonthier et Werner ont publié cette preuve [Gon05, Gon07] dans l'assistant de preuve Coq. Le théorème de Feit-Thompson, lui aussi démontré dans l'assistant de preuve Coq par Gonthier [GAA+13], affirme que chaque groupe fini d'ordre impair est résoluble. La preuve originale publiée par les auteurs du même nom était longue de plus de deux cent cinquante pages rendant sa compréhension globale particulièrement difficile. Le dernier résultat mathématique comparable est la démonstration de la conjecture de Kepler par Hales [Hal98]. De la même manière que pour le théorème des quatre couleurs, cette preuve initialement réalisée grâce à un outil informatique n'a été accueillie positivement qu'après sa formalisation [HAB+17] dans HOL-Light [Har96] et Isabelle [NPW02]. D'autres résultats mathématiques importants qui n'ont pas été sujets à ce type de controverse ont été démontrés avec des assistants à la preuve comme le théorème des nombres premiers [ADGR05, Har09] ou le théorème de la courbe de Jordan [Hal07].

2. La logique du second ordre est une logique formelle utilisant des variables qui réfèrent à des fonctions ou à des prédicats. Elle étend le calcul des prédicats.

À côté de toutes ces grandes démonstrations mathématiques, les assistants de preuve ont aussi été utilisés pour certifier des programmes informatiques. On assure ainsi que le programme respecte ses spécifications et ne contient pas de bugs. Dans certains secteurs critiques comme les banques, la médecine ou l'aéronautique, il est important de garantir qu'aucune perte financière ou humaine n'est susceptible de se produire. Le travail le plus connu se rattachant à cette certification logicielle est la preuve formelle du microkernel *seL4* [KEH+09]. Cette preuve construite et vérifiée en Isabelle comporte 200 000 lignes de script pour valider 7 500 lignes de langage C [KRB84]. Cette validation a permis de découvrir et corriger plus d'une centaine de bugs dans une des versions initiales de ce noyau. L'autre réalisation considérable que nous souhaitons mentionner est la programmation du compilateur CompCert C pour le langage C et sa preuve dans le langage Coq accomplie par l'équipe de Xavier Leroy [Ler06].

Formalisation de la géométrie

La géométrie, branche des mathématiques millénaire notamment avec les éléments d'Euclide [EDH02], joue un rôle tout aussi central dans l'évolution des systèmes de preuves. C'est un candidat idéal proposant un cadre propice à la mise en place de procédés automatiques de déductions qui a été étudié par de nombreuses équipes de recherche dont la nôtre. En effet, l'étude de ses fondements par des mathématiciens comme Euclide [EDH02], Hilbert [Hil60] ou Tarski [Tar59] a apporté des bases solides pour des développements systématiques ayant de nombreuses applications en mathématiques mais aussi en physique ou en robotique. Nous présentons ainsi dans la suite quelques formalisations connexes à nos travaux géométriques dans lesquelles la mécanisation des preuves a été évaluée. Ces formalisations sont réparties selon les deux approches classiques de la géométrie : synthétique ou analytique.

Avec une approche « synthétique » de la géométrie, un système d'axiomes est défini à partir de quelques objets géométriques et d'axiomes sur ces objets. L'idée de mécaniser les preuves géométriques fondées sur l'approche synthétique grâce aux outils informatiques commence au début des années 1950 avec les travaux de mathématiciens comme H. Gelernter, J.R Hanson et D. W. Loveland [GHL60]. Quand ces développements n'utilisent pas une axiomatique *ad hoc* spécialement conçue pour la tâche, ils s'appuient sur les formalisations axiomatiques modernes comme celles d'Hilbert [Hil60] ou de Tarski [SST13]. La première formalisation dans un assistant de preuve de la théorie hilbertienne, issue de l'ouvrage *Grundlagen der Geometrie*, est réalisée en Coq par Christophe Dehlinger, Jean-François Dufourd et Pascal Schreck [DDS00]. Dans un contexte intuitionniste, ils montrent que la décidabilité sur l'égalité et la colinéarité des points sont nécessaires pour vérifier les preuves de ce livre. Une autre formalisation, proposée par Laura Meikle et Jacques Fleuriot dans l'assistant de preuve Isabelle, est effectuée quelques années plus tard [MF03] avant d'être étoffée par Phil Scott [Sco08]. Ces trois articles montrent que les preuves d'Hilbert présentent des faiblesses, notamment en ce qui concerne le traitement des conditions de non-dégénérescences. Parallèlement, de nombreux travaux ont été réalisés sur la géométrie de Tarski et sur ses liens avec Hilbert [BBN16, BN12]. En effet, Pierre Boutry, Gabriel Braun et Julien Narboux ont longuement étudié la formalisation des fondements de la géométrie euclidienne en s'appuyant sur le système d'axiomes proposé par Tarski [Nar06b]. La librairie *GeoCoq*[3], développée principalement par leur soin, propose la formalisation complète de l'ouvrage *Metamathematische Methoden in der Geometrie* [SST13], d'une connexion entre les différentes formalisations axiomatiques modernes, d'une preuve d'équivalence entre plusieurs postulats des parallèles ainsi que d'un ensemble de tactiques pour simplifier et automatiser les démonstrations. À côté de cela, Frédérique Guilhot développe une important librairie Coq sur la

3. Une bibliothèque formalisant les fondements de la géométrie en Coq selon Tarski et Hilbert disponible à l'adresse : http ://geocoq.github.io/GeoCoq/

géométrie Euclidienne avec une perspective d'enseignement dans les lycées français [Gui04]. Par ailleurs, Jean Duprat propose une axiomatique pour la formalisation de la géométrie en utilisant uniquement la règle et le compas [Dup08].

La formalisation rigoureuse d'une théorie au sein d'un assistant de preuve fait apparaître rapidement et naturellement un choix qui est celui de travailler en logique intuitionniste ou en logique classique[4]. En considérant uniquement la notion de décidabilité, nous nous plaçons entre ces deux philosophies. Dans un environnement purement intuitionniste, nous interdisons le principe du tiers-exclu et les preuves existentielles sont toujours effectuées par la construction effective d'un témoin. C'est dans ce cadre qu'une formalisation avancée de la géométrie projective constructiviste décrite par von Plato [VP95] est développée en Coq par Gilles Kahn [Kah95]. Puis plus récemment, Guillermo Calderón implante cette même géométrie dans Agda [BDN09] en utilisant l'axiomatisation proposée par Mandelkern [Cal18]. Tous ces développements en géométrie synthétique intuitionniste ou non permettent de construire formellement des théories géométriques avec un haut niveau de confiance où les cas dégénérés doivent être minutieusement analysés [DDS00, MF03, Nar06b]. Notre travail s'inscrit dans la continuité de ces développements en géométrie synthétique avec la formalisation de la géométrie d'incidence projective.

En considérant une approche « analytique », on définit un espace $\mathbb{F}^n$ à partir d'un corps $\mathbb{F}$. C'est dans ce cadre que de nombreuses méthodes automatiques algébriques ont vu le jour pour mécaniser les démonstrations en géométrie. Tarski introduit tout d'abord dans le début des années 1930 une méthode d'élimination des quantificateurs [Tar98] pour prouver de nombreux théorèmes de géométrie élémentaire. Une percée majeure dans l'utilisation des méthodes algébriques provient de la méthode introduite par Wen-Tsün Wu [Wu78] en utilisant la notion d'ensemble caractéristique. L'implantation de la version simple de cette méthode a été réalisée en Coq [GNS11]. Une technique indépendante bien que fondée sur le même concept d'ensemble caractéristique réalisée par Bruno Buchberger [BW98], appelée base de Gröbner, a aussi été developpée en Coq [Pot08] et plus récemment en Isabelle [EP11]. Un autre algorithme notable que l'on peut citer est la méthode des aires présentée par Chou, Gao et Zhang [CGZ94] puis développée en Coq [JNQ12, Nar06a]. Finalement, l'algèbre de Grassmann-Cayley [LW03] formalisée en Coq par Fuchs et Laurent Théry [FT10] permet d'automatiser le raisonnement dans le cas de la géométrie projective.

Nous invitons le lecteur à se référer à l'article [NJF18] pour un état de l'art très complet et très détaillé sur la formalisation de la géométrie et sa mécanisation.

Dans cette thèse

Ce travail de thèse s'inscrit dans la problématique très générale de la preuve assistée par ordinateur en se plaçant du point de vue méthodologique. La question est de savoir comment à l'intérieur de ces systèmes, il est possible d'aider l'utilisateur à fabriquer la preuve formelle du résultat auquel il s'intéresse dans un cadre donné. La mise en place d'une aide à la preuve dans toute sa généralité est compliquée et hors de portée d'un seul travail de thèse. C'est pourquoi nous réduisons nos ambitions à la preuve en mathématiques à un thème spécifique bien caractérisé et bien connu dans notre équipe : la géométrie et sa formalisation dans l'assistant de preuve Coq. Tous les succès dans le domaine de la preuve interactive nous motivent à considérer et évaluer la mécanisation des preuves aussi complète que possible dans ce cadre. Pour cela, nous mettons premièrement en évidence les niveaux auxquels on peut agir, à savoir le contexte scientifique à travers les méthodes de formalisation, mais aussi le contexte méthodologique et technique au

4. La logique intuitionniste, parfois appelée logique constructive, diffère de la logique classique en reflétant plus étroitement la notion de preuve constructive.

sein de l'assistant de preuve Coq. Dans un second temps, nous essayons de montrer comment nos méthodes et nos idées sont généralisables. Pour simplifier, on peut dire que cette thèse résulte d'une combinaison de formalisation géométrique et de génie logiciel. Afin de mieux mettre en évidence les aspects méthodologiques, nous nous plaçons dans un cadre géométrique très simple, la géométrie d'incidence projective. À travers deux approches complémentaires de cette géométrie, nous fournissons à l'utilisateur des principes généraux et des outils facilitant l'élaboration de preuves formelles.

La géométrie d'incidence projective est choisie comme base de ce développement pour son axiomatique simple et ses propriétés méta-mathématiques bien connues, les plus importantes étant la consistance et la complétude [Cox03]. Le développement formel est effectué dans l'assistant de preuve Coq qui est fondé sur une théorie des types intuitionniste : *le calcul des constructions inductives* [CP88]. Le lecteur non familier avec cet outil parfaitement adapté pour l'étude théorique de propriétés mathématiques peut trouver dans [BC04, Chl13, Coq02, GM10, MT17] une introduction à cet assistant de preuve et ses extensions.

Tous les travaux de formalisation de la géométrie et d'automatisation des preuves décrits dans ce manuscrit sont intégrés dans une nouvelle librairie Coq appelée *ProjectiveGeometry* dont l'architecture globale est détaillée dans l'Annexe G. Dans sa globalité, notre développement, comportant plus de 600 000 lignes, 1 500 lemmes, 350 définitions, 250 tactiques dont 570 000 lignes ont été engendrées de manière automatique, peut être trouvé à l'adresse suivante :

https ://github.com/ProjectiveGeometry

Les contributions principales de cette thèse peuvent être résumées comme suit :

- Nous formalisons une bibliothèque sur la géométrie d'incidence projective en utilisant deux approches axiomatiques complémentaires : une approche classique fondée sur la géométrie synthétique et une approche combinatoire utilisant le concept de rang issu de la théorie des matroïdes ;
- Nous définissons la bi-interprétation entre les deux théories grâce à un dictionnaire permettant une traduction bidirectionnelle, puis nous prouvons l'équivalence entre les deux formalisations à la fois en 2D, $\geq$3D et 3D ;
- Nous mettons en évidence une méthodologie pour mécaniser les preuves en géométrie d'incidence projective en identifiant les étapes clés dans les démonstrations ;
- Nous analysons cette méthodologie de la preuve dans le cadre particulier des géométries finies définies par extension en énumérant l'intégralité des objets qui les composent. Dans ce contexte, nous présentons un ensemble de critères d'aide à la preuve généralisables pour l'automatisation des preuves de modèles et du théorème de Desargues tout en contrôlant l'explosion combinatoire ;
- Nous implantons un prouveur automatique par saturation utilisant les propriétés matroïdales issues de la théorie des matroïdes pour produire un certificat contenant la preuve d'un théorème en incidence pure ;
- Nous exposons dans un catalogue de démonstrations la génération automatique des preuves pour de grands théorèmes classiques de la géométrie incluant la propriété de Desargues, la configuration du conjugué harmonique et le théorème de Dandelin-Gallucci.

Les quatre premières contributions sont déjà synthétisées dans les articles de recherche suivants :

- David Braun, Nicolas Magaud et Pascal Schreck. Two cryptomorphic formalizations of projective incidence geometry. *Annals of Mathematics and Artificial Intelligence*, Springer, 2019, vol. 85, no 2-4, p. 193-212.
- David Braun, Nicolas Magaud et Pascal Schreck. Formalizing Some "Small" Finite Models of Projective Geometry in Coq. *Proceedings of the 13th International Conference on Artificial Intelligence and Symbolic Computation (AISC'2018)*, Springer, 2018, Suzhou, China, p. 54-69.
- David Braun, Nicolas Magaud et Pascal Schreck. An equivalence proof between rank theory and incidence projective geometry. *Automated Deduction in Geometry (ADG 2016)*, 2016, p. 62-77.

Les deux dernières contributions correspondent à des travaux qui n'ont pas encore été publiées. Le reste de cette thèse est divisé en quatre parties de la manière suivante :

La Partie I présente la formalisation de deux approches de la géométrie d'incidence projective : l'approche classique fondée sur la géométrie synthétique et l'approche combinatoire s'appuyant sur la théorie des matroïdes [GM12,Oxl06,Wel10]. Après avoir formalisé dans l'assistant de preuve Coq plusieurs systèmes d'axiomes, nous prouvons, en utilisant la traduction d'une théorie vers l'autre et vice versa, l'équivalence entre ces deux approches à la fois en 2D, $\geq$3D et 3D. Dans cette partie, nous étudions en simultané la méthodologie de la démonstration en géométrie d'incidence projective en examinant à la fois l'expressivité des énoncés géométriques et les étapes principales que l'on souhaite mécaniser lors des démonstrations. La mise en place de la preuve d'équivalence entre les deux théories montre que la fonction de rang issue de la théorie des matroïdes est plus adaptée pour une automatisation globale des démonstrations dans ce contexte.

La Partie II décrit une étude de cas où les géométries finies [Bat97, BW11, Dem12, RK70, Ros17] sont utilisées comme un banc d'essai pour l'automatisation des preuves en géométrie d'incidence projective. Ces géométries exprimées à partir d'un nombre fini de points et définies par extension en énumérant les objets qui les composent vont permettre de confronter les deux approches géométriques. Nous étudions la mise en place des procédés d'automatisation et l'efficacité des deux approches pour prouver que les espaces finis sont effectivement des modèles de la géométrie d'incidence projective respectant par ailleurs les propriétés de Desargues et Pappus. Pour rendre ces preuves réalisables, nous mettons en exergue un ensemble de critères généraux qui vont permettre de gérer au mieux l'explosion combinatoire du nombre de cas à traiter et d'optimiser les performances de l'assistant de preuve Coq dans le contexte des géométries finies. Par ailleurs, nous en profitons pour comparer les performances globales des prouveurs automatiques dans les mêmes démonstrations.

La Partie III est consacrée à l'implantation d'un prouveur automatique par saturation de configurations géométriques d'incidence. Cet outil d'aide à la preuve s'appuyant sur les propriétés matroïdales permet d'automatiser et vérifier automatiquement la plupart des raisonnements dans des preuves géométriques en incidence pure. En analysant et en optimisant les performances de ce prouveur, nous sommes en mesure de produire automatiquement la preuve détaillée de plusieurs théorèmes classiques de la géométrie d'incidences.

La Conclusion globale effectue une synthèse des travaux réalisés dans cette thèse et donne quelques axes de recherche à explorer.

Finalement la Partie IV rassemble une collection de documents annexes dont certains permettent une meilleure lecture (systèmes d'axiomes, théorème de Desargues et Pappus, ...).

Tous les tests de performances présentés dans cette thèse sont effectués sur une machine standard dont les spécificités sont les suivantes : Intel(R) Core(TM) i5-4460 CPU @3.20GHz avec 16Go de mémoire.

Quelques notations rencontrées au cours de ce manuscrit sont résumées :

- E. C. signifiant explosion combinatoire
- GD signifiant graphe de déductions
- T. E signifiant temps d'exécution
- M. signifiant mémoire
- Nb. signifiant nombre

De plus, nous utilisons une convention de nommage $AXYN$ pour nos systèmes d'axiomes. Les lettres correspondent à A pour axiome, X pour le numéro de l'axiome, Y peut prendre deux valeurs (P = projectif, R = rang) et N désigne la dimension.

Première partie

Deux approches cryptomorphiques pour la mécanisation des preuves en géométrie d'incidence projective

CHAPITRE I.1

Mécanisation de la démonstration en géométrie d'incidence projective

"Geometry, like arithmetic, requires for its logical development only a small number of simple, fundamental principles. These fundamental principles are called the axioms of geometry. Geometry is the most complete science."

David Hilbert (1862–1943)

Résumé

Dans ce chapitre, nous présentons le cadre géométrique qui nous permet d'étudier l'aide à la preuve en introduisant la géométrie d'incidence et la structure qui lui est associée (section 1). En précisant le concept de droites parallèles, nous clarifions la classe des théorèmes qu'il est possible de prouver en considérant uniquement la variante projective de la géométrie d'incidence. Nous formalisons à travers plusieurs systèmes d'axiomes la géométrie d'incidence projective à la fois en 2D, ≥3D et 3D. Parallèlement, nous décrivons notre implantation hiérarchique de tous ces systèmes d'axiomes au sein de l'assistant de preuve Coq (section 2). Dans la suite, la méthodologie de la démonstration en géométrie d'incidence projective est analysée grâce à une classification des éléments de preuves qui ont été identifiés comme des étapes indépendantes et nécessaires permettant l'automatisation des preuves (section 3). Puis en examinant l'expressivité de la géométrie synthétique, nous observons qu'il est commode d'introduire de nouvelles relations d'incidence en fonction de la dimension et de modifier en conséquence à chaque fois la mécanisation des preuves qui est déjà en soi une tâche bien complexe. Au final, une alternative intéressante évitant l'ajout de ces nouvelles relations et qui laisse présager une automatisation plus simple apparaît à travers le concept de rang (section 4).

Contenu

Dans ce chapitre, nous précisons le cadre géométrique qui quoique très simple nous permet dans la suite de ce mémoire d'étudier et d'évaluer nos idées sur l'aide à la preuve. De nombreux articles traitent de l'automatisation ou de la mécanisation des preuves dans différents contextes mathématiques (plus de 500 articles ayant le mot « *proof automation* », « *proof mechanization* » ou « *automated reasoning* » dans leurs mots-clés parmi les publications de la *ACM digital library* uniquement), dont voici une sélection : [Bee13, Bou97, BNSB14b, FT10, GNS11, GZND11, JNQ12, LW03, MF06, MNKJ16, Nar04, Nar06b, NJF18, RG95, SF12, Wu86]. Nombre de ces travaux, dont certains sont présentés dans le Chapitre Introduction, sont des procédures ou des méthodologies qui visent à simplifier ou à automatiser des preuves dans un environnement spécifique. Notre travail s'inscrit dans une démarche similaire puisque nous étudions les possibilités de mécanisation dans un contexte géométrique précis en nous aidant du langage *Gallina* [BC04, Coq02]. Cependant, notre objectif consiste aussi à contribuer plus largement au domaine de l'aide à la preuve formelle en dégageant des idées qui sortent de ce cadre spécifique.

En effet, comme nous l'avons indiqué dans le Chapitre Introduction, nous cherchons dans cette thèse à analyser et à mettre en place des mécanismes généralisables d'aide à la preuve en contexte géométrique. Le choix du terrain d'expérimentation, i.e. dans notre cas le choix du cadre géométrique, est donc crucial. D'un côté, il est important de considérer un cadre suffisamment riche pour pouvoir étudier des théorèmes pertinents. De l'autre côté, il ne doit pas multiplier les concepts pour éviter de masquer la généralité des idées que nous défendons en complexifiant le contexte. Notons, par ailleurs, que l'étude des fondements axiomatiques de la géométrie fait partie intégrante de l'ADN de notre petite équipe [BNSB14a, DDS00, MNS11, Nar06a, Nar06b] et ce travail a aussi pour objectif de contribuer à ce domaine.

Parmi les différentes approches axiomatiques, les éléments d'Euclide, repris plus formellement par Hilbert, proposaient déjà il y a plus de deux millénaires une structuration où la géométrie euclidienne usuelle est décrite progressivement en l'enrichissant de notions de plus en plus complexes. Ainsi, dans son ouvrage « *Grundlagen der Geometrie* » [Hil60], Hilbert propose une décomposition hiérarchique en cinq groupes d'axiomes qui, en considérant tous les modèles possibles, définissent des géométries de plus en plus complexes jusqu'à obtenir l'espace euclidien complet correspondant à $\mathbb{R}^3$.

(I) (*Incidence*) Ce premier groupe introduit les notions de point, droite, plan reliées entre elles par une relation d'incidence qui doit répondre aux axiomes de ce groupe ;

(II) (*Ordre*) Ce second groupe enrichit le cadre précédent en permettant de considérer une relation d'ordre entre les points sur une droite, tout en introduisant l'axiome de Pasch ;

(III) (*Congruence*) Les axiomes de ce troisième groupe complètent les deux premiers en définissant la notion de congruence et, par là, celle de déplacement. Hilbert caractérise ainsi les concepts de segments, de demi-droites, de demi-plans, d'ensembles convexes et d'angles ;

(IV) (*Parallèles*) Ce groupe introduit un unique axiome supplémentaire, celui des parallèles indiquant que : « par un point, il passe toujours une unique droite parallèle à une autre » ;

(V) (*Continuité*) Le dernier groupe ajoute le concept de continuité à travers les axiomes d'Archimède et d'intégrité linéaire.

En ne retenant que le groupe **(I)**, on obtient une géométrie appelée géométrie d'incidence très générale et dans laquelle la notion de coordonnée n'est pas pertinente. Cette géométrie purement synthétique est cependant décrite de manière un peu compliquée, quoique naturelle, par Hilbert qui introduit la notion de « plan ». Or la géométrie d'incidence, quelque soit la dimension visée, peut être formalisée en n'utilisant que les notions primitives de « point » et « droite ». On pourra ensuite considérer les plans en les définissant à l'aide de deux droites concourantes distinctes.

Le cadre obtenu pourtant très simple est suffisamment riche pour étudier de nombreux théorèmes fondamentaux dont certains sont présentés au fil des chapitres.

1 Géométrie d'incidence

Cette section présente une formalisation ainsi qu'une analyse des fondements axiomatiques de cette géométrie.

1.1 Structure d'incidence

Précisons qu'une géométrie d'incidence est un modèle des axiomes provenant du groupe **(I)** qui sont énoncés formellement un peu plus loin. De manière sémantique, une telle géométrie est définie par un triplet (Ω, Δ, Φ) appelé structure d'incidence où Ω et Δ sont deux ensembles d'éléments disjoints liés par la relation binaire d'incidence Φ. Les éléments qui sont en relation d'incidence constituent un sous-ensemble $\Phi \subseteq \Omega \times \Delta$. En utilisant la terminologie géométrique usuel : les éléments de Ω sont les points, ceux de Δ sont les droites. Si $A \in \Omega$ et $\delta \in \Delta$ sont tels que $A \in \delta$, on dira, indifféremment, que le point A appartient à la droite δ, que δ passe par A ou que A et δ sont incidents. Cela permet de redéfinir certaines notions de la géométrie affine plane comme le parallélisme : deux droites sont parallèles si elles sont égales ou si aucun point n'est sur les deux simultanément.

Si l'abstraction et la caractérisation de cette structure d'incidence avec ses variantes projectives proviennent de la géométrie, la notion d'incidence trouve des applications dans d'autres domaines des mathématiques comme la combinatoire et la théorie des graphes mais aussi dans la vie de tous les jours avec les plans de métro et même des jeux pour enfants comme Dooble [1].

1.2 Description informelle

Tous les exemples que nous venons d'évoquer partagent un ensemble de règles simples. Ces règles ont été au cours du temps formalisées en des systèmes axiomatiques qui sont détaillés dans la suite. La notation $\in$ provenant de la théorie ensembliste désigne dans la suite l'appartenance d'un point à une droite.

(I1) Soient deux point distincts A et B du plan, il existe une unique droite l dans le plan tel que $A \in l$ et $B \in l$.

(I2) Soit une droite l du plan, il existe deux points distincts A et B du plan tel que $A \in l$ et $B \in l$.

(I3) Il existe trois points distincts non alignés dans le plan.

TABLE I.1.1 – Description informelle de la géométrie d'incidence.

La formalisation de ces règles est présentée dans [Hil60] en s'inspirant des travaux d'Hilbert. Nous notons l_{AB} l'unique droite dans le plan passant par les points distincts A et B et $\cap$ l'intersection entre deux droites. À partir des règles de la description I.1.1, nous pouvons déduire deux résultats directs qui sont utilisés dans les preuves le plus souvent sans aucune mention.

1. Dooble est un jeu de société dans lequel les joueurs doivent associer des dessins communs entre deux cartes. Un tel paquet de cartes est conçu à partir des droites de $P^2(K)$ où K est un corps fini.

Lemme I.1.1. *Soient deux droites distinctes l et m, si $X \in (l \cap m)$, alors $l \cap m = \{X\}$.*

Démonstration. Supposons qu'il existe un point A tel que $A \neq X$ et que $A \in (l \cap m)$. D'après l'axiome **(I1)**, $l = l_{AX} = m$. Nous obtenons une contradiction avec l'hypothèse que les deux droites sont distinctes, un tel point A ne peut donc pas exister. Autrement dit, l'intersection entre nos deux droites est unique.

□

Lemme I.1.2. *Soient trois points non colinéaires A, B et C. Alors A, B et C sont distincts deux à deux.*

Démonstration. Supposons dans un premier temps que les trois points soient égaux $A = B = C$. Grâce à l'axiome **(I3)**, il existe un point $Q \neq A$. En utilisant l'axiome **(I1)**, il existe une droite l contenant A et Q. Nous déduisons que A, B, $C \in l$ et donc que A, B et C sont colinéaires ce qui est contradictoire avec l'hypothèse.
Dans un second temps, si les trois points forment cette fois-ci une droite, les points A, B et C sont colinéaires, ce qui est impossible.
Étant donné que les trois points ne peuvent pas être égaux ou alignés, ils déterminent nécessairement un plan où les points sont nécessairement distincts deux à deux. □

1.3 Variante projective

Ces deux lemmes illustrent le raisonnement mathématique que l'on peut effectuer dans un cadre géométrique en incidence pure. Pour compléter cet ensemble de règles, nous définissons le concept de parallélisme introduit dans le groupe **(IV)** de la décomposition d'Hilbert. Les deux principales généralisations des espaces munis d'une structure dite d'incidence sont les espaces affines et les espaces projectifs. Dans un espace affine, le parallélisme est défini comme l'existence d'une unique droite parallèle à une autre passant par un point donné. Dans le cas projectif, on ajoute des points à l'infini pour chaque direction de manière à ce que deux droites strictement parallèles se coupent en un tel point. La géométrie projective est une complétion de la géométrie affine sans perte de généralité en ajoutant une droite à l'infini, chaque parallèle se coupant sur cette droite. Il est donc possible de passer d'une géométrie à l'autre sans difficulté. Dans la suite de cette thèse, nous privilégions la variante projective de la géométrie d'incidence car elle est plus simple, il n'est pas nécessaire d'introduire un nouveau concept pour définir le parallélisme.

2 Géométrie d'incidence projective

Cette structure d'incidence offre une alternative pour définir les espaces projectifs. En effet, étant donné un triplet (Ω, Δ, Φ), représentant un ensemble de points, un ensemble de droites, et une relation d'incidence, on peut résumer en quelques axiomes, dus à Veblen et Young [VY18] repris par Coxeter [Cox03], toute l'information de la structure projective en dimension ≥ 1.

2.1 Système d'axiomes pour la géométrie d'incidence projective 2D

Pour décrire la géométrie d'incidence projective plane, nous présentons le système d'axiomes standard à la Table I.1.2 que nous illustrons dans la Table I.1.3.

2.1.1 Système d'axiomes standard

Les deux premiers axiomes **(A1P2)** **(A2P2)** concernent la construction des points et des droites. Le premier permet de créer la droite passant par deux points tandis que le second définit l'intersection entre deux droites. L'axiome **(A2P2)** correspond à notre axiome des parallèles en géométrie projective : deux droites du plan se coupent toujours.

Nous n'avons pas besoin de spécifier que les points impliqués (resp. droites) doivent être différents dans l'axiome *Line-Existence* (resp. *Point-Existence*). En effet, si les points sont égaux, une telle droite (resp. point) existe bel et bien. Dans les faits, il existe une infinité de droites (resp. points) passant par deux points confondus (resp. droites). Ce choix théorique n'est pas sans importance, nous séparons ici la définition d'existence de l'unicité pour les points et les droites. Cela nous permet d'examiner une géométrie où il est toujours possible de montrer l'existence d'une droite passant par un point sans étendre notre système d'axiomes même si elle n'est pas toujours explicitement définie (il n'est pas impossible de choisir une droite parmi une infinité de droites passant par un point). Grâce à ces considérations, nous pouvons toujours fournir des témoins d'existence dans certains cas dégénérés lors des démonstrations. Néanmoins dans la plupart des livres [BR98, Bue95, Cox03], ces conditions de dégénérescence sont exclues pour être cohérent avec l'introduction des géométries finies où tous les points sont spécifiés distincts. Rajoutons que ce choix est relié aux fondements de la géométrie (incluant l'approche constructiviste de cette dernière [VP95]) qui est largement étudiée dans notre équipe. Dans ce cas, la construction d'une droite à partir de deux points nécessairement distincts n'est qu'un cas dérivé de l'axiome général que nous introduisons, où les points peuvent être confondus. L'axiomatisation de certaines parties de la géométrie est un problème contemporain que nous étudions à nouveau à travers la formalisation de ces dernières dans des assistants de preuves [DDS00, FT10, MF03, MNS11, Nar06a].

L'axiome **(A3P2)** concerne l'unicité de deux objets bien définis. En effet, si deux points sont incidents aux mêmes deux droites alors soit les points sont égaux, soit les droites sont confondues. Ensuite, l'axiome **(A4P2)** énonce que chaque droite contient au minimum trois points. Cet axiome empêche que certains petits systèmes exceptionnels soient appelés plans projectifs. Finalement, l'axiome **(A5P2)** exprime qu'il existe toujours deux droites distinctes pour former au minimum une géométrie planaire, ce qui signifie que la dimension est au moins 2. En couplant cet axiome avec l'axiome **(A2P2)**, cette formalisation ne décrit que des plans. Par conséquent, ce système d'axiomes décrit une géométrie d'incidence projective plane.

(A1P2) Line-Existence : $\forall\ A\ B : \mathit{Point}, \exists\ l : \mathit{Line}, A \in l \wedge B \in l$

(A2P2) Point-Existence : $\forall\ l\ m : \mathit{Line}, \exists\ A : \mathit{Point}, A \in l \wedge A \in m$

(A3P2) Uniqueness : $\forall\ A\ B : \mathit{Point}, \forall\ l\ m : \mathit{Line},$
$A \in l \wedge B \in l \wedge A \in m \wedge B \in m \Rightarrow A = B \vee l = m$

(A4P2) Three-Points : $\forall\ l : \mathit{Line}, \exists\ A\ B\ C : \mathit{Point},$
$A \neq B \wedge B \neq C \wedge A \neq C \wedge A \in l \wedge B \in l \wedge C \in l$

(A5P2) Lower-Dimension : $\exists\ l\ m : \mathit{Line}, l \neq m$

TABLE I.1.2 – Système d'axiomes standard pour la géométrie projective 2D.

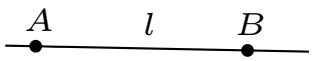
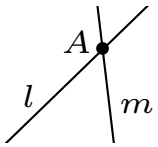
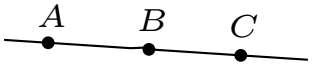
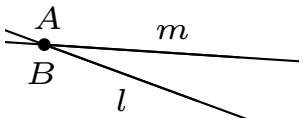
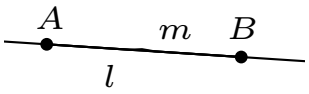
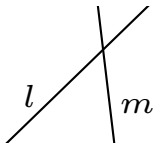
A1P2	A2P2	A3P2
		
A4P2		**A5P2**
		

TABLE I.1.3 – Illustrations du système d'axiomes standard pour la géométrie projective 2D.

Pour éliminer certains détails techniques et améliorer la compréhension de nos systèmes d'axiomes, nous supprimons dans ce manuscrit la distinction entre le quantificateur existentiel standard `exist x` et la notation `{x | ...}` qui est présente dans notre développement. Cette seconde syntaxe dénote la quantification existentielle constructive sur la sorte `Type` noté `sig` en *Gallina*. Les deux définitions diffèrent uniquement au niveau de la hiérarchie des univers en Coq, là où `sig` utilise `Type`, `exist` utilise `Prop`. Nous rencontrons des problèmes avec la version classique de l'existence quand il s'agit de faire des éliminations dans des preuves constructives. Les règles de type *Gallina* nous interdisent d'éliminer un motif dont le type appartient à l'univers `Prop`, chaque fois que le type du résultat de l'élimination est différent de `Prop`. Le système de typage garantit ainsi que les détails des preuves ne peuvent en aucun cas affecter les parties d'un développement qui ne sont pas aussi marquées comme des preuves. À l'aide de `sig`, il est possible d'effectuer une analyse de cas sur des éléments existentiels pour produire des objets dans `Set`.

Tous ces détails liés à la théorie des univers dans Coq sont disponibles dans le livre de référence [BC04]. Des explications plus détaillées sur notre approche constructiviste de la géométrie d'incidence projective plane sont disponibles dans [MNS11].

2.1.2 Système d'axiomes alternatif

Le système d'axiomes précédent inspiré des travaux [BR98,Cox03] est souvent décrit sous une forme minimaliste alternative dans la littérature (voir Table I.1.4). Les trois premiers axiomes **(A1P2)** **(A2P2)** et **(A3P2)** restent strictement identiques. L'axiome **(A4P2')** indique qu'il existe quatre points sans aucun triplet de points colinéaires. L'équivalence entre cet axiome *Four-Points* et la décomposition *Three-Point*, *Lower-Dimension* est établie dans [MNS11]. Intuitivement, l'axiome *Four-Points* sous-entend que le plan projectif ne peut pas être réduit à une simple droite et qu'il est possible grâce à l'axiome *Point-Existence* de construire systématiquement un troisième point sur une des droites (voir Table I.1.5). Nous préférons la décomposition de la Table I.1.2 pour faciliter la décomposition et la mécanisation des preuves. En effet, l'axiome **(A4P2')** est souvent plus compliqué à manipuler dans les preuves.

(A1P2) Line-Existence : $\forall\ A\ B : Point, \exists\ l : Line, A \in l \wedge B \in l$

(A2P2) Point-Existence : $\forall\ l\ m : Line, \exists\ A : Point, A \in l \wedge A \in m$

(A3P2) Uniqueness : $\forall\ A\ B : Point, \forall\ l\ m : Line,$
$A \in l \wedge B \in l \wedge A \in m \wedge B \in m \Rightarrow A = B \vee l = m$

(A4P2') **Four-Points** : $\exists\ A\ B\ C\ D : Point,$
$A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D \wedge$
$(\forall\ l : Line,$
$(A \in l \wedge B \in l \Rightarrow C \notin l \wedge D \notin l) \wedge$
$(A \in l \wedge C \in l \Rightarrow B \notin l \wedge D \notin l) \wedge$
$(A \in l \wedge D \in l \Rightarrow B \notin l \wedge C \notin l) \wedge$
$(B \in l \wedge C \in l \Rightarrow A \notin l \wedge D \notin l) \wedge$
$(B \in l \wedge D \in l \Rightarrow A \notin l \wedge C \notin l) \wedge$
$(C \in l \wedge D \in l \Rightarrow A \notin l \wedge B \notin l)\ )$

TABLE I.1.4 – Système d'axiomes alternatif pour la géométrie projective 2D.

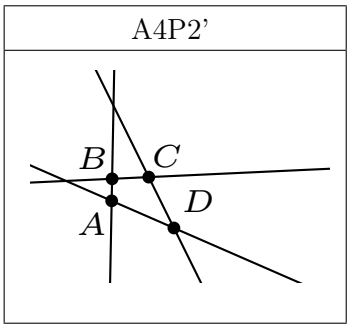


TABLE I.1.5 – Illustration de l'axiome alternatif (A4P2').

Notons qu'il existe bien d'autres axiomatisations capturant la géométrie d'incidence projective plane. Il est par exemple possible de considérer la formalisation d'un système d'axiomes à partir de points et d'une relation d'incidence.

2.1.3 Aparté sur la dualité

Il est bien connu que la géométrie projective plane bénéficie d'un principe de dualité, à savoir que chaque définition reste significative et chaque théorème reste vrai, lorsque nous échangeons les objets point et droite. Ce principe est un résultat théorique intéressant qui permet de manière pratique de prouver pour chaque théorème géométrique son dual en utilisant un foncteur qui échange les objets et les prédicats. De plus, cette dualité se généralise aux dimensions supérieures en échangeant chaque objet de dimension avec sa co-dimension associée. Dans un espace de dimension n, les points (dimension 0) correspondent aux hyperplans (codimension 1), les droites passant par deux points (dimension 1) correspondent à l'intersection de deux hyperplans (codimension 2), etc.

2.2 Système d'axiomes pour la géométrie d'incidence projective ≥3D et 3D

De la même manière, nous présentons dans la Table I.1.6 et illustrons dans la Table I.1.7, un système d'axiomes pour la géométrie en dimension supérieure.

Le système contient toujours cinq axiomes dont trois d'entre eux restent inchangés **(A1P3)** **(A3P3)** et **(A4P3)**. *Pasch* se substitue à **(A2P2)** en affirmant que deux droites coplanaires sont toujours concourantes. De plus, nous modifions l'axiome *Lower-Dimension* pour définir une géométrie qui est au moins un espace. Pour cela, nous supposons qu'il existe deux droites qui ne se coupent pas. Cette fois-ci, l'axiome de *Pasch* ne nous permet pas de borner la dimension supérieure, c'est pourquoi nous la qualifions de géométrie ≥3D.

Bien évidemment, il est possible de restreindre la dimension de cette géométrie à l'espace en ajoutant un axiome optionnel pour limiter la dimension à exactement 3. Cet axiome **(A6P3)** spécifie qu'il y a toujours une droite qui coupe trois autres droites non coplanaires.

(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 I.1.6 – Système d'axiomes pour la géométrie projective 3D.

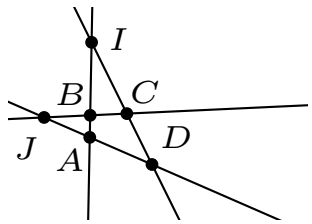
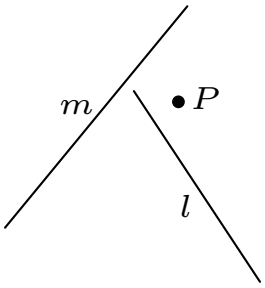
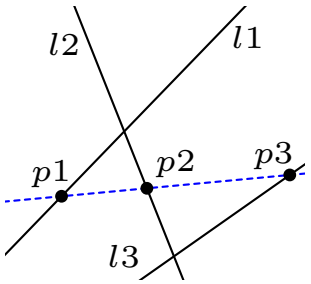
A2P3	A5P3	A6P3
		

TABLE I.1.7 – Illustrations du système d'axiomes standard pour la géométrie projective 3D.

2.3 Formalisation Coq

La formalisation Coq de ces quatre systèmes d'axiomes est plutôt immédiate. Par souci de clarté, nous exposons ci-dessous une version simplifié de notre implantation de la géométrie d'incidence projective à la fois en 2D, ≥3D et 3D.

Pour améliorer la modularité, nous exploitons le mécanisme de « *type classes* » [CS12, SO08] et de foncteurs en Coq. Les classes de types d'abord définies dans Haskell [Jon03] sont un outil très utile pour écrire des programmes sur des structures abstraites tout en apportant un mécanisme de surcharge des notations. Afin d'analyser les dépendances entre les différents systèmes d'axiomes et observer les équivalences entre différentes variantes d'axiomatisations, nous divisons nos ensembles d'axiomes en différentes classes. Nous examinons ainsi en détails l'équivalence entre chaque classe d'axiomes avec les deux formalisations de la géométrie d'incidence projective. De plus, ces classes sont construites incrémentalement en ajoutant les axiomes un à un. Cette classification nous permet de mettre en évidence le système d'axiomes minimal requis pour démontrer un ensemble de propriétés. Finalement, les classes de types en Coq permettent de vérifier que nous avons bien réussi à prouver une théorie à partir d'une autre grâce à l'instantiation.

Premièrement, nous créons la classe `ProjectiveStructure` avec les définitions des deux objets : point et droite, de la relation d'incidence et de sa décidabilité [2] (voir Table I.1.8). Nous regroupons ensuite les axiomes communs entre les différentes dimensions à savoir *Line-Existence*, *Uniqueness* et *Three-Points* dans la Table I.1.10. Enfin, nous construisons trois classes, la première capture l'intégralité du plan projectif (voir Table I.1.11), la seconde (voir Table I.1.12) décrit l'espace projectif ≥3D et la dernière (voir Table I.1.13) formalise exactement l'espace projectif 3D.

2. {Incid A l} + {~ Incid A l} est la notation constructive de la décidabilité (Incid A l) ∨ (¬ Incid A l)

```
(* Types *)
Class ProjectiveStructure := {

Point : Set;
Line : Set;
Incid : Point -> Line -> Prop;

(* décidabilité pour la relation d'incidence *)
incid_dec : forall (A : Point)(l : Line), {Incid A l} + {~Incid A l}

}.
```

TABLE I.1.8 – Classe de types pour la structure projective.

Les questions de décidabilité et les relations d'égalité sont les principales différences entre la formalisation Coq et les systèmes d'axiomes montrés précédemment.

Étant donné que la logique sous-jacente à Coq est intuitionniste, nous devons explicitement déclarer quels prédicats sont décidables. Il s'agit d'indiquer au système les concepts qui satisfont la propriété du tiers-exclu [3] de la logique classique. De cette manière, nous élargissons le fragment décidable des théorèmes qu'il est possible de prouver. En admettant seulement la décidabilité du prédicat `Incid`, nous prouvons la décidabilité de l'égalité à la fois pour les points et pour les droites [MNS11]. Les trois décidabilités sont toutes reliées à travers la définition de l'axiome **(A3P2)** de la géométrie d'incidence projective. Ce choix de conception ne complique en rien les preuves. Une fois que la décidabilité de l'égalité sur les points ou les droites est démontrée, nous différencions nos objets comme en logique classique : soit deux objets sont égaux, soit ils ne le sont pas.

L'autre choix de formalisation que nous avons effectué est que l'égalité sur les points diffère de l'égalité classique de Leibniz [4]. Cette égalité sur les points, notée `[==]`, est une relation d'équivalence qui devient un paramètre de notre théorie formellement défini en amont de la classe `ProjectiveStructure` dans la Table I.1.9. Elle permet de rendre transparente la manière dont les points sont construits. En d'autres termes, l'égalité rend possible la substitution d'un point par un autre si ils sont constructivement équivalents mais pas nécessairement syntaxiquement égaux. En associant cette égalité à la non distinction des points lors de la formalisation du premier axiome **(A1P2)**, il est possible de construire une partie de la géométrie d'incidence projective avec une approche constructiviste [Kah95, VP95]. Lorsque nous souhaitons rendre une partie de notre formalisation calculatoire, nous introduisons la décidabilité sur cette égalité paramétrique qui a été prouvée en utilisant celle sur l'incidence. Pour les droites, un traitement analogue reste à faire : l'égalité de Leibniz est encore employée pour représenter l'égalité entre les droites. Nous justifions la non utilisation d'une égalité paramétrique pour les droites de deux manières : premièrement, les droites ne forment pas un objet primitif de la formalisation de la géométrie d'incidence projective du Chapitre I.2 ; en effet dans ce cadre les droites seront représentées par des ensembles de points ; deuxièmement, nous souhaitons mesurer les principales différences entre les deux égalités dans l'automatisation des preuves. Pour rigoureusement compléter notre librai-

3. Le tiers exclu aristotélicien dit que : si ça n'est pas vrai, c'est faux ; si ça n'est pas faux, c'est vrai ; pas d'autres possibilités. En d'autres termes, deux propositions contradictoires ne peuvent pas être toutes les deux fausses. Cette propriété possède des formulations équivalentes comme par exemple la double négation, la loi de Peirce ou la loi de De Morgan.

4. L'égalité classique respecte les axiomes généraux de l'identité, de la symétrie et de la transitivité. Leibniz propose le principe suivant comme définition de l'égalite : deux êtres sont égaux lorsque tout ce qui est vrai de l'un est vrai de l'autre.

rie sur la géométrie, une égalité paramétrique pour les droites similaire à celle des points doit être considérée. Ce détail technique permettant de mieux étudier la logique sous-jacente à l'assistant de preuve Coq n'empêche en rien la formalisation de la géométrie d'incidence projective ainsi que les nombreuses preuves qui lui sont associées. Nous n'y reviendrons pas dans la suite de ce manuscrit.

```
(* décidabilité sur les points *)
Class EqDecidability `(U : Set) := {

(* notation et décidabilité *)
Notation "s [==] t" := (s === t) (at level 70, no associativity).
eq_dec_u : forall A B : U, {A [==] B} + {~ A [==] B}

}.
```

TABLE I.1.9 – Classe de types pour l'égalité sur les points.

```
(* nD *)
Class ProjectiveStructureLEU `(PS : ProjectiveStructure) := {

(* A1P2-P3 Line-Existence *)
a1_exist: forall (A B : Point) , exists l : Line, Incid A l /\ Incid B l;

(* A3P2-P3 Uniqueness *)
uniqueness: forall (A B : Point)(l1 l2 : Line), Incid A l1 -> Incid B l1 ->
  Incid A l2 -> Incid B l2 -> A [==] B \/ l1 = l2;

(* A4P2-P3 Three-Points *)
a3_1: forall l : Line, exists A : Point, exists B : Point, exists C : Point,
  (~ A [==] B /\ ~ A [==] C /\ ~ B [==] C /\
  Incid A l /\ Incid B l /\ Incid C l)

}.
```

TABLE I.1.10 – Classe de types pour la structure projective indépendante de la dimension.

```
(* 2D *)
Class ProjectiveStructurePlane `(PSLEU : ProjectiveStructureLEU) := {

(* A2P2 Point-Existence *)
a2_exist : forall (l1 l2 : Line), exists A : Point, Incid A l1 /\ Incid A l2;

(* A5P2 Lower-Dimension *)
a3_2 : exists l1 : Line, exists l2 : Line, l1 <> l2;

}.
```

TABLE I.1.11 – Classe de types pour le plan projectif.

```
(* >= 3D *)
Class ProjectiveSpaceOrHigher `(PSLEU : ProjectiveStructureLEU) := {

(* A2P3 Pasch *)
a2: forall A B C D : Point, forall lAB lCD lAC lBD : Line,
  ~ A [==] B /\ ~ A [==] C /\ ~ A [==] D /\
  ~ B [==] C /\ ~ B [==] D /\ ~ C [==] D ->
  Incid A lAB /\ Incid B lAB ->
  Incid C lCD /\ Incid D lCD ->
  Incid A lAC /\ Incid C lAC ->
  Incid B lBD /\ Incid D lBD ->
  (exists I : Point, (Incid I lAB /\ Incid I lCD)) ->
  exists J : Point, (Incid J lAC /\ Incid J lBD);

(* A5P3 Lower-Dimension *)
a3_2: exists l1 : Line, exists l2 : Line, forall p : Point,
  ~(Incid p l1 /\ Incid p l2)

}.
```

TABLE I.1.12 – Classe de types pour l'espace projectif au moins en dimension 3.

```
(* 3D *)
Class ProjectiveSpace `(PSH : ProjectiveSpaceOrHigher) := {

Intersect_In (l1 l2 : Line) (P : Point) := Incid P l1 /\ (Incid P l2);

(* A6P3 Upper-Dimension *)
a3_3 : forall l1 l2 l3 : Line, ~ l1 = l2 /\ ~ l1 = l3 /\ ~ 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)

}.
```

TABLE I.1.13 – Classe de types pour l'espace projectif exactement en dimension 3.

3 Méthodologie et automatisation de la démonstration en géométrie d'incidence

Dans cette section, nous décrivons la méthodologie que nous avons suivie pour effectuer des preuves en géométrie d'incidence ainsi que l'automatisation qu'il est possible d'apporter dans ces démonstrations. Afin de mécaniser le déroulement des preuves dans un contexte géométrique, nous devons identifier les étapes clés de démonstration apparaissant de façon récurrente et évaluer leur difficulté. Ces étapes générales sont ensuite automatisées grâce aux différents outils proposés par les assistants de preuves. Dans la suite, cette segmentation des démonstrations en petits morceaux de preuves représentatifs avec les stratégies de mécanisation qui les accompagnent est appelé « élément de preuves ». Nous divisons la classification de ces élements de preuves en 5 catégories distinctes que nous ne hiérarchisons pas :

- Tests de décidabilité
- Clôture des hypothèses
- Résolution ou contradiction
- Création d'objets
- Identification et application d'un motif

3.1 Tests de décidabilité - difficulté modérée

Cette partie regroupe tous les tests de décidabilité qui sont effectués lors d'une démonstration. En effectuant un test de décidabilité, la démonstration est scindée en deux parties à cause de la disjonction naturelle que l'on retrouve dans la définition de chacune des décidabilités (points, droites, incidence, ...). Ces disjonctions peuvent être provoquées d'une part par des tests d'égalité entre objets, d'autre part par des tests d'incidence. Les disjonctions sont essentielles pour traiter séparément les cas dégénérés d'un énoncé géométrique et diviser la démonstration en plusieurs cas de figure. L'importance de la prise en compte de ces cas dégénérés est soulevée lors de l'introduction [BNSB14a, DDS00, MF03, Nar06b]. De plus, ces tests sont bien souvent mis en œuvre pour avancer dans la démonstration et permettre l'application des autres catégories.

Exemple. $\forall\ A\ B : Point, \forall\ l\ m : Line,$
$A \in l \wedge B \in l \wedge A \in m \wedge A \neq B \Rightarrow l = m \vee l \neq m$

Afin d'illustrer ce mécanisme de disjonction, nous donnons un exemple de lemme ad hoc dérivé de l'axiome *Uniqueness* **(A3P2)** qui ne contient pas l'hypothèse $B \in m$. Le résultat étant trivial si la décidabilité de l'égalité sur les droites est prouvée, nous considérons uniquement la décidabilité de l'incidence dans cet exemple. L'étape clé de cette démonstration est le test de décidabilité sur l'incidence du point B à la droite m. Soit $B \in m$ alors $l = m$ d'après l'axiome d'*Uniqueness*. Soit $B \notin m$ alors $l \neq m$ puisqu'il existe un point distinct de A sur l qui n'appartient pas à m.

Pour finir, l'imbrication des tests de décidabilité subdivise à chaque fois la preuve en deux (voir Figure I.1.1). Il est donc important d'uniquement considérer les tests qui sont nécessaires à l'accomplissement de la démonstration pour éviter d'alourdir cette dernière. Nous omettons pour le moment l'ordre de ces imbrications et l'élagage de l'arbre de tests que nous détaillons dans le Chapitre II.1.

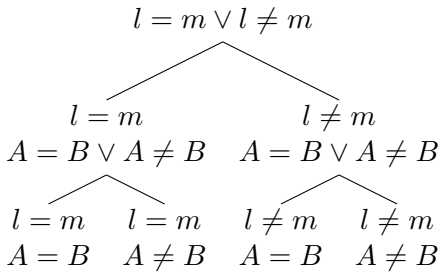


FIGURE I.1.1 – Imbrication des tests de décidabilité.

3.2 Clôture des hypothèses - difficulté modérée

Cette étape consiste à compléter le contexte en déduisant toutes les informations qui sont simplement des transformations et des manipulations d'hypothèses. Le terme de *saturation* est aussi employé pour décrire cette tâche. Pour réaliser cette saturation, il faut déterminer pour chaque définition de notre théorie les transformations et les propriétés élémentaires qui lui sont associées. À partir du prédicat de base qu'est l'incidence, il est possible de construire des définitions plus complexes permettant de représenter une configuration particulière de notre énoncé géométrie.

Par exemple, en géométrie spatiale, il est très rapidement nécessaire d'introduire des définitions pour les différentes généralisations d'une relation d'incidence dans un espace projectif (colinéarité, coplanarité, ...). La création de ces définitions est une démarche obligatoire pour simplifier l'expression des configurations en géométrie d'incidence projective. En effet, pour prouver une relation de coplanarité non dégénérée entre quatre points non alignés, nous devons trouver une application de l'axiome de *Pasch* **(A2P3)** qui compte pas moins de 10 incidences et 6 inégalités entre points. Identifier plusieurs coplanarités en simultané dans notre contexte devient donc rapidement une tâche laborieuse. Nonobstant, pour être capable de saturer le contexte en utilisant ces définitions, nous devons à la fois pouvoir identifier de nouvelles relations d'incidences et appliquer des propriétés de réflexivité (trivialité), symétrie (permutation) et "transitivé" (pseudo-transitivité) sur ces dernières [BNS15]. Une relation d'incidence de dimension n et d'arité $n+1$ respecte un ensemble de propriétés résumé pour la colinéarité dans la Table I.1.15, pour la coplanarité dans la Table I.1.16 et de manière généralisée par la Table I.1.17. Remarquons que la généralisation de la relation d'incidence à la dimension 1 représente l'égalité entre points et possède uniquement 3 propriétés étant donné que les propriétés de permutations sont équivalentes (voir Table I.1.14).

Eg-trivial : $\forall\ A : Point,\ Eg\ A\ A$

EG-perm1 : $\forall\ A\ B : Point,\ Eg\ A\ B \rightarrow Eg\ B\ A$

Eg-ptrans : $\forall\ X\ A\ B : Point,$
$Eg\ X\ A \wedge Eg\ X\ B \rightarrow Eg\ A\ B$

TABLE I.1.14 – Propriétés vérifiées par la relation d'incidence *Eg* d'arité 2.

Col-trivial : $\forall\ A\ B : Point, Col\ A\ A\ B$

Col-perm1 : $\forall\ A\ B\ C : Point, Col\ A\ B\ C \rightarrow Col\ B\ C\ A$

Col-perm2 : $\forall\ A\ B\ C : Point, Col\ A\ B\ C \rightarrow Col\ B\ A\ C$

Col-ptrans : $\forall\ X\ Y\ A\ B\ C : Point, X \neq Y$
$Col\ X\ Y\ A \wedge Col\ X\ Y\ B \wedge Col\ X\ Y\ C \rightarrow Col\ A\ B\ C$

TABLE I.1.15 – Propriétés vérifiées par la relation d'incidence Col d'arité 3.

Cop-trivial : $\forall\ A\ B\ C : Point, Cop\ A\ A\ B\ C$

Cop-perm1 : $\forall\ A\ B\ C\ D : Point, Cop\ A\ B\ C\ D \rightarrow Cop\ B\ C\ D\ A$

Cop-perm2 : $\forall\ A\ B\ C\ D : Point, Cop\ A\ B\ C\ D \rightarrow Cop\ B\ A\ C\ D$

Cop-ptrans : $\forall\ X\ Y\ Z\ A\ B\ C\ D : Point, \neg\ Col\ X\ Y\ Z$
$Cop\ X\ Y\ Z\ A \wedge Cop\ X\ Y\ Z\ B \wedge$
$Cop\ X\ Y\ Z\ C \wedge Cop\ X\ Y\ Z\ D \rightarrow Cop\ A\ B\ C\ D$

TABLE I.1.16 – Propriétés vérifiées par la relation d'incidence Cop d'arité 4.

Co_n-trivial : $\forall\ X_1\ X_2 \ldots X_{n-1} : Point, Co_n\ X_1\ X_1\ X_2 \ldots X_{n-1}$

Co_n-perm1 : $\forall\ X_1\ X_2 \ldots X_n : Point, Co_n\ X_1\ X_2 \ldots X_n \rightarrow Co_n\ X_n\ X_1 \ldots X_{n-1}$

Co_n-perm2 : $\forall\ X_1\ X_2 \ldots X_n : Point, Co_n\ X_1\ X_2 \ldots X_n \rightarrow Co_n\ X_2\ X_1 \ldots X_n$

Co_n-ptrans : $\forall\ Y_1\ Y_2 \ldots Y_{n-1}\ X_1\ X_2 \ldots X_n : Point, \neg\ Co_{n-1}\ Y_1\ Y_2 \ldots Y_{n-1}$
$Co_n\ Y_1\ Y_2 \ldots Y_{n-1}\ X_1 \wedge Co_n\ Y_1\ Y_2 \ldots Y_{n-1}\ X_2 \wedge \ldots \wedge$
$Co_n\ Y_1\ Y_2 \ldots Y_{n-1}\ X_{n-1} \wedge Co_n\ Y_1\ Y_2 \ldots Y_{n-1}\ X_n \rightarrow$
$Co_n\ X_1\ X_2 \ldots X_{n-1}\ X_n$

TABLE I.1.17 – Propriétés vérifiées par la relation d'incidence Co_n d'arité n.

Ces propriétés forment les lemmes usuels associés à chacune de ces relations d'incidence. Ces lemmes doivent ensuite être intégrés dans le mécanisme de saturation. Dans la Figure I.1.2, nous illustrons sur un exemple simple ce mécanisme en saturant la définition de colinéarité `on_line` (voir Table I.1.18) indiquant que trois points A, B, C sont alignés sur une droite l. De plus, nous profitons de l'incidence du point D à la droite l pour découvrir d'autres colinéarités sur la droite

l. Notons que la clôture déploie aussi récursivement les définitions pour effectuer de nouvelles déductions. Toutefois, ce mécanisme de saturation ne doit pas surcharger inutilement le contexte d'hypothèses en empêchant ainsi la suite de la progression de la preuve. Cette étape souvent terminale doit mettre en évidence un cas absurde ou conclure un but courant.

```
Definition on_line A B C l := Incid A l /\ Incid B l /\ Incid C l.
```

TABLE I.1.18 – Définition de la colinéarité à une droite.

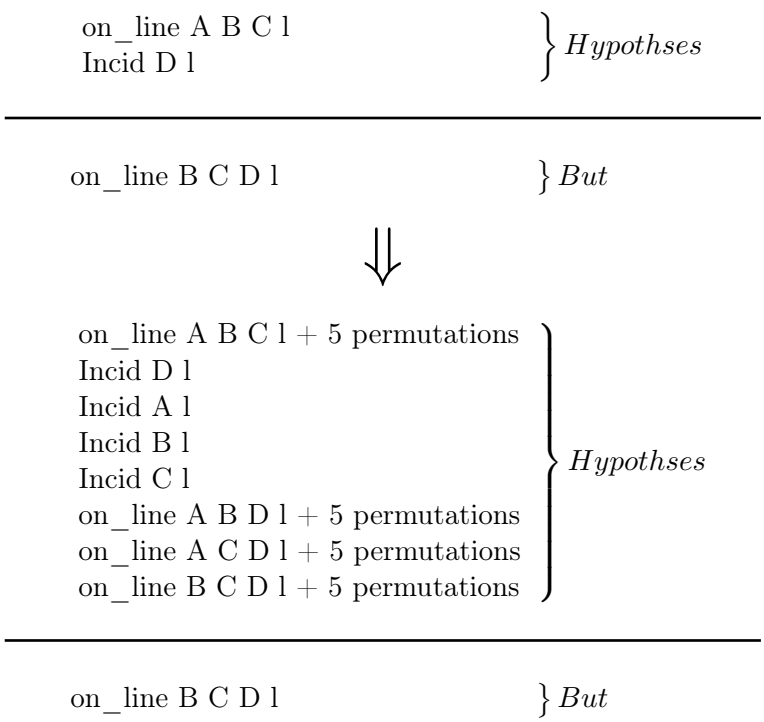


FIGURE I.1.2 – Clôture des hypothèses dans un exemple simple.

3.3 Résolution ou contradiction - difficulté faible

La troisième méthode vient compléter la clôture des hypothèses en terminant la preuve du but courant. Deux cas de figure se présentent : soit nous réussissons à déduire une hypothèse identique au but recherché, soit nous sommes en présence d'un contexte incohérent où il n'est pas possible de prouver le but. Dans ce dernier cas, l'objectif est alors d'identifier deux hypothèses contradictoires qui permettent de mettre fin à la démonstration. Cette étape relativement courte est fortement dépendante de la capacité de saturation de la sous-section précédente.

3.4 Création d'objets - difficulté très forte

Les trois premières étapes permettent de faire évoluer un contexte à partir des informations qui sont déjà connues. Parfois, toute l' information nécessaire au déroulement d'une preuve n'est pas encore disponible. Il est nécessaire de créer de nouvelles entités (points, droites, ...). Cette

création d'objets s'effectue à partir des axiomes disposant d'une quantification existentielle. En géométrie plane, nous pouvons créer de nouveaux points à partir de **(A1P2)** **(A4P2)** et de nouvelles droites avec **(A2P2)** **(A5P2)**. En géométrie spatiale, nous construisons de nouveaux points grâce à **(A2P3)** **(A4P3)** et de nouvelles droites avec **(A1P3)** **(A5P3)** **(A6P3)**. Nous remarquons que la quasi-totalité des axiomes permet de créer de nouveaux objets dans des configurations bien spécifiques. C'est une étape charnière qui est difficile à maîtriser sans intuition sur la résolution du problème. En effet, trouver l'idée générale d'une preuve en construisant tous les objets intermédiaires nécessaires à la démonstration est une tâche difficile même pour un mathématicien.

Dans un système général qui est automatique ; si on applique systématiquement les axiomes dès que possible, on obtient très rapidement une explosion du nombre d'objets manipulés sans pour autant garantir la construction des objets nécessaires à la finalisation de la démonstration. Afin de contrôler cette explosion, il est possible de considérer une construction par niveau de profondeur où on effectue d'abord toutes les déductions possibles avant de rajouter les nouveaux objets parmi les hypothèses. En pratique, on se limite au maximum à une profondeur 2 ou 3 comme nous l'illustrons dans I.1.19. Cette limitation n'est cependant pas suffisante dans certaines configurations géométriques compliquées où le niveau de profondeur requis est plus élevé et l'explosion du nombre d'objets engendrés devient trop importante pour le système.

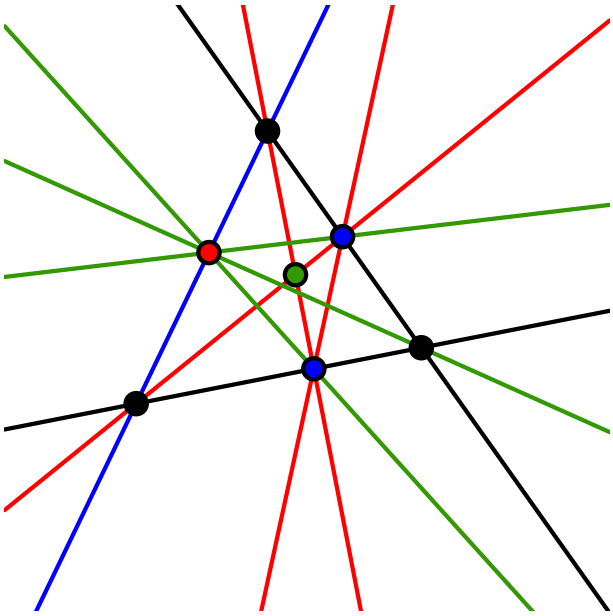


TABLE I.1.19 – Illustration de la construction d'objets par niveau de profondeur sur un exemple à trois points et deux droites. Noir configuration initiale, bleu : 1er niveau, rouge : 2ème niveau, vert : 3ème niveau.

Niveau 1 : Droites bleues construites à partir de l'axiome **(A1P2)** ; deux points bleus construits à partir de l'axiome **(A4P2)** ;

Niveau 2 : Droites rouges construites à partir de l'axiome **(A1P2)** ; point rouge construit à partir de l'axiome **(A4P2)** ;

Niveau 3 : Droites vertes construites à partir de l'axiome **(A1P2)** ; point vert construit à partir de l'axiome **(A2P2)**.

Il est important de minimiser la création de nouveaux objets pour simplifier l'étape de saturation. Étant donné qu'il est difficile de guider le système pour construire les « bons » objets, nous préférons confier cette tâche à l'utilisateur. Dans le cadre d'une aide à la preuve qui n'est pas entièrement automatisée, il est acceptable que l'utilisateur indique les objets qui sont nécessaires pour continuer la preuve. Cette intuition facilite grandement la tâche pour les autres étapes. Dans la suite de nos travaux, nous considérons l'hypothèse suivante :

Hypothèse. *La construction d'objets complémentaires à l'énoncé est une tâche exclusivement manuelle qu'on ne va pas chercher à automatiser.*

En admettant cette hypothèse, le système a toutes les informations nécessaires à disposition. Il peut se concentrer uniquement sur les déductions logiques solutionnant le problème.

3.5 Identification et application d'un motif - difficulté forte

La dernière étape consiste à identifier dans les preuves, les pas de démonstrations qu'on peut factoriser et regrouper dans un lemme. Ces morceaux de preuves doivent être paramétrables et apparaître plusieurs fois. L'idée sous-jacente est de décomposer une preuve complexe en utilisant des lemmes intermédiaires. Ces lemmes font partie du mécanisme d'aide à la preuve puisqu'ils facilitent la compréhension globale et découpent les preuves en petits morceaux réutilisables.

De la même manière que pour la création de points, si l'on souhaite que le système découvre automatiquement de nouveaux motifs pouvant être regroupés dans un lemme, le système doit posséder une intuition géométrique. Cette intuition géométrique est une tâche réservée aux mathématiciens et que l'on retrouve dans des domaines qui sont plutôt relatifs à l'intelligence artificielle. La conception d'une méthode d'identification automatique de ces motifs sort du cadre d'aide à la preuve que nous nous sommes fixés. Nous souhaitons assister le mathématicien dans sa preuve en prenant en charge les étapes triviales et répétitives. Si l'utilisateur identifie et isole dans un lemme séparé une telle étape, nous voulons l'aider à finaliser la preuve de ce dernier. Ensuite, nous pouvons continuer les preuves en ajoutant à la base de connaissances du système ce nouveau lemme. L'application des lemmes inclus dans cette base pour déduire de nouveaux résultats doit se faire automatiquement lors de la phase de saturation.

3.6 Stratégie d'ordonnancement

Une fois la classification de ces éléments de preuves terminée, il est possible de s'intéresser à l'ordonnancement de ces derniers. Nous réfléchissons ainsi à l'élaboration d'un premier prototype de tactique générale qui permet de simplifier les schémas de démonstrations en géométrie d'incidence projective en nous appuyant sur la méthodologie de la démonstration que nous venons de présenter. Rappelons que l'objectif principal est d'aider l'utilisateur à résoudre la plus grande classe de problèmes possibles en géométrie d'incidence projective sans négliger l'efficacité. En éliminant la création d'objets et l'identification d'un nouveau motif du processus automatique, nous pouvons proposer une ébauche de tactique en ordonnant les éléments de preuves comme suit :

1. Tests de décidabilité
2. Clôture des hypothèses et application de motifs

3. Résolution ou contradiction

Cet ordonnancement n'est cependant pas parfait. De nombreuses questions sur l'optimisation et l'ordre de ces éléments de preuves se posent dans le but de minimiser le temps d'exécution et/ou l'occupation mémoire.

Chaque démonstration est différente et nécessite un cheminement précis (avec des variantes) pour être menée à son terme. L'application d'un procédé automatique qui teste chacun des éléments de preuves séquentiellement à différents niveaux de profondeurs ne peut pas être sans aucun défaut. Une tactique doit être correctement conçue pour résoudre une classe spécifique de problèmes en manipulant les hypothèses sans jamais les éliminer. Pour optimiser cette tactique, l'utilisation d'astuces de génie logiciel orienté Coq permet par exemple de limiter le nombre d'unifications du système ou de fournir un maximum d'informations pouvant aider à la résolution. Nous détaillons quelque peu ce mécanisme d'unification dans le Chapitre I.2.

Si la terminaison d'une tactique et sa correction ne posent pas de problèmes, il est impossible de certifier la complétude de cette dernière. En effet, si la tactique appliquée permet d'effectuer un ensemble de déductions de manière automatique, son échec à résoudre le but courant peut être interprété de deux façons différentes : soit le design de la tactique n'est pas assez soigné et général pour prendre en compte ce cas de figure, soit la tactique n'est pas censée résoudre ce fragment de la théorie et il n'existe donc aucun cheminement possible pour cette dernière. Même en ayant identifié préalablement les différents éléments de preuves à inclure dans un tel procédé automatique, la validation d'une telle tactique nécessite beaucoup de vérifications et de tests.

La clôture des hypothèses que nous avons présentée peut être apparentée à une saturation du contexte avec toutes les informations qu'il est possible de déduire. À partir de là, le système dispose de toutes les connaissances pour continuer automatiquement la preuve du but courant. Néanmoins, cette méthode est coûteuse en temps et en espace. Comment optimiser cette clôture des hypothèses pour ne garder dans le contexte que les hypothèses encore utiles à la démonstration ?

Bien que nous éliminions l'identification automatique de nouveaux lemmes intermédiaires de notre prototype, nous souhaitons pouvoir appliquer mécaniquement dès que possible des lemmes à partir d'une base de connaissances. Comment identifier les parties de preuves qui méritent d'être segmentées dans un lemme intermédiaire ? Puis dans quel ordre doit-on les appliquer ?

On peut rapidement identifier une boucle de déductions très longue où la clôture permet de créer de nouvelles hypothèses qui sont utilisées par les lemmes intermédiaires pour déduire eux aussi de nouveaux résultats et ainsi de suite. Tous ces éléments de preuves sont ainsi entrelacés et répétés. À quel niveau de profondeur doit-on arrêter les nouvelles déductions ?

Toutes ces questions sont étudiées au chapitre suivant lors de la preuve de l'équivalence entre les deux formalisations de la géométrie, mais aussi au cours de ce manuscrit dans la formalisation des géométries finies et la mise en place du prouveur généralisé. Dans la section suivante, nous nous intéressons plus particulièrement au mécanisme de clôture des hypothèses en géométrie synthétique en analysant la capacité d'une telle géométrie à exprimer des énoncés de manière concise.

4 Expressivité de la théorie en géométrie synthétique

La taille d'un énoncé géométrique en nombre d'hypothèses dépend de l'expressivité de notre approche de la géométrie d'incidence. C'est-à-dire que les concepts de base que nous manipulons sont souvent mal adaptés pour exprimer des énoncés de haut niveau. En utilisant uniquement le concept d'incidence, il est très difficile de résoudre des problèmes impliquant des colinéarités

et des coplanarités entre objets. Il est alors habituel de fabriquer une hiérarchie de concepts pour améliorer la concision. Afin d'améliorer l'expressivité de l'approche en géométrie synthétique, nous sommes dans la nécessité d'introduire de nouvelles définitions pour les différentes relations d'incidence (égalité, colinéarité, coplanarité, ...). De cette manière, nous pouvons diminuer le nombre d'hypothèses traitées et l'opération de clôture des hypothèses est plus simple à réaliser. Nous ne nous limitons pas aux relations d'incidence, nous introduisons des définitions pour d'autres configurations géométriques plus complexes tel que les hexamys [5] [MNS12, MS06] pour faciliter l'expression d'un énoncé géométrique. L'inconvénient majeur est que la tactique générale que nous cherchons à mettre en place pour mécaniser la démonstration en géométrie d'incidence projective doit être modifiée et adaptée pour chaque définition supplémentaire que nous incluons. Définir une nouvelle notion implique d'ajouter un ensemble de règles permettant de faire des déductions et des simplifications sous forme de lemmes intermédiaires dans notre base de connaissance. En augmentant le nombre de définitions globales et le nombre de lemmes intermédiaires, nous complexifions sans cesse la méthode de résolution qui doit être capable d'entremêler tous ces nouveaux concepts. Cette géométrie élémentaire axiomatisée à partir de la seule notion d'incidence montre ici ses limites intrinsèques.

4.1 Propriétés fondamentales de la géométrie d'incidence projective

Pour examiner en détail l'expressivité de la géométrie synthétique sur des exemples précis, nous étudions plusieurs propriétés classiques de la géométrie d'incidence projective. Ces différentes propriétés que nous retrouvons tout au long de ce manuscrit sont tout d'abord rappelées au lecteur à travers une description générale.

4.1.1 Propriété de Desargues

Nous commençons par énoncer la propriété de Desargues (voir Figure I.1.3) qui est un aspect fondamental de la géométrie d'incidence projective.

Cette propriété a la particularité d'être toujours vraie dans un espace projectif de dimension supérieure ou égale à 3 construit sur un corps non forcément commutatif souvent appelé corps gauche ("skew field") dans la littérature. Autrement dit, si la propriété de Desargues est vérifiée alors il est possible de construire un espace projectif sur un corps gauche. Cette propriété s'énonce uniquement en termes d'incidence ; Hilbert met en évidence dans son livre « *Grundlagen der Geometrie* » [Hil60] que ses axiomes du plan similaires à ceux présentés dans la Table I.1.2 ne suffisent pas pour la démontrer. Cependant, si le plan peut être plongé dans l'espace, alors les axiomes d'incidence de la géométrie dans l'espace permettent sa démonstration. En d'autres termes, cette propriété non forcément vérifiée dans le plan devient un théorème en dimension supérieure. Les plans (affines ou projectifs) où la propriété de Desargues est vérifiée sont appelés plans désarguésiens (ou arguésiens).

Proprieté I.1.1 (Propriété de Desargues)**.** *Soit E un espace projectif* ≥ 2 *et P, Q, R, P', Q', R' des points de cet espace. Soient PQR et P'Q'R' deux triangles non aplatis et non confondus. Si les droites (PP'), (QQ') et (RR') sont concourantes en un point O alors* α*,* β *et* γ *sont alignés avec* $\alpha \in (PR) \cap (P'R')$*,* $\beta \in (QR) \cap (Q'R')$ *et* $\gamma \in (PQ) \cap (P'Q')$*.*

Description informelle. *Si deux triangles sont en perspectives par rapport à un point, alors ils sont en perspectives par rapport à une droite. Cette droite est la droite d'intersection des plans des deux triangles.*

5. Introduction aux hexamys : http ://hexamys.free.fr/

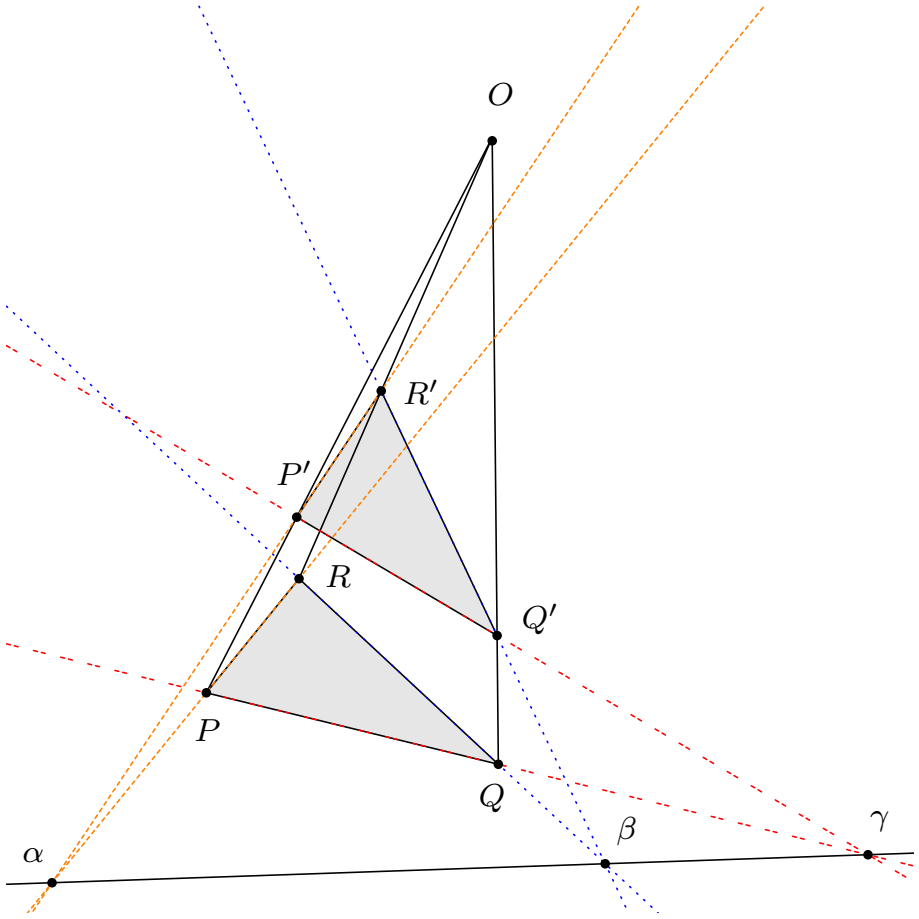


FIGURE I.1.3 – Une configuration du théorème de Desargues dans l'espace projectif.

Le plan fini de Fano illustré dans la Figure I.1.4 est l'exemple le plus simple de plan désarguésien que l'on peut donner. C'est le plus petit plan projectif fini, c'est à dire celui comportant le plus petit nombre de points et de droites, à savoir 7 de l'un et de l'autre. Ce plan projectif peut être défini de deux façons, soit comme le plan projectif sur le corps à deux éléments $F_2 = Z/2Z$ noté simplement PG(2,2), soit comme le plus petit plan projectif vérifiant les axiomes d'incidences de la Table I.1.2. Nous détaillons toute la théorie autour de la construction de ces plans finis et démontrons que ce plan est désarguésien dans le Chapitre II.1.

À contrario, il existe des plans non arguésiens, satisfaisant les axiomes de la géométrie plane où la propriété de Desargues n'est pas vérifiée tel que le célèbre plan de Moulton [Mou02] (voir Figure I.1.5). C'est un plan affine dans lequel les droites qui ont une pente négative voient leur pente doubler lorsqu'elles franchissent l'axe des ordonnées. Le plan de Moulton est fondé sur une structure d'incidence (P, L, I) vérifiant les propriétés d'un plan affine ; il peut être facilement étendu en un plan projectif en rajoutant une droite à l'infini [BR98, Cox03]. La figure I.1.5 illustre une configuration de la propriété de Desargues dans le plan de Moulton où les points α, β et γ ne sont pas alignés (le point α n'est pas représenté).

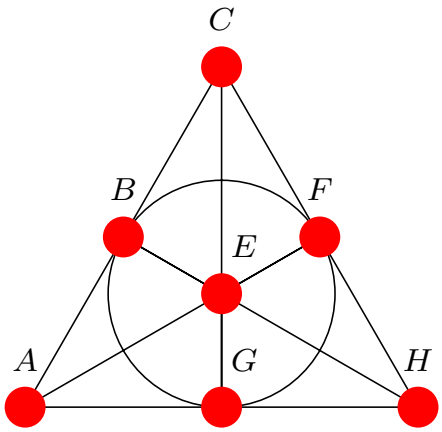


FIGURE I.1.4 – Plan projectif fini de Fano.

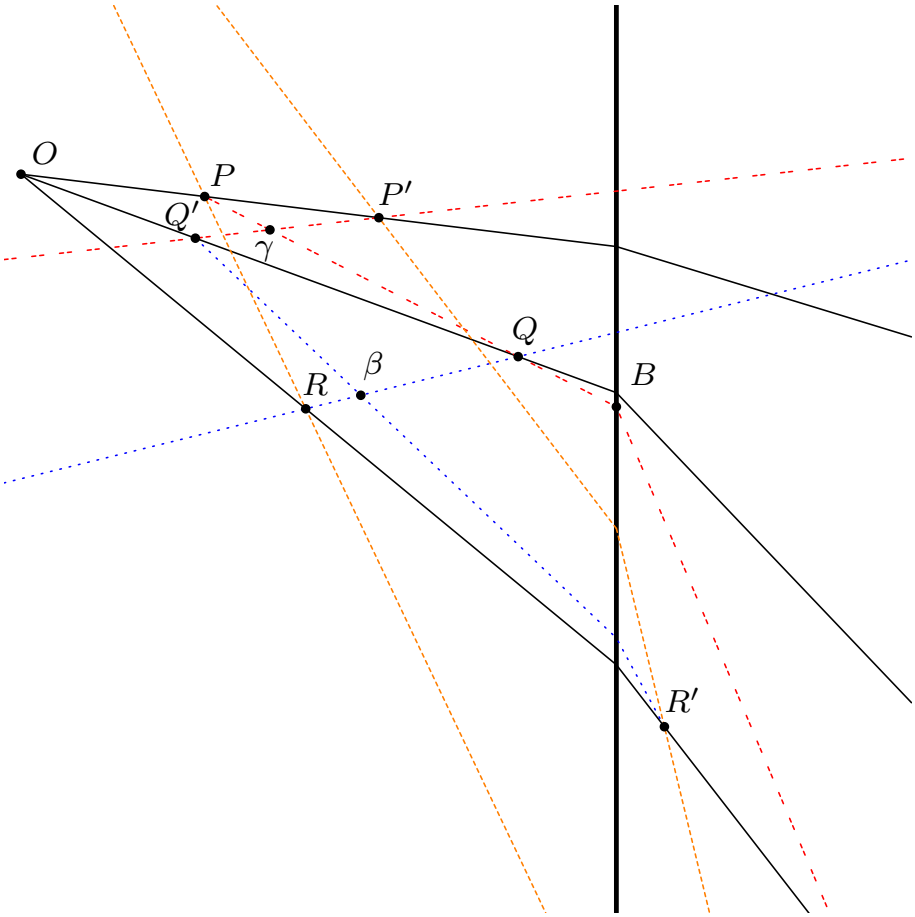


FIGURE I.1.5 – Configuration de Desargues dans le plan de Moulton.

Ajoutons une petite précision en rapport avec la dualité de la géométrie projective plane où les points correspondent à des droites et la colinéarité des points correspond à la concourance des droites. L'énoncé de la propriété de Desargues est « auto-dual » : l'axe de perspective devient le centre de perspective et vice versa.

4.1.2 Propriété de Pappus

Avec une approche axiomatique de la géométrie projective, si on veut construire un espace projectif cette fois-ci sur un corps commutatif, nous devons ajouter la propriété de Pappus aux différents systèmes d'axiomes.

Proprieté I.1.2 (Propriété de Pappus)**.** *Soit E un espace projectif de dimension ≥ 2 et F un sous-espace de E formant un plan. Dans ce plan F, soient P, Q, R trois points distincts alignés sur une droite d, et soient P' Q' R' trois autres points distincts alignés sur une droite d', alors les point α β γ sont alignés avec $\alpha = (RQ') \cap (QR')$, $\beta = (PR') \cap (RP')$ et $\gamma = (PQ') \cap (QP')$.*

Description informelle. *La propriété de Pappus est une configuration à 9 points et 9 droites où chaque droite passe par 3 points et chaque point est l'intersection de trois droites*[6].

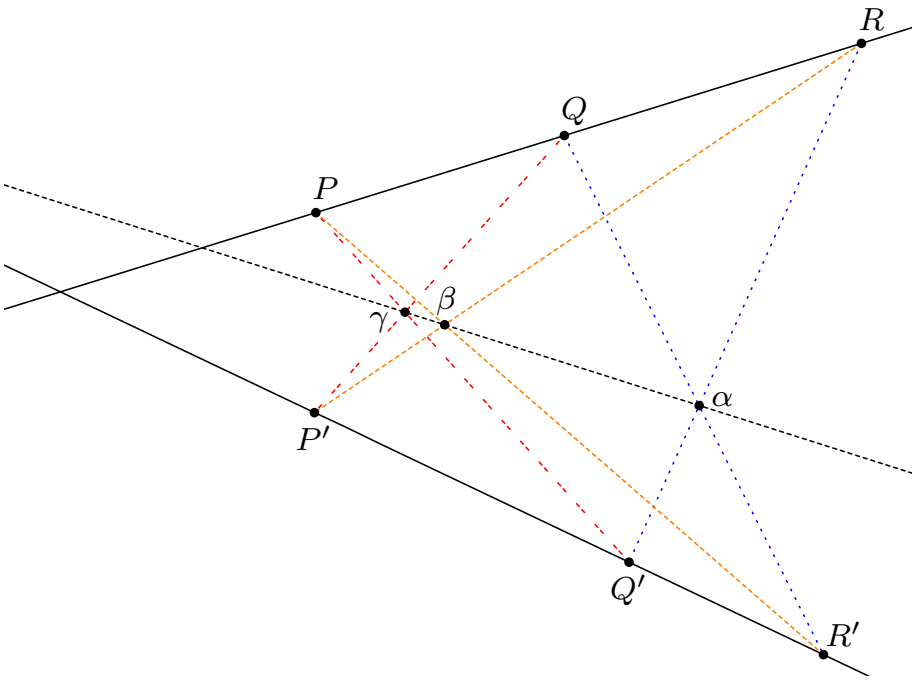


FIGURE I.1.6 – Une configuration du théorème de Pappus dans l'espace projectif.

4.1.3 Théorème de Hessenberg

Le théorème de Hessenberg [Cox03, RK70] montre que la propriété de Desargues se déduit de la propriété de Pappus en plus des axiomes d'incidence. De manière générale, la propriété de Pappus est vérifiée pour tout espace projectif construit sur un corps commutatif : on parle d'espace projectif pappusien. Un plan projectif pour lequel la propriété de Pappus n'est pas vérifiée est soit desarguésien lié à un corps gauche, soit non désarguésien.

6. Toute transformation projective entre deux droites d et d' est définie par l'image P', Q', R' de trois points distincts P, Q, R de d. P', Q', R' sont aussi tous distincts.

Théorème I.1.1 (Théorème de Hessenberg)**.** *Dans un plan projectif satisfaisant les axiomes d'incidence et la propriété de Pappus, la propriété de Desargues est vérifiée.*

4.2 Étude de l'expressivité avec l'approche en géométrie synthétique

Nous reprenons l'exemple de la propriété de Desargues que nous énonçons formellement de différentes manières en choisissant des cadres axiomatiques différents. Nous illustrons ainsi les problèmes d'expressivité introduits au début de la section 4.

4.2.1 Exemple du théorème de Desargues en 3D

Le théorème de Desargues s'exprime à partir des informations suivantes :

- 10 points : 6 points pour les deux triangles, le point de vue O et la construction des points α β γ qui sont alignés ;
- 10 droites : 3 droites concourantes en O le point de vue, 6 droites pour la construction des points α β γ et une droite pour l'alignement de ces 3 points ;
- 30 incidences : incidence de chacun des points à exactement 3 droites ;
- Conditions de non dégénérescence : deux triangles non aplatis et non confondus ainsi qu'un point de vue non coplanaire avec le triangle à la base du tétraèdre.

En regroupant toutes ces informations, nous obtenons l'énoncé Coq de la Table I.1.20.

```
(* Théorème de Desargues exprimé uniquement en incidence *)
(* Avec les axiomes de la géométrie d'incidence projective *)

Theorem Desargues :
forall O P Q R P' Q' R' alpha beta gamma lP lQ lR lPQ lPR lQR lP'Q' lP'R' lQ'R',
Incid P lPQ /\ Incid Q lPQ /\ Incid gamma lPQ /\
Incid P' lP'Q' /\ Incid Q' lP'Q' /\ Incid gamma lP'Q' /\
Incid P lPR /\ Incid R lPR /\ Incid alpha lPR /\
Incid P' lP'R' /\ Incid R' lP'R' /\ Incid alpha lP'R' /\
Incid Q lQR /\ Incid R lQR /\ Incid beta lQR /\
Incid Q' lQ'R' /\ Incid R' lQ'R' /\ Incid beta lQ'R' /\
Incid O lP /\ Incid P lP /\ Incid P' lP /\
Incid O lQ /\ Incid Q lQ /\ Incid Q' lQ /\
Incid O lR /\ Incid R lR /\ Incid R' lR /\
~ (exists l, Incid O l /\ Incid P l /\ Incid Q l) /\
~ (exists l, Incid O l /\ Incid P l /\ Incid R l) /\
~ (exists l, Incid O l /\ Incid Q l /\ Incid R l) /\
~ (exists l, Incid P l /\ Incid Q l /\ Incid R l) /\
~ (exists l, Incid P' l /\ Incid Q' l /\ Incid R' l) /\
((P<>P') \/ (Q<>Q') \/ (R<>R')) ->
(exists l, Incid alpha l /\ Incid beta l /\ Incid gamma l).
```

TABLE I.1.20 – Énoncé Coq du théorème de Desargues exprimé uniquement en incidence.

Pour simplifier l'expression de ce théorème et sa résolution, nous introduisons la colinéarité. L'énoncé Coq de la Table I.1.20 se simplifie en I.1.21. Notons qu'il n'est pas pratique d'exprimer le théorème de Desargues uniquement avec la définition `collinear`. L'ajout de la définition `on_line` permet d'éviter le traitement de certains quantificateurs existentiels en donnant directement le témoin.

```
(* Théorème de Desargues exprimé avec la colinéarité *)
(* Avec les axiomes de la géométrie d'incidence projective *)

Definition on_line A B C l := Incid A l /\ Incid B l /\ Incid C l.

Definition collinear A B C :=  exists l, Incid A l /\ Incid B l /\ Incid C l.

Theorem Desargues :
forall O P Q R P' Q' R' alpha beta gamma lP lQ lR lPQ lPR lQR lP'Q' lP'R' lQ'R',
((on_line P Q gamma lPQ) /\ (on_line P' Q' gamma lP'Q')) /\
((on_line P R beta lPR) /\ (on_line P' R' beta lP'R')) /\
((on_line Q R alpha lQR) /\ (on_line Q' R' alpha lQ'R')) /\
((on_line O P P' lP) /\ (on_line O Q Q' lQ) /\ (on_line O R R' lR)) /\
~collinear O P Q /\  ~collinear O P R /\ ~collinear O Q R /\
~collinear P Q R /\ ~collinear P' Q' R' /\
((P<>P') \/ (Q<>Q') \/ (R<>R')) ->
collinear alpha beta gamma.
```

TABLE I.1.21 – Énoncé Coq du théorème de Desargues exprimé avec la colinéarité.

4.2.2 Expressivité dans une théorie uniforme

Bien que l'énoncé de ce théorème soit satisfaisante, la méthode de résolution devient quant à elle plus complexe en introduisant la notion de colinéarité. Nous souhaitons conserver l'intuition géométrique apportée par la relation de colinéarité et sa généralisation en dimension supérieure afin de faciliter l'expression des théorèmes avec l'approche géométrique. Néanmoins, il n'est pas souhaitable de complexifier l'automatisation des preuves en ajoutant ces notions de colinéarité, coplanarité, etc.

Une idée que l'on retrouve dans les géométries combinatoires [Bat97, Mao11] consiste à introduire la notion de « rang » d'un espace projectif pour unifier le concept d'incidence et ces relations. Il est alors possible d'exprimer de manière homogène les relations usuelles de la géométrie d'incidence projective, non seulement les relations d'incidence qui sont nombreuses (point-droite, point-plan, droite-plan, point-espace ...) mais aussi les relations d'égalité, de colinéarité ou de coplanarité. Sans formaliser pour le moment cette notion de rang, nous posons que la fonction *rang* est égale à la dimension de l'espace projectif plus un :

- Rang d'un espace projectif vide : par convention, $\text{rang}(\emptyset) = 0$
- Rang d'un espace projectif E représentant un unique point : $\text{rang}(E) = 1$
- Rang d'un espace projectif E représentant une droite : $\text{rang}(E) = 2$
- Rang d'un espace projectif E représentant un plan : $\text{rang}(E) = 3$
- Rang d'un espace projectif E représentant un espace : $\text{rang}(E) = 4$

- ...

Notons que le rang d'un ensemble de points est égal au rang de l'espace affine engendré par ces points. Ainsi si deux points distincts définissent une droite, le rang de ces deux points est étendu à tout l'espace engendré par ces deux points c'est à dire la droite toute entière. En utilisant cette notion de rang, nous exprimons une nouvelle fois le théorème de Desargues en manipulant uniquement les points. Le résultat obtenu est présenté dans la Table I.1.22. L'unique fonction de rang suffit à exprimer l'intégralité des relations d'incidences contenues dans ce problème. En définissant les règles de simplifications et les lemmes intermédiaires pour ce concept de rang, il est maintenant envisageable d'établir un mécanisme de résolution homogène plus simple pour les problèmes de géométrie d'incidence projective. Pour cela, nous devons décrire formellement la notion de rang en l'axiomatisant. Nous abordons la formalisation de ce concept dans le Chapitre suivant I.2 grâce à la théorie des matroïdes.

```
(* Théorème de Desargues exprimé en utilisant les rangs *)
(* Avec un système d'axiomes conçu autour de la notion de rang *)

(* Fonction qui prend en paramètre un ensemble de points et qui renvoie un entier *)
rang : set Point -> nat

Theorem Desargues :
forall O P Q R P' Q' R' alpha beta gamma,
rang(P Q gamma) = 2 /\ rang(P' Q' gamma) = 2 /\
rang(P R alpha) = 2 /\ rang(P' R' alpha) = 2 /\
rang(Q R beta) = 2 /\ rang(Q' R' beta) = 2 /\
rang(O P P') = 2 /\ rang(O Q Q') = 2 /\ rang(O R R') = 2 /\
rang(O P Q) = 3 /\ rang(O P R) = 3 /\ rang(O Q R) = 3 /\
rang(P Q R) = 3 /\ rang(P' Q' R') = 3 /\
( rang(P P') = 2 \/ rang(Q Q') = 2 \/ rang(R R') = 2 ) ->
rang(alpha beta gamma) = 2.
```

TABLE I.1.22 – Énoncé Coq du théorème de Desargues exprimé avec le concept de rang.

CHAPITRE I.2

La théorie des matroïdes : une approche combinatoire cryptomorphique

"Logic is justly considered the basis of all other sciences, even if only for the reason that in every argument we employ concepts taken from the field of logic, and that ever correct inference proceeds in accordance with its laws."

Alfred Tarski (1901–1983)

Résumé

La formalisation de la géométrie d'incidence projective avec l'approche classique présente quelques limites au niveau de l'expressivité, ce qui rend difficile d'envisager une aide à la preuve systématique et une automatisation des preuves qui soit complète et efficace en toutes dimensions. Nous apportons une solution élégante dans ce chapitre, en suivant les travaux de Dominique Michelucci et al. [MS04, MS06], qui décrit comment l'approche matroïdale permet de capturer et de généraliser les problèmes d'indépendance linéaire que l'on retrouve dans les problèmes géométriques. Cette approche cryptomorphique [1] combinatoire permet de caractériser formellement la notion de rang présentée précédemment.

Nous commençons par introduire la théorie des matroïdes, plus précisément un fragment de cette dernière, permettant de formaliser la géométrie d'incidence projective (section 1). Pour valider cette approche et permettre une traduction bilatérale entre les deux théories, nous étudions la preuve d'équivalence entre les deux théories en dimension quelconque (section 2).

Contenu

1. En mathématiques, deux objets et plus spécialement deux systèmes d'axiomes ou leurs sémantiques sont dit cryptomorphes s'ils sont équivalents mais pas de manière évidente.

La notion de rang introduite dans le Chapitre I.1 permet d'exprimer avec efficacité et simplicité toutes les relations d'incidences qui peuvent se présenter dans notre cadre géométrique. Ce changement d'approche permet d'envisager une aide à la preuve plus adaptée en ne manipulant qu'un concept à la fois sans pour autant perdre en expressivité. Nous décrivons dans la suite de cette thèse la formalisation de ce concept en utilisant la théorie des matroïdes.

1 Approche combinatoire de la géométrie d'incidence projective

Pour décrire cette approche combinatoire alternative basée sur la notion de dimension, nous introduisons le concept de rang provenant de la théorie des matroïdes [GM12, Oxl06, Wel10].

1.1 Théorie des matroides

Les matroïdes ont été introduits dans les années 30 par Whitney dans ses travaux concernant son approche axiomatique de la dépendance linéaire et algébrique. Comme le nom le suggère, Whitney décrit les matroïdes comme une généralisation abstraite de la notion d'indépendance des colonnes d'une matrice. Les matroïdes fournissent un traitement unificateur de la dépendance en algèbre linéaire et en théorie des graphes. Autrement dit, c'est une structure combinatoire qui permet de capturer et généraliser les principales propriétés ensemblistes de la dépendance linéaire dans les espaces vectoriels. Depuis, il a été reconnu que les matroïdes apparaissent naturellement dans l'optimisation combinatoire et qu'ils peuvent être utilisés comme cadre pour aborder une étonnante diversité de problèmes. Ainsi cette notion apparaît naturellement dans la théorie des graphes, des corps, des algorithmes gloutons ou de la topologie. Il existe plusieurs définitions cryptomorphiques des matroïdes basé sur les concepts d'ensemble des indépendants, des circuits, des bases, des rangs et des plats. Nous décrivons dans un premier temps l'axiomatisation la plus fréquente de la théorie des matroïdes définissant la notion d'objets indépendants. Puis, nous détaillons les systèmes d'axiomes équivalents fondés sur les plats et les rangs que nous utilisons plus spécifiquement pour décrire les problèmes géométriques.

1.1.1 Les matroïdes pour caractériser la notion d'indépendance

En termes d'indépendance, un matroïde M est un couple (E, ξ) où E est un ensemble fini et ξ est une famille de sous-ensembles de E qui vérifie les conditions suivantes :

(P1I) $\emptyset \in \xi$

(P2I) Si $E_1 \subseteq E_2$ et $E_2 \in \xi$, alors $E_1 \in \xi$

(P3I) Si $E_1, E_2 \in \xi$ et $\left|E_1\right| < \left|E_2\right|$, alors $\exists\, e \in E_2 \setminus E_1$ tel que $E_1 \cup \{e\} \in \xi$

TABLE I.2.1 – Axiomatisation des matroïdes avec les indépendants.

Les sous-ensembles de ξ sont appelés les indépendants de M. Un sous-ensemble de E qui n'est pas dans ξ est dit dépendant. L'axiome **(P1I)** indique que l'ensemble vide est indépendant. C'est-à-dire qu'il existe toujours au moins un ensemble de E qui est indépendant $\xi \neq \emptyset$. Le deuxième **(P2I)** précise que les sous-ensembles d'un ensemble indépendant sont indépendants, c'est la propriété d'hérédité. Pour finir l'axiome de l'échange **(P3I)** dit que : Si les ensembles U et V sont indépendants, et si le cardinal de V est strictement plus grand que celui de U, alors il est possible de compléter U avec un élément $v \in V \setminus U$ tel que $U \cup \{v\}$ reste indépendant.

1.1.2 Les matroïdes pour caractériser la notion de rang

Soit $M = (E, \xi)$ et $X \subseteq E$. Le rang de X, noté $rk(X)$, est le cardinal d'un indépendant contenu dans X et maximal pour l'inclusion, c'est à dire :

$$\boxed{rk(X) = max\{|Y| : Y \subseteq X, Y \in \xi\}}$$

L'axiome de l'échange **(P3I)** permet de dire que cette définition est correcte. Inversement, on a que la fonction rk est la fonction rang d'un matroïde (E, ξ) avec E un ensemble fini si et seulement si rk vérifie les conditions suivantes :

(P1R) $\forall\ X \subseteq E,\ 0 \leq rk(X) \leq |X|$

(P2R) $\forall\ X \subseteq Y \subseteq E,\ rk(X) \leq rk(Y)$

(P3R) $\forall\ X, Y \subseteq E,\ rk(X \cup Y) + rk(X \cap Y) \leq rk(X) + rk(Y)$

TABLE I.2.2 – Axiomatisation des matroïdes avec les rangs.

L'ensemble des indépendants ξ de M est alors l'ensemble des parties A de E telles que : $rk(A) = |A|$. De plus, l'axiome **(P1R)** indique que le rang est une fonction entière dont la valeur est toujours positive et inférieure ou égale au cardinal de l'ensemble. L'axiome **(P2R)** précise que cette fonction est non décroissante. Le dernier axiome **(P3R)** est appelée propriété de sous-modularité.

1.1.3 Les matroïdes pour caractériser la notion de fermeture et de plat

Soit M un matroïde défini sur un ensemble fini E. La fermeture $cl(X)$ d'un sous-ensemble $X \subseteq E$ est l'ensemble :

$$\boxed{cl(X) = \{x \in E \mid rk(X) = rk(X \cup x)\}}$$

Inversement, on a que la notion de fermeture est complétement caractérisée si et seulement si les propriétés suivantes sont satisfaites :

(P1F) $\forall\ X \subseteq E,\ X \subseteq cl(X)$

(P2F) $\forall\ X \subseteq E,\ cl(X) = cl(cl(X))$

(P3F) $\forall\ X\ Y \subseteq E$, avec $X \subseteq Y$, $cl(X) \subseteq cl(Y)$

(P4F) $\forall\ a\ b \in E, \forall\ X \subseteq E$, si $a \in (cl(Y \cup \{b\}) \setminus cl(Y))$ alors $b \in (cl(Y \cup \{a\}) \setminus cl(Y))$

TABLE I.2.3 – Axiomatisation des matroïdes par la fermeture.

Les deux premières propriétés **(P1F)** **(P2F)** assurent l'extensitivité et l'idempotence. L'axiome **(P3F)** indique que la fonction de fermeture est croissante. Enfin **(P4F)** est un axiome d'échange connu sous le nom de propriété d'échange de Mac Lane-Steinitz [Oxl06, Wel10].

Un ensemble égal à sa fermeture est appelé ensemble fermé ou plat. Soit M un matroïde défini sur un ensemble fini E et soit $\mathcal{F}$ une famille de sous-ensembles de E, cette famille représente les plats d'un matroïde si et seulement si les propriétés suivantes sont vérifiés :

(P1P) $E \in \mathcal{F}$

(P2P) Si F_1, $F_2 \in \mathcal{F}$, alors $F_1 \cap F_2 \in \mathcal{F}$

(P3P) Si $F \in \mathcal{F}$ et $\{F_1, F_2, \ldots, F_k\}$ représente l'ensemble des plats qui couvrent F, alors $\{F_1 \setminus F, F_2 \setminus F, \ldots, F_k \setminus F\}$ partitionne $E \setminus F$

TABLE I.2.4 – Propriétés sur les plats.

D'autre part, un ensemble est un plat s'il est maximal pour son rang, ce qui signifie que l'ajout de tout autre élément à l'ensemble augmenterait le rang. En d'autres termes, le rang d'un plat E est le cardinal du plus petit ensemble nécessaire pour engendrer E. Dans la Table I.2.5, nous donnons quelques exemples de rang de plats en géométrie projective.

$rk\{A, B\} = 1$	$A = B$
$rk\{A, B\} = 2$	$A \neq B$
$rk\{A, B, C\} = 2$	A, B, C sont colinéaires avec au moins deux points distincts
$rk\{A, B, C\} \leq 2$	A, B, C sont colinéaires
$rk\{A, B, C\} = 3$	A, B, C ne sont pas colinéaires
$rk\{A, B, C, D\} = 3$	A, B, C, D sont coplanaires mais pas tous collinéaires
$rk\{A, B, C, D\} = 4$	A, B, C, D ne sont pas coplanaires

TABLE I.2.5 – Illustration de la notion de rang sur des ensembles de points.

En utilisant chacune de ces définitions, on peut montrer que tout espace projectif a une structure matroïdale, la réciproque étant fausse. Pour capturer l'intégralité de la géométrie projective d'incidence avec la notion de rang, nous avons besoin d'introduire des axiomes supplémentaires à ceux des matroïdes de la Table I.2.2.

1.2 Système d'axiomes fondé sur la notion de rang en 2D

La Table I.2.6 présente les propriétés matroïdales de la fonction rk qui sont communes à tous les espaces projectifs de dimension N.

(A1R2-R3) **Rk-SubCardinal** : $\forall\ X \subseteq E,\ 0 \leq rk(X) \leq |X|$

(A2R2-R3) **Rk-NonDecreasing** : $\forall\ X \subseteq Y \subseteq E,\ rk(X) \leq rk(Y)$

(A3R2-R3) **Rk-SubModular** : $\forall\ X, Y \subseteq E,\ rk(X \cup Y) + rk(X \cap Y) \leq rk(X) + rk(Y)$

TABLE I.2.6 – Système d'axiomes basé sur les rangs pour la géométrie projective nD.

Nous présentons ensuite dans la Table I.2.7 le système d'axiomes basé sur les rangs pour décrire la géométrie projective plane. Les deux premiers axiomes **(A4R2)** et **(A5R2)** que l'on retrouve en dimension quelconque établissent la non dégénérescence de la fonction de rang. Les autres axiomes **(A6R2)** **(A7R2)** **(A8R2)** sont plus ou moins une traduction directe des axiomes de la géométrie projective plane de la Table I.1.2.

(A4R2) Rk-Singleton : $\forall\ P : Point,\ rk\{P\} \geq 1$

(A5R2) Rk-Couple : $\forall\ P\ Q : Point,\ P \neq Q \Rightarrow rk\{P, Q\} \geq 2$

(A6R2) Rk-Inter : $\forall\ A\ B\ C\ D : Point, \exists\ J : Point,\ rk\{A, B, J\} = rk\{C, D, J\} = 2$

(A7R2) Rk-Three-Points : $\forall\ A\ B : Point, \exists\ C, rk\{A, B, C\} = rk\{B, C\} = rk\{A, C\} = 2$

(A8R2) Rk-Lower-Dimension : $\exists\ A\ B\ C : Point,\ rk\{A, B, C\} \geq 3$

TABLE I.2.7 – Système d'axiomes basé sur les rangs pour la géométrie projective 2D.

1.3 Système d'axiomes fondé sur la notion de rang en ≥3D et 3D

De la même manière, nous définissons un système d'axiomes basé sur les rangs pour décrire l'espace projectif en dimension 3 et plus dans la Table I.2.8. Une nouvelle fois, nous modifions seulement les axiomes de *Pasch* et *Lower-Dimension* pour capturer au moins une géométrie projective spatiale (≥3). Pour limiter cette axiomatisation à la 3D, nous devons ajouter l'axiome **(A9R3')** ou l'axiome **(A9R3)** qui sont deux versions équivalentes plus ou moins directes de l'axiome **(A6P3)** de la géométrie synthétique.

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

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

(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-Dim : $\exists\ A\ B\ C\ D : Point,\ rk\{A, B, C, D\} \geq 4$

TABLE I.2.8 – Système d'axiomes basé sur les rangs pour la géométrie projective ≥3D.

(A9R3) Rk-Upper-Dim : $\forall\ A\ B\ C\ D\ E\ F, \exists\ J1\ J2\ J3$
$rk\{A, B\} = 2 \wedge rk\{C, D\} = 2 \wedge rk\{E, F\} = 2 \wedge$
$rk\{A, B, C, D\} \geq 3 \wedge$
$rk\{A, B, E, F\} \geq 3 \wedge$
$rk\{C, D, E, F\} \geq 3 \Rightarrow$
$rk\{A, B, J1\} = 2 \wedge rk\{C, D, J2\} = 2 \wedge$
$rk\{E, F, J3\} = 2 \wedge rk\{J1, J2, J3\} \leq 2$

(A9R3') **Rk-Upper-Dim** : $\forall\ A\ B\ C\ D\ E, rk\{A, B, C, D, E\} \leq 4$

TABLE I.2.9 – Système d'axiomes basé sur les rangs pour la géométrie projective 3D.

1.4 Formalisation Coq

L'implantation Coq et le découpage de ces systèmes d'axiomes suit exactement le même processus que pour la géométrie synthétique : nous utilisons des classes de types pour augmenter la modularité de notre développement en fonction de la dimension. Pour représenter les ensembles de points en Coq, nous utilisons la bibliothèque standard *Containers* [Les11]. Elle permet de raisonner de manière abstraite sur les ensembles ou de manipuler des implantations comme les listes ou les AVL[2]. Étant donné que nous travaillons avec des ensembles de petite taille, nous privilégions l'implantation par listes. Pour des questions de performance dans l'environnement Coq, nous prouvons certains résultats théoriques avec les listes plutôt qu'avec les ensembles abstraits. L'usage des ensembles abstraits pour effectuer des calculs d'inclusion, d'intersection ou d'union peut en effet être long principalement à cause du mécanisme d'unification de Coq et de la vérification des types[3].

L'algorithme d'unification joue un rôle central dans un assistant de preuve tel que Coq [Zil14]. En effet, il infère les termes implicites et les annotations manquantes tout en s'occupant de la bonne application des lemmes. L'unification est ainsi chargée de mettre en relation et de comparer le type des arguments de la fonction avec le type des éléments qui sont passés en paramètre à cette fonction. Par exemple, la tactique `apply` a pour objectif d'unifier le but courant avec la conclusion du lemme appliqué.

Nous rapportons dans les Tables I.2.10 et I.2.11 la hiérarchie des classes de types commençant la description de la géométrie d'incidence projective avec les rangs. La formalisation Coq complète est disponible dans l'Annexe D.

```
(* Types *)
Class MatroidRk '(S : FSetSpecs Point) := {

Point : Set;
rk : set Point -> nat

}.
```

TABLE I.2.10 – Classe de types pour la structure projective basé sur les rangs.

2. Arbres binaires de recherche automatiquement équilibrés

3. Une optimisation plus poussée de la structure de données manipulée permettrait d'améliorer les performances globales de nos méthodes d'automatisation. Nous pourrions par exemple stocker les ensembles sous forme d'entier binaire en nous appuyant sur l'article suivant [BDL16].

```
(* Structure matroïdale *)
Class Matroid '(MR : MatroidRk) := {

(* A1R2-R3 Rk-SubCardinal *)
matroid1  : forall X, rk X >= 0 /\ rk X <= cardinal X;

(* A2R2-R3 Rk-NonDecreasing *)
matroid2 : forall X Y, Subset X Y -> rk X <= rk Y;

(* A3R2-R3 Rk-SubModular *)
matroid3 : forall X Y, rk(union X Y) + rk(inter X Y) <= rk X + rk Y

}.
```

TABLE I.2.11 – Classe de types pour la structure matroïdale.

2 Deux approches cryptomorphiques

Maintenant que nous avons précisé l'axiomatisation de la géométrie d'incidence projective basée sur les rangs, nous nous intéressons à la preuve de l'équivalence entre ces deux systèmes d'axiomes : d'un côté, nous avons l'approche par la géométrie synthétique qui permet de décrire de manière intuitive des problèmes géométriques, de l'autre nous avons une approche combinatoire utilisant le concept de rang introduit par les matroïdes qui laisse envisager une automatisation plus efficace au détriment de la lisibilité des preuves.

Dans des travaux précédents [MNS09, MNS12], seule l'implication des rangs vers la géométrie projective ≥3D a été étudiée. Cette implication était suffisante pour faire une étude de cas sur la preuve du théorème de Desargues en 3D avec les rangs. Afin d'augmenter les possibilités d'automatisations dans les preuves et permettre une traduction bidirectionnelle entre ces deux approches, nous effectuons la preuve complète de cette équivalence. Avant d'établir cette preuve, nous commençons par définir la bi-interprétation en introduisant les deux dictionnaires I.2.12 et I.2.13 qui permettent d'effectuer une traduction des objets et relations d'une théorie vers l'autre. La conception de ces dictionnaires s'inspire de la notion de plat introduite précédemment I.2.5. Grâce à cette double traduction, nous avons le théorème d'équivalence suivant :

Théorème I.2.1. *Les systèmes de la géométrie d'incidence projective avec l'approche en géométrie synthétique et ceux basés sur les rangs sont équivalents respectivement en 2D, ≥3D et 3D.*

Pour parvenir à cette preuve, nous divisons la démonstration de l'équivalence en fonction de la dimension et de la direction.

2.1 Des rangs vers la géométrie synthétique

Nous commençons par l'implication des rangs vers la géométrie d'incidence projective en 2D, ≥3D et 3D. Les preuves sont très similaires quelle que soit la dimension.

2.1.1 Préliminaires

Pour commencer, nous devons caractériser le concept de droite et d'incidence qui n'existe pas dans le système d'axiomes fondé sur les rangs I.2.12. Nous construisons, en utilisant une définition inductive, une droite à partir de deux points distincts. Un point P est incident à la droite l, si le rang du triplet formé par les deux points engendrant la droite et le point P reste égal à 2. Finalement, deux droites l et m sont égales, si le rang du quadruplet formé par les deux points engendrant chaque droite reste égal à 2.

```
(* Caractérisation de la géométrie synthétique à partir des rangs *)

Definition Point := Point.

Inductive LineInd : Type :=
|Cline : forall (A B : Point)(H : ~ A[==]B), LineInd.

Definition Line := LineInd.

Definition Incid (P : point)(l : Line) := rk ((fstP l)(sndP l) P) = 2.

Definition line_eq (l m : Line) := rk((fstP l)(sndP l)(fstP m)(sndP m)) = 2.
```

TABLE I.2.12 – Caractérisation de la géométrie synthétique à partir de la notion de rang.

À partir de maintenant, nous pouvons exprimer les énoncés de la géométrie d'incidence projective classique et les prouver en utilisant les axiomes de rangs.

2.1.2 Sous-Modularité

Nous détaillons quelques techniques de preuves utilisées pour démontrer les cinq axiomes de la géométrie synthétique à la fois en 2D, ≥3D et 3D.

Premièrement, nous prouvons généralement les égalités entre rangs ($rk(a) = rk(b)$) en deux étapes : en premier $rk(a) \leq rk(b)$, en second $rk(a) \geq rk(b)$. Deuxièmement, nous travaillons de manière systématique avec l'axiome de la sous-modularité **(A3R2-R3)**. Déterminer les intersections entre deux ensembles finis de points est un problème, nous devons distinguer tous les cas possibles d'égalités entre points. La preuve résultante devient plus complexe avec ces distinctions. C'est pourquoi, nous définissons un lemme particulier dérivé de l'axiome **(A3R2-R3)** qui ne considère pas l'intersection théorique mais une approximation plus faible de cette intersection (notée dans la suite $\sqcap$).

Définition (Intersection théorique)**.** *Soient L_1 et L_2 deux ensembles de points. Par définition $L_1 \cap L_2$ est l'intersection exacte entre les deux ensembles de points considérés. Tous les points qui sont syntaxiquement identiques ou qui sont égaux par construction apparaissent une seule et unique fois dans l'ensemble résultat.*

Par exemple, soit $L_1 = \{A, C\}$ et $L_2 = \{B, C\}$ deux ensembles de points. Nous ajoutons l'hypothèse que $A = B$, alors il est possible de déduire que l'intersection théorique de ces deux ensembles est $L_1 \cap L_2 = \{A, C\} = \{B, C\}$.

Définition (Intersection littérale)**.** *Soient L_1 et L_2 deux ensembles de points. Par définition $L_1 \sqcap L_2$ est l'intersection syntaxique entre les deux ensembles de points considérés. Tous les points qui sont syntaxiquement identiques apparaissent une seule et unique fois dans l'ensemble résultat. En pratique, nous savons que : $L_1 \sqcap L_2 \subseteq L_1 \cap L_2$. En utilisant **(A2R2-R3)**, nous déduisons donc que : $rk(L_1 \sqcap L_2) \leq rk(L_1 \cap L_2)$.*

Par exemple, soient $L_1 = \{A, C\}$ et $L_2 = \{B, C\}$ deux ensembles de points. Avec l'hypothèse que $A = B$, alors l'intersection littérale entre ces deux ensembles de points est $L_1 \sqcap L_2 = \{C\}$.

À partir de cette intersection littérale, nous pouvons déduire une version plus appropriée de l'axiome **(A3R2-R3)** en ignorant les cas d'égalités entre points :

Lemme I.2.1 (A3R2-R3-lit)**.** $\forall\ X\ Y,$
$$rk(X \cup Y) + rk(X \sqcap Y) \leq rk(X) + rk(Y).$$

Cependant, le système Coq ne permet pas de facilement définir la notion d'intersection littérale de manière calculatoire. Nous préférons définir une version alternative qui capture le sens du lemme I.2.1 :

Lemme I.2.2 (A3R2-R3-alt)**.** $\forall\ X\ Y\ I,$
$$I \subseteq (X \sqcap Y) \Rightarrow rk(X \cup Y) + rk(I) \leq rk(X) + rk(Y).$$

Cette version fonctionnelle de la sous-modularité est beaucoup utilisée dans chacune des preuves. Contrairement à l'intersection, il n'est pas nécessaire de définir une union théorique et une union littérale puisqu'elles capturent toujours le même ensemble de points indépendamment des égalités entre points.

Troisièmement, la connaissance du rang d'un ensemble de points est bien souvent incomplète. Pour représenter sa dimension, nous devons donc définir un intervalle d'entiers en l'absence d'informations supplémentaires ou de la mise en évidence d'une contradiction. En d'autres termes, le rang d'un ensemble de points n'est pas toujours déterminé, il n'est pas assez contraint par l'énoncé ou par la preuve, le système a besoin de plus d'informations ou d'une contradiction pour continuer. Dans la plupart des cas, la seule méthode possible pour gérer cela est de faire un raisonnement cas par cas pour chacun des rangs possibles de l'ensemble. La démonstration de l'axiome *Upper-Dimension* est un exemple qui illustre parfaitement ce schéma de preuve. Les trois droites qui sont coupées par une quatrième doivent avoir pour unique contrainte d'être différentes deux à deux. Pour réaliser cette démonstration, il est obligatoire de distinguer si les couples de droites sont coplanaires ou non.

2.1.3 Preuve de la propriété *Uniqueness*

Pour illustrer les mécanismes précédents de preuves, nous détaillons la démonstration de la propriété *Uniqueness* en partant des rangs.

Lemme I.2.3 (A3P2-P3)**.** *Uniqueness,* $\forall\ A\ B : Point, \forall\ l\ m : Line,$
$A \in l \wedge B \in l \wedge A \in m \wedge B \in m \Rightarrow A = B \vee l = m$

Démonstration. Nous commençons la preuve par une distinction de cas sur l'égalité entre les points A et B :

- Si $A = B$, alors la conclusion est triviale.
- Si $A \neq B$, nous déplions les définitions de `Line` et `Incid`, il s'ensuit que :

$$\left.\begin{array}{l} \text{Nous avons } A <> B \\ \text{Soit } P \in l,\ Q \in l \text{ and } P <> Q \\ \text{Soit } R \in m,\ S \in m \text{ and } R <> S \\ \text{Incid } A\ l \ \Rightarrow rk\{P, Q, A\} = 2 \\ \text{Incid } B\ l \ \Rightarrow rk\{P, Q, B\} = 2 \\ \text{Incid } A\ m \Rightarrow rk\{R, S, A\} = 2 \\ \text{Incid } B\ m \Rightarrow rk\{R, S, B\} = 2 \end{array}\right\} \textit{Assumptions}$$

$$\hline$$

$$l = m \Rightarrow rk\{P, Q, R, S\} = 2 \qquad \} \textit{Goal}$$

Pour continuer, nous déterminons le rang des deux ensembles suivants en utilisant A3R2-R3-alt : $rk\{P, Q, A, B\}$ et $rk\{R, S, A, B\}$.

$$\begin{array}{l} \quad rk(\{P, Q, A\} \cup \{P, Q, B\}) + rk(\{P, Q, A\} \cap \{P, Q, B\}) \\ \leq rk\{P, Q, A\} + rk\{P, Q, B\} \\ \Rightarrow rk\{P, Q, A, B\} + rk\{P, Q\} \leq rk\{P, Q, A\} + rk\{P, Q, B\} \\ \Rightarrow rk\{P, Q, A, B\} \leq 2 \end{array}$$

De manière analogue, nous calculons que $rk\{R, S, A, B\} \leq 2$. Puis nous établissons que : $rk\{P, Q, R, S, A, B\} \leq 2$.

$$\begin{array}{l} \quad rk(\{P, Q, A, B\} \cup \{R, S, A, B\}) + rk(\{P, Q, A, B\} \cap \{R, S, A, B\}) \\ \leq rk\{P, Q, A, B\} + rk\{R, S, A, B\} \\ \Rightarrow rk\{P, Q, R, S, A, B\} + rk\{A, B\} \leq rk\{P, Q, A, B\} + rk\{R, S, A, B\} \\ \Rightarrow rk\{P, Q, R, S, A, B\} \leq 2 \end{array}$$

En utilisant A2R2-R3, nous prouvons que :

$$\{P, Q, R, S\} \subset \{P, Q, R, S, A, B\} \Rightarrow rk\{P, Q, R, S\} \leq rk\{P, Q, R, S, A, B\}$$

Donc $rk\{P, Q, R, S\} \leq 2$ et comme l'ensemble $\{P, Q, R, S\}$ contient au moins deux points distincts, nous concluons que $rk\{P, Q, R, S\} = 2$. □

2.1.4 Implantation Coq

Au niveau de l'implantation, pour effectuer les déductions sur les ensembles abstraits, nous utilisons la tactique `fsetdecide` fournie par la librairie Coq [Les11]. Cette tactique implémente une procédure de décision pour les propositions concernant des ensembles finis abstraits. D'autre part, nous utilisons le type `setoid` pour effectuer des substitutions en indiquant que les ensembles sont identiques. Les setoïdes en Coq permettent de déclarer une nouvelle relation d'équivalence qui peut être utilisée dans certaines tactiques et notamment pendant les réécritures. Quand on peut prouver que deux termes sont équivalents mais pas nécessairement égaux, il est possible de

remplacer l'un des termes par l'autre en utilisant les setoïdes. Nous associons ainsi notre égalité paramétrique qui est une relation d'équivalence pour effectuer des réécritures généralisées dans les différentes parties de la preuve.

2.2 De la géométrie synthétique vers les rangs

L'implication dans l'autre direction est plus difficile à établir puisque nous devons spécifier le concept de rang du point de vue de la géométrie projective. Rappelons que la fonction de rang s'applique à tous les objets de la géométrie d'incidence projective possibles mais aussi à toutes les incidences : l'égalité entre points, la colinéarité ou la coplanarité. Pour prouver cette implication, nous utilisons l'implantation reposant sur les listes plutôt que la structure abstraite pour les raisons de performance mentionnées précédemment.

2.2.1 Préliminaires

Pour définir le rang, il est nécessaire de concevoir plusieurs concepts intermédiaires représentant les différentes valeurs de retour de la fonction de rang. Pour spécifier le rang (voir Table I.2.13), nous nous inspirons de la caractérisation des matroïdes sur les plats (voir Table I.2.5). Un ensemble de points est vide, soit il définit un point, une droite, un plan ou un espace.

```
(* Caractérisation de la notion de rang à partir de la géométrie synthétique *)

Definition rkl s := match s with
| nil => 0
| x :: nil => 1
| s => if contains_four_non_coplanar_points s then 4 else
          if contains_three_non_collinear_points s then 3 else
             if contains_two_distinct_points s then 2 else 1 end.
```

TABLE I.2.13 – Caractérisation de la fonction de rang à partir de la géométrie d'incidence projective.

Les trois prédicats présentés dans la Table I.2.13 `contains_four_non_coplanar_points`, `contains_three_non_collinear_points` et `contains_two_distincts_points` (voir l'Annexe E) définissent les limites sur la dimension de l'ensemble considéré. Le système vérifie en premier la coplanarité : soit il existe dans l'ensemble un quadruplet de points non coplanaires et la dimension est donc un espace, ou alors le système continue en analysant la colinéarité. Ces prédicats forment ce qu'on appelle une couche intermédiaire. Ils aident à effectuer la transition entre les deux systèmes d'axiomes et sont accompagnés par de nombreux lemmes. Notons qu'il faut rester vigilant lors du développement de ces définitions pour prendre en compte tous les cas dégénérés possibles (principalement des points qui coïncident).

Le prédicat récursif `contains_four_non_coplanar_points` de la Table I.2.14 étudie si un ensemble contient 4 points qui ne sont pas coplanaires. Pour cela, nous définissons la fonction `coplanar_with_all` dans la Table I.2.15 qui teste si le premier point est coplanaire avec tous les autres. La définition `all_triples` permet de construire tous les triplets de points possibles en évitant les permutations (voir l'Annexe E).

```
(* Test d'existence d'une non coplanarité dans la liste l *)
Fixpoint contains_four_non_coplanar_points l := match l with
| nil => false
| a :: r => if coplanar_with_all a (all_triples r)
              then contains_four_non_coplanar_points r else true
end.
```

TABLE I.2.14 – Définition récursive du prédicat `contains_four`.

```
(* Test de coplanarité entre le point a et la liste l *)
Fixpoint coplanar_with_all a l :=
match l with
| nil => true
| (b,c,d) :: r => if coplanar a b c d then coplanar_with_all a r else false
end.
```

TABLE I.2.15 – Définition récursive du prédicat `coplanar_with_all`.

2.2.2 Techniques de preuve

Maintenant que la notion de rang est caractérisée, nous présentons de manière similaire les principaux points techniques que nous avons rencontrés lors de la démonstration des 9 axiomes sur les rangs du point de vue de la géométrie synthétique. Nous exploitons principalement le mécanisme d'induction sur les entiers pour effectuer une analyse de cas sur le rang de chaque ensemble de points. La manipulation des trois conditionnelles dans la caractérisation de la fonction de rang I.2.13 et la gestion des cas dégénérés multiplient le nombre de buts qui doivent être vérifiés augmentant significativement la taille d'une démonstration. Les preuves résultantes sont souvent laborieuses et répétitives bien qu'il soit possible d'automatiser de nombreuses étapes. Dans ce but, nous concevons plusieurs tactiques en utilisant *Ltac* le langage de tactique fourni par Coq et permettant à l'utilisateur d'écrire ses propres schémas de preuves.

Pour faire cela, nous identifions des motifs spécifiques parmi les hypothèses et le but que nous pouvons simplifier par une séquence de tactiques et de lemmes, l'objectif visé étant de traiter toutes les configurations possibles. Cela nécessite une bonne stratégie d'application des simplifications, une optimisation de la profondeur de recherche et quelques essais pour résoudre un problème. Si une tactique simplifie trop rapidement ou trop profondément, elle ne sera pas en mesure de prouver le résultat, dans le cas contraire l'arbre de recherche peut devenir rapidement très grand et considérablement ralentir l'utilisation de telles tactiques.

Les tactiques comme celle décrite dans la Table I.2.16 nous permettent de déplier le plus possible le but tout en introduisant un maximum d'hypothèses dans le contexte. Si le but est devenu trivial en appliquant une simple contradiction, égalité ou inégalité, le travail est fini. Autrement, l'utilisateur doit rendre explicite la contradiction parmi les hypothèses en appliquant d'autres lemmes intermédiaires ou en effectuant des simplifications plus subtiles. La majorité de ces étapes habituellement réalisées à la main sont regroupées dans la tactique I.2.16.

```
(* Tactique de simplification pour la fonction rang *)
Ltac my_rank :=
  repeat match goal with
  |[H : _ |- _] => progress [intro|intros]
  |[H : _ |- _ <-> _] => split
  |[H : _ |- _ /\ _] => split
  |[H : _ /\ _ |- _] => destruct H
  |[H : _ |- _] => solve[intuition]
  |[H : _ |- _] => progress contradiction
  |[H : ?X :: _ = nil |- _] => inversion H
  |[H : false = true |- _] => inversion H
  |[H : true = false |- _] => inversion H
  |[H : ?X[==]?X -> False |- _] => apply False_ind;apply H;reflexivity

  |[H : _ |- (if if ?X then _ else _
                then _ else _) = _ \/ _] => case_eq X
  |[H : _ |- (if ?X then _ else _) = _ \/ _] => case_eq X
  |[H : _ |- (if if ?X then _ else _
                then _ else _) = _] => case_eq X
  |[H : _ |-  (if ?X then _ else _) = _] => case_eq X
  |[H : _ |- _ = (if if ?X then _ else _
                then _ else _)] => case_eq X
  |[H : _ |- _ = (if ?X then _ else _)] => case_eq X
  |[H : _ |- (if if ?X then _ else _
                then _ else _) >= _] => case_eq X
  |[H : _ |- (if ?X then _ else _) >= _ ] => case_eq X
  |[H : _ |- (if if ?X then _ else _
                then _ else _) <= _] => case_eq X
  |[H : _ |- (if ?X then _ else _) <= _ ] => case_eq X
  |[H : _ |- _ >= (if if ?X then _ else _
                then _ else _)] => case_eq X
  |[H : _ |- _ >= (if ?X then _ else _)] => case_eq X
  |[H : _ |- _ <= (if if ?X then _ else _
                then _ else _)] => case_eq X
  |[H : _ |- _ <=(if ?X then _ else _)] => case_eq X
  |[H : _ |- _ + (if if ?X then _ else _
                then _ else _) >= _] => case_eq X
  |[H : _ |- _ + (if ?X then _ else _) >= _ ] => case_eq X
  |[H : _ |- _ + (if if ?X then _ else _
                then _ else _) <= _] => case_eq X
  |[H : _ |- _ + (if ?X then _ else _) <= _ ] => case_eq X

  |[H : _ |- (if if ?X then _ else _
                then _ else _) + _ >= _] => case_eq X
  |[H : _ |- (if ?X then _ else _) + _ >= _ ] => case_eq X
  |[H : _ |- (if if ?X then _ else _
                then _ else _) + _ <= _] => case_eq X
  |[H : _ |- (if ?X then _ else _) + _ <= _ ] => case_eq X
  |[H : _ |- _ >= (if if ?X then _ else _
                then _ else _) + _] => case_eq X
  |[H : _ |- _ >= (if ?X then _ else _) + _] => case_eq X
  |[H : _ |- _ <= (if if ?X then _ else _
                then _ else _) + _] => case_eq X
  |[H : _ |- _ <=(if ?X then _ else _) + _] => case_eq X
  end.
```

Table I.2.16 – Tactique de simplification du contexte Coq impliquant la fonction de rang.

La Table I.2.16 décrit la tactique principale de décomposition, simplification et résolution des buts de la preuve de l'implication géométrie synthétique vers les rangs. Cette tactique récursive s'arrête soit si le but a été prouvé, si il n'existe plus aucune simplification à effectuer ou si le niveau de profondeur Coq par défaut associé à la tactique a été atteint. Ce niveau de profondeur permet de limiter l'arbre de recherche et de bloquer les tactiques qui se répéteraient indéfiniment. Le détail des commandes Coq est disponible dans [Coq02] :

- Avec la tactique `intro/intros`, nous introduisons dans le contexte les hypothèses soit en début de preuve ou soit après avoir déplié une définition ;
- Avec la tactique `split/destruct`, nous découpons respectivement le but et les hypothèses ;
- Avec la tactique `solve[intuition]`, nous regardons si le but peut être trivialement prouvé avec des simplifications mineures. Si le système n'arrive pas à résoudre le but courant, il annule toutes les modifications qui ont été testées ;
- Avec la tactique `progress[contradiction]`, nous regardons si une contradiction existe dans les hypothèses tout en conservant les modifications effectués par cette tactique.
- Avec la tactique `inversion`, nous mettons en contradiction des constructeurs différents ;
- Finalement, avec la tactique `case_eq X`, nous identifions des motifs de conditionnelles apparaissant dans la définition du prédicat `rkl` dans la Table I.2.13. Puis nous effectuons une distinction de cas sur la variable X créant ainsi deux sous-buts à prouver.

2.2.3 Preuve de la propriété matroïdale de non-décroissance

Nous illustrons notre méthodologie pour prouver les axiomes de la théorie des matroïdes à partir des axiomes de la géométrie synthétique en détaillant la démonstration de la propriété de non-décroissance.

Lemme I.2.4 (A2R2-R3)**.** *Rk-NonDecreasing,*
$\forall\ X \subseteq Y \subseteq E,\ rk(X) \leq rk(Y)$

La caractérisation I.2.13 indique que la fonction de rang peut prendre cinq valeurs différentes (de 0 à 4) en fonction de la dimension de l'ensemble de points considéré. Cette caractérisation mène à 25 cas différents lorsque nous effectuons une induction structurelle sur les variables X et Y qui sont des listes. Premier cas : $0 \leq 0$, deuxième cas : $0 \leq 1$, etc. En utilisant, la tactique I.2.17 qui calcule automatiquement l'inclusion entre listes, nous éliminons directement 15 cas où $X \subseteq Y$. Pour prouver les buts restants, nous mettons en évidence une contradiction au sein du contexte pour que la tactique `my_inAS` I.2.17 arrive à traiter cette configuration. Par exemple, nous montrons clairement qu'une droite ne peut pas contenir un plan rendant l'inégalité suivante impossible : $3 \leq 2$.

Pour automatiser les preuves dans cet environnement, nous employons le prédicat `InA` qui dénote la relation d'inclusion sur un type A dans le cas des listes relativement à une égalité issue des setoïdes. Dans notre contexte, nous utilisons notre égalité paramétrique `[==]` qui diffère de l'égalité classique de Coq. Nous devons donc spécifier pour quel prédicat les réécritures sont autorisées en utilisant cette égalité. Le prédicat `InA` accepte seulement les réécritures si l'égalité fournie est une relation d'équivalence. Nous détaillons ce mécanisme dans la sous-section suivante.

Nous intégrons la manipulation automatique de ce prédicat `InA` dans la tactique `my_inAS` I.2.17 qui fournit un traitement analogue à `fsetdecide` dans le cas où les ensembles sont représentés sous forme de listes tout en étant bien plus efficace en temps d'exécution. Effectivement,

toutes les couches abstraites de la librairie *Containers* [Les11] sont éliminées facilitant l'unification des types et la vérification du terme de preuve.

```
(* Tactique simple pour du calcul d'inclusion *)
Ltac my_inAS := intuition;unfold inclA in *;unfold equivlistA in *;
  repeat match goal with
  |[H : _ |- _] => progress [intro|intros]
  |[H : _ |- _] => try split;intuition
  |[H : InA eq _ (?P ::  _ ) |- _] => inversion H;clear H
  |[H : _ = _ |- _] => rewrite <-H
  |[H : InA eq _ nil |- _] => inversion H | [H : InA eq ?P ( ?Q :: ?R ) -> _ |- _] =>
            let T:=fresh in assert(T : InA eq P (Q :: R)) by (my_inAS);
            generalize (H T);clear H;clear T;intro
  end.
```

TABLE I.2.17 – Tactique simple pour du calcul automatique d'inclusion.

La première version de cette tactique permet de traiter les calculs d'inclusions d'une liste dans une autre lorsqu'aucune opération d'intersection ou d'union entre listes n'est effectuée. Il est possible de l'étendre pour prendre en compte ces deux cas supplémentaires mais les performances de cette dernière se dégradent considérablement. L'inconvénient majeur de la tactique `my_inAS` est l'utilisation de la tactique `intuition` qui est une primitive de haut-niveau regroupant plusieurs stratégies de résolution, elle n'est donc pas spécifiquement conçue pour démontrer automatiquement nos buts. En effet, certaines des stratégies inclues dans cette tactique ne servent pas à faire avancer la démonstration dans notre cadre géométrique. Elle a aussi l'inconvénient de conserver les modifications des hypothèses lorsqu'elle progresse dans ces différentes stratégies ce qui n'est pas souhaitable dans certaines démonstrations où une trop grande décomposition complique la terminaison de la preuve. Pour corriger ces problèmes, nous proposons une version étendue `my_inAO` I.2.18 qui, selon le motif identifié, applique un lemme intermédiaire adapté. Cette nouvelle tactique se découpe en 3 parties :

- La tactique `solve_equivlistA` récursive permettant de mettre en évidence l'inclusion d'un élément dans une liste. Exemple d'application de cette tactique :

  `inA B (A :: C :: C :: B :: A :: nil)`

- La tactique `inv_unifA` récursive qui simplifie le contexte au maximum en décomposant les hypothèses sur les listes impliquant aussi des intersections et des unions. Exemple d'application de cette tactique, les notations `list_inter` et `++` désignent respectivement l'intersection et l'union entre deux listes :

  `inclA (B :: nil)(list_inter(A :: C :: B :: nil)(C :: B :: nil ++ A :: nil))` $\Rightarrow$
  `inA B (A :: C :: B :: nil)` $\wedge$ `inA B (C :: B :: nil ++ A :: nil)` $\Rightarrow$
  `inA B (A :: C :: B :: nil)` $\wedge$ `(inA B (C :: B :: nil)` $\vee$ `inA B (A :: nil))`

- La tactique `my_inAO` qui teste si la combinaison des deux tactiques précédentes aboutit à une résolution de l'inclusion ou à l'équivalence entre deux listes. Exemple d'application de cette tactique :

  `equivlistA (list_inter(A :: C :: nil)(B :: nil)) (C :: B :: nil ++ A :: nil)`

```
(Tactique optimisée pour du calcul d'inclusion *)
Ltac solve_equivlistA :=
first[assumption | apply InA_cons_hd;reflexivity | apply InA_cons_tl;solve_equivlistA].

Ltac inv_unifA := unfold inclA in *; try split; intros;
         repeat match goal with
         | [H : InA eq _ _ |- _] => inversion H;clear H
         | [H: _ = _ |- _] => rewrite <- H in *;
                   try solve [contradiction|apply eq_sym in H;contradiction];clear H
         | [H : InA eq _ nil |- _] => inversion H
         | [H : InA eq _ (_++_) |- _] => apply InA_app_iff in H; destruct H
         | [H :_ |- InA eq _ (_++_) ] => apply InA_app_iff
         | [H : InA eq _ (list_inter _ _) |- _] =>
                   apply list_inter_split_bis in H; destruct H
         | [H : _ |- InA eq _ (list_inter _ _)] => apply list_inter_split_reverse
         | [H : InA eq ?P ( ?Q :: ?R ) -> _ |- _] => let T:=fresh in
                   assert(T : InA eq P (Q :: R)) by (solve_equivlistA);
                   generalize (H T);clear H;clear T;intro
         end.

Ltac my_inAO := solve[inv_unifA ; first[solve_equivlistA]].
```

TABLE I.2.18 – Tactique optimisée pour du calcul automatique d'inclusion.

Pour revenir à la démonstration de la propriété matroïdale de non-décroissance, nous pouvons maintenant présenter dans la Table I.2.19 le schéma de cette preuve avec les tactiques qui ont été présentées ci-dessus. Pour rappel, cette preuve comporte 25 cas dont 15 sont directement simplifiés par l'utilisation de la tactique I.2.16. Les 10 buts restant correspondant à des situations contradictoires sont prouvés grâce à la mise en évidence d'une contradiction dans les hypothèses.

```
(* Preuve de la propriété matroïdale de non-décroissance *)
Lemma matroid2 : forall e e', inclA eq e e' -> rkl(e)<=rkl(e').
Proof.

intros;case_eq e;case_eq e'; (* pré-traitement & inductions *)
try my_inAO;my_rank. (* simplification des 15 cas triviaux 0 <= 1, 0 <= 2 ... *)
my_rank;subst;unfold inclA in *;assert(HH := H p);my_inAO. (* cas trivial 1 <= 0 *)

unfold rkl;my_rank;subst. (* pré-traitement & simplification pour les cas non triviaux *)
assert( HH := contains_four_non_coplanar_points_sublist [...];my_rank. (* cas 4 <= 0 *)
assert( HH := contains_four_non_coplanar_points_sublist [...];my_rank. (* cas 4 <= 1 *)
assert( HH := contains_four_non_coplanar_points_sublist [...];my_rank. (* cas 4 <= 2 *)
assert( HH := contains_four_non_coplanar_points_sublist [...];my_rank. (* cas 4 <= 3 *)
assert( HH := contains_three_non_collinear_points_sublist [...];my_rank. (* cas 3 <= 0 *)
assert( HH := contains_three_non_collinear_points_sublist [...];my_rank. (* cas 3 <= 1 *)
assert( HH := contains_three_non_collinear_points_sublist [...];my_rank. (* cas 3 <= 2 *)
assert( HH := contains_two_distinct_points_sublist [...];my_rank. (* cas 2 <= 0 *)
assert( HH := contains_two_distinct_points_sublist [...];my_rank. (* cas 2 <= 1 *)
Qed.
```

TABLE I.2.19 – Preuve Coq de la propriété matroïdale de non-décroissance.

2.2.4 Implantation Coq

Le lemme de sous-modularité, $\forall\ X, Y \subseteq E$, $rk(X \cup Y) + rk(X \cap Y) \leq rk(X) + rk(Y)$, est la propriété la plus difficile à démontrer à cause de la gestion des intersections et des unions. Cette preuve contient de nombreux cas non triviaux qui doivent être décomposés dans des lemmes intermédiaires comme illustré dans la Table I.2.20. Le premier exemple indique que si deux droites sont confondues, il est possible de représenter le problème par une unique droite contenant tous les points appartenant aux deux droites. Le deuxième spécifie qu'une droite m peut être incluse dans un plan l pour former un unique objet. Le troisième indique que si deux plans sont confondus, l'union de ces deux plans engendre un plan qui les contient tous les deux. Tous ces lemmes, décrivant des résultats d'intersection et d'union entre différents objets, sont des configurations classiques de la géométrie synthétique qui peuvent être déduit à partir des axiomes.

```
(* L'union de deux droites confondues génère une droite *)
Lemma matroid3_rk2_rk2_interrk2_to_unionrk2 :
forall l m,
rkl l = 2 ->
rkl m = 2 ->
rkl (inter l m) = 2 ->
rkl (l ++ m) = 2.
```

```
(* L'union d'un plan et d'une droite de ce plan génère un plan *)
Lemma matroid3_rk3_rk2_interrk2_to_unionrk3 :
forall l m,
rkl l = 3 ->
rkl m = 2 ->
rkl (list_inter l m) = 2 ->
rkl (l ++ m) = 3.
```

```
(* L'union de deux plans confondus génère un plan *)
Lemma matroid3_rk3_rk3_interrk3_to_unionrk3 :
forall l m,
rkl l = 3 ->
rkl m = 3 ->
rkl (list_inter l m) = 3 ->
rkl (l ++ m) = 3.
```

TABLE I.2.20 – Lemmes intermédiaires sur les cas non triviaux de la sous-modularité.

Finalement, la difficulté principale pendant l'implantation provient de la gestion du mécanisme généralisé de réécriture introduit par l'utilisation des setoïdes avec l'égalité paramétrique. Quand nous définissons un nouveau prédicat en accord avec une égalité qui n'est pas incluse dans la librairie standard, nous devons formaliser une règle autorisant l'application de cette égalité pour effectuer des réécritures dans les prédicats. Dans la terminologie Coq, on parle ici de morphisme, nous fournissons à travers ce morphisme la preuve qu'il est possible d'utiliser notre relation d'équivalence dans ce prédicat. Donc, pour toute définition impliquant le type point, il sera nécessaire d'établir un morphisme semblable à I.2.21. Cela garantit que notre égalité est compatible et cohérente avec la définition dans laquelle elle est utilisée. Il est dès lors possible de substituer un ensemble X par un ensemble Y directement dans la définition du rang.

```
(* Morphisme pour le prédicat rang *)
Instance rank_morph : Proper (@equivlistA Point eq => (@Logic.eq nat)) rkl.
Proof.
[...]
Qed.
```

TABLE I.2.21 – Exemple de morphisme.

Ce morphisme indique que le prédicat `rkl` doit utiliser l'égalité classique de Coq pour les réécritures si les listes en paramètre sont équivalentes. En dépliant le prédicat `Proper`, nous devons prouver le lemme suivant où `equivlist` indique que deux listes sont égales :

Lemme I.2.5 (rank_morph)**.** $\forall$ *x y : list Point, equivlist x y* $\rightarrow$ *rkl x = rkl y*

Toutes les preuves de la géométrie synthétique vers les rangs suivent le même modèle. Une fois que la caractérisation intermédiaire est établie, il suffit de déplier et simplifier à l'aide de la tactique I.2.16 avant d'utiliser des lemmes intermédiaires caractérisant une situation typique en géométrie synthétique ou mettre en évidence des contradictions dans les hypothèses en s'aidant de la tactique I.2.18.

2.3 Statistiques

Nous donnons quelques informations à propos du développement de la librairie Coq qui inclut cette preuve d'équivalence entre la géométrie synthétique et la théorie des matroïdes. Cette preuve d'équivalence a été réalisée en deux étapes : la première s'intéresse à la validité du résultat et la construction de la couche intermédiaire pour formaliser cette démonstration, la seconde se préoccupe de l'automatisation qu'il est possible d'apporter dans ce contexte et du développement des tactiques au fur et à mesure du processus. Nous étudions par la même occasion l'impact de cette automatisation sur les performances et la taille des preuves produites dans l'assistant de preuve Coq. La preuve de l'équivalence totalise 17 000 lignes décomposées comme décrit dans la Table I.2.22.

En automatisant la preuve de l'équivalence, grâce à une décomposition systématique en lemmes intermédiaires et l'utilisation de tactiques, nous avons divisé la taille de la preuve initiale par un facteur 3. Dans le seul but d'établir cette équivalence et de maximiser l'automatisation, ce résultat peut être largement amélioré. Ce développement servant aussi de librairie sur la géométrie d'incidence projective, de nombreux résultats intermerdiaires sont conservés.

	Des rangs vers la G. S.		De la G. S. vers les rangs	
	2D	3D	2D	3D
Lignes de spécification Coq	250	400	650	1 200
Lignes de preuves Coq	300	1 500	2 600	12 500

TABLE I.2.22 – Taille de la preuve de l'équivalence selon la direction et la dimension. L'abréviation G. S. dénote la « Géométrie Synthétique ».

2.4 Traduction bilatérale

La preuve de l'équivalence entre ces deux approches permet de mettre en place un procédé de traduction bidirectionnel. Il est possible de traduire un énoncé géométrique d'une théorie vers l'autre en appliquant une tactique de conversion. Cette traduction peut très bien s'effectuer avant le commencement d'une preuve ou en plein milieu de cette dernière si l'on souhaite changer de formalisme pour continuer à avancer. Pour faire cette traduction, nous utilisons la Table de correspondance suivante I.2.23. Rappelons que les prédicats `collinear` et `coplanar` sont définis à partir du prédicat d'incidence `Incid` dans l'Annexe F.

Configuration géométrique	Géométrie synthétique	Avec les rangs
Un point X :		rk(X) = 1 (facultatif)
Deux points identiques X Y :	`X = Y`	rk(X, Y) = 1
Deux points distincts X Y :	`X <> Y`	rk(X, Y) = 2
Trois points colinéaires X Y Z :	`collinear X Y Z`	rk(X, Y, Z) <= 2
Trois points non colinéaires X Y Z :	¬ `collinear X Y Z`	rk(X, Y, Z) = 3
Quatre points coplanaires X Y Z W :	`coplanar X Y Z W`	rk(X, Y, Z, W) <= 3
Quatre points non coplanaires X Y Z W :	¬ `coplanar X Y Z W`	rk(X, Y, Z, W) = 4

TABLE I.2.23 – Table de correspondance pour la traduction bilatérale entre les deux théories.

Pour compléter cette table de traduction, il faut préciser que les énoncés en géométrie synthétique ne sont plus définis directement à partir de l'incidence, nous utilisons les définitions de plus haut niveau qui les encapsulent afin de faciliter le processus de traduction. En effet, il est plus difficile d'identifier dans un énoncé plusieurs points qui sont incidents à une même droite qu'un simple prédicat de colinéarité. Il est néanmoins toujours possible de revenir à un énoncé purement exprimé en terme d'incidence en dépliant les définitions. En n'utilisant pas directement le prédicat `Incid`, la traduction depuis la géométrie synthétique vers les rangs devient immédiate.

Pour la traduction des rangs vers géométrie synthétique, il suffit d'indiquer comment traiter la fonction `rk` qui est d'arité variable. Dans le cas où le nombre de points dans l'ensemble dépasse l'arité du prédicat correspondant en géométrie synthétique, il suffit de décomposer l'ensemble en n-uplets de points en respectant la propriété de ptransitivité de la généralisation des relations de colinéarités (voir I.1.17). Considérons l'ensemble $rk(X, Y, Z, W, U) = 2$ d'arité 5, nous souhaitons décomposer cet ensemble selon le prédicat de colinéarité. Pour cela, nous conservons toujours les points X et Y définissant la droite et nous rajoutons un par un chacun des points restants pour former une nouvelle règle de colinéarité. Nous obtenons ainsi trois colinéarités : $rk(X, Y, Z) = 2$, $rk(X, Y, W) = 2$ et $rk(X, Y, U) = 2$ qui deviendront respectivement `collinear X Y Z`, `collinear X Y Z` et `collinear X Y U`.

Nous illustrons dans la Table I.2.24 l'application de la tactique de traduction `trad` sur l'énoncé en géométrie synthétique `translate_test` afin d'obtenir l'énoncé équivalent du point de vue des rangs. Ce changement global de contexte avant et après l'application de la tactique de traduction dans l'assistant de preuve Coq est détaillé dans la Table I.2.25.

```
(* Exemple de traduction sur un lemme en Coq *)
Lemma translate_test : forall X Y W Z : Point,
X = Y -> W <> Z -> collinear Y W Z -> collinear X W Z.
Proof.
intros.
trad. (* tactique de traduction )*
[...]
```

TABLE I.2.24 – Exemple de traduction en Coq de la géométrie synthétique vers les rangs.

```
(* Changement de contexte Coq de la géométrie synthétique vers les rangs *)

X, Y, W, Z : Point
H : X = Y
H0 : W <> Z
H1 : collinear Y W Z
______________________________________(1/1)
collinear X W Z

X, Y, W, Z : Point
H : rk (X :: Y :: nil) = 1
H0 : rk (W :: Z :: nil) = 2
H1 : rk (Y :: W :: Z :: nil) <= 2
______________________________________(1/1)
rk (X :: W :: Z :: nil) <= 2
```

TABLE I.2.25 – Illustration du changement de contexte Coq suite à l'application de la tactique de traduction.

Grâce à cette traduction bidirectionnelle, il est possible d'alterner les représentations pour un même énoncé géométrique. D'un côté, nous utilisons la géométrie synthétique qui est très intuitive et très visuelle pour exprimer une configuration mais peu propice à l'automatisation. De l'autre côté, nous considérons l'approche combinatoire avec la fonction de rang permettant de mécaniser plus simplement les démonstrations autour d'une seule et même notion mais qui a le désavantage de rendre les énoncés moins lisibles. Dans la conclusion de cette partie, nous revenons sur les possibilités d'automatisation qui peuvent être envisagées en combinant les deux approches. Nous donnons pour cela, un aperçu des pistes qui ont été explorées dans les chapitres suivants.

Conclusion : partie I

Bilan

Dans cette partie, nous avons présenté le simple cadre géométrique choisi pour étudier la mise en place de l'automatisation et évaluer nos idées sur l'aide à la preuve. Cette géométrie très simple, appelée géométrie d'incidence projective est définie à partir de points et de droites ainsi qu'une relation d'incidence liant ces deux objets. La construction de cette dernière est fondée sur le premier groupe d'axiomes introduit par Hilbert dans l'ouvrage [Hil60] et sa description ne nécessite qu'un ensemble d'axiomes très restreint. Pour spécifier un peu plus cette géométrie, nous considérons la variante projective où deux droites du plan se coupent toujours en un point. Nous formalisons au sein de l'assistant de preuve Coq quatre systèmes d'axiomes en nous inspirant des travaux de Coxeter [Cox03] : deux pour la géométrie d'incidence projective plane, un pour $\geq$3D et un pour la 3D.

Nous nous intéressons ensuite à la méthodologie de la démonstration dans cet environnement géométrique. Pour cela, nous identifions dans les preuves les différentes étapes de démonstration apparaissant de manière récurrente et nous évaluons la difficulté pour mécaniser ces dernières. L'hypothèse forte que nous considérons dans la suite de cette thèse est que la création de nouveaux objets (principalement des points) est une tâche difficile que nous laissons à l'utilisateur ou à des systèmes comportant une intelligence artificielle [Sch19, WCA+14]. Nous proposons dès cette partie une première réflexion sur le méta-problème de l'ordonnancement de ces éléments de preuves et des stratégies de mécanisation qui les accompagnent.

Cette géométrie synthétique permet d'énoncer un problème géométrique assez simplement en nous appuyant sur la description visuelle de la figure associée. L'introduction de la relation de colinéarité et de sa généralisation à la dimension supérieure devient rapidement nécessaire pour conserver une bonne expressivité dans cette théorie. Cependant, nous observons que la mise en place des procédés d'automatisation et leur maintien devient plus complexe pour chaque nouvelle définition prise en compte. Cette constatation nous pousse à considérer une approche alternative combinatoire fondée sur la notion de rang d'un espace projectif pour mécaniser les démonstrations.

Ce concept de rang provenant de la théorie des matroïdes permet d'exprimer de façon homogène toutes les relations usuelles de la géométrie d'incidence projective, non seulement les relations d'incidences qui sont nombreuses mais aussi les relations d'égalité, de colinéarité ou de coplanarité. L'utilisation de ce concept en géométrie d'incidence est initialement introduite par Dominique Michelucci et al. dans leurs travaux [MS04, MS06]. Une fois le fragment de la théorie des matroïdes définissant le concept de rang formellement exposé, nous présentons trois systèmes d'axiomes capturant l'intégralité de la géométrie d'incidence projective.

Pour valider cette nouvelle approche et permettre une traduction bidirectionnelle entre les deux théories, nous étudions la preuve d'équivalence entre ces deux dernières en dimension quelconque. Cette preuve d'équivalence est une tâche complexe (preuves des propriétés matroïdales)

et conséquente (15 000 pas de preuves dans la librairie). De plus, elle apporte une première comparaison entre les deux approches tout en analysant l'automatisation dans ces démonstrations. La contribution importante de cette démonstration est bien évidemment la possibilité de changer de théorie en fonction des besoins qui sont principalement la lisibilité et l'automatisation.

Perspectives

Le procédé de traduction bidirectionnel détaillé dans cette partie ne représente qu'un prototype fonctionnant sur des exemples géométriques simples. Le but est d'étoffer la tactique de traduction pour élargir le pipeline du prouveur automatique de la Partie III. À l'heure actuelle, nous pouvons traduire un énoncé en géométrie synthétique vers un énoncé avec des rangs sur lequel on peut appliquer nos procédés d'automatisations. L'idée consiste à élargir cette traduction de contexte d'une théorie vers l'autre à une traduction globale des pas de preuves. Nous pourrions ainsi obtenir une preuve en géométrie synthétique à partir d'une démonstration qui a été automatisée avec les rangs et ainsi analyser la lisibilité de la démonstration selon les deux approches.

Par ailleurs, tous les travaux de formalisation dans l'assistant de preuve Coq détaillés dans cette partie sont intégrés dans notre librairie sur la géométrie projective. Comme évoqué dans le Chapitre I.1, nous désirons modifier certains aspects de notre implantation. La première modification que nous visons est une meilleure séparation entre le développement constructif et non constructifs de la géométrie d'incidence projective. Le deuxième point technique est le remplacement de l'égalité classique sur les droites par une égalité paramétrique.

Une piste à plus long terme suggérée par les rapporteurs des articles [BMS16, BMS19] est d'analyser d'autres formalisations cryptomorphiques de la théorie des matroïdes pour l'automatisation des preuves en géométrie d'incidence projective. Notons que la notion de plat utilisée pour définir l'intuition géométrique de la fonction de rang est déjà considérée dans notre développement.

Dans la suite, nous analysons attentivement la manière dont la mécanisation a été mise en place dans la preuve de l'équivalence et plus particulièrement dans l'implication des rangs vers la géométrie synthétique. Cette preuve permet d'identifier le coeur du raisonnement en géométrie d'incidence projective. En effet, ce sont les propriétés matroïdales capturant intrinsèquement l'inclusion, l'union et l'intersection entre les plats (générés par des ensembles de points) qui représentent la majorité des étapes lors des démonstrations. Dans une optique d'automatisation de la preuve, il devient alors primordial de mécaniser l'application de ces propriétés pour déterminer de nouvelles déductions à partir des hypothèses. Cependant, il reste le problème majeur de comment guider l'automatisation vers le calcul d'ensembles de points utiles à la démonstration afin de diminuer l'explosion combinatoire provenant de la clôture systématisée des hypothèses. En effet, si nous considérons une configuration géométrique à n points, le calcul du rang pour chaque ensemble de points de cet énoncé revient à calculer le rang de chaque partie de l'ensemble contenant tous les points à savoir le rang de 2^n parties [4]. Pour limiter temporairement cette explosion combinatoire, nous continuons d'observer la mécanisation des preuves dans le contexte précis des géométries finies définies par extension dans la partie Partie II où la clôture des hypothèses devient inutile. Nous réintégrons ce mécanisme de clôture à partir des propriétés matroïdales dans la Partie III lorsque le prouveur automatique de configuration géométrique d'incidence sera décrit.

4. Remarque qui est vrai en considérant uniquement les propriétés matroïdales. En ajoutant les propriétés sur la généralisation des relations d'incidence, nous pouvons simplifier le problème et considérer les k-tuples, avec k $\leq$ 3 en 2D et k $\leq$ 4 en 3D.

Deuxième partie

Étude de cas en géométrie finie

CHAPITRE II.1

Formalisation de « petits » modèles finis en géométrie projective

"Inspiration is needed in geometry, just as much as in poetry"

Aleksandr Sergeyevich Pushkin (1799–1837)

Résumé

En utilisant à loisir les deux formalisations équivalentes de la géométrie d'incidence projective, nous étudions une méthologie possible dans le cadre très spécifique des géométries finies. Ces géométries ne permettent de considérer qu'un nombre fini de points. En outre, la formalisation des différents modèles de la géométrie finie est réalisée par extension en énumérant l'intégralité des objets composant le modèle à savoir l'ensemble des points, l'ensemble des droites, l'ensemble des plans, l'ensemble des incidences, etc. Nous éliminons ainsi la recherche d'informations supplémentaires en donnant dès le départ toutes les données dont le système a besoin pour prouver un résultat. Dans ce contexte précis, nous étudions l'automatisation des démonstrations que les espaces finis $pg(n, q)$ sont effectivement des modèles de la géométrie d'incidence projective respectant de plus les propriétés de Desargues et Pappus.

Nous démarrons ce chapitre par l'introduction du cadre théorique sur les géométries finies en rappelant notamment la notion de corps de Galois et la méthodologie à suivre pour construire de tels espaces finis (section 1). Dans la suite, nous décrivons comment ces modèles peuvent être formalisés algorithmiquement et importés dans l'assistant de preuve Coq (section 2). Puis, nous détaillons avec soin la vérification formelle de ces modèles en énonçant un ensemble de critères généraux qui aident à rendre les preuves hautement combinatoires traitables (section 3). Au final, nous confrontons les résultats obtenus grâce à nos deux formalisations complémentaires de la géométrie avec la librairie *tptp* dans le but d'obtenir un point de comparaison avec les prouveurs automatiques (section 3 suite).

Contenu

Prouver des propriétés en géométrie projective, notamment que des plans ou des espaces sont des modèles de cette géométrie, repose sur l'analyse de quelques configurations générales ainsi que les nombreux cas dégénérées associés. L'utilisation d'un assistant de preuve tel que Coq [BC04] facilite la tâche de l'utilisateur pour écrire une preuve correcte. En effet, le système Coq force ce dernier à considérer tous les cas possibles dans une preuve. De plus, tous les détails de la preuve doivent être fournis, ce qui permet au système de vérifier le terme de preuve. En retour, l'inconvénient majeur est que le développement de ces preuves nécessite une importante quantité de travail qui peut être heureusement partiellement automatisée grâce au langage de tactiques *Ltac* [Del00].

Dans ce chapitre, nous étudions la mécanisation des preuves en géométrie d'incidence projective dans le cadre spécifique des géométries finies. Lorsqu'on donne en extension l'ensemble des points, l'ensemble des droites, et l'ensemble des relations d'incidence, le problème est grandement simplifié, il « suffit » de valider ce système en montrant que c'est un modèle de la géométrie projective. En utilisant deux descriptions formelles équivalentes de la géométrie projective, nous vérifions dans quelle mesure chacune de ces théories permet d'obtenir des preuves qui sont lisibles, faciles à écrire et simples à vérifier. Pour atteindre cet objectif, nous travaillons sur quelques modèles de la géométrie finie : les plans finis $pg(2,\ 2)$, aussi connu comme le plan de Fano, $pg(2,\ 3)$, $pg(2,\ 4)$, $pg(2,\ 5)$ et $pg(2,\ 7)$; mais aussi le plus petit espace fini $pg(3,\ 2)$ et $pg(3,\ 3)$. Au fur et à mesure que les modèles devenaient plus grands, nous avons dû optimiser nos techniques de preuves pour faire face à la complexité intrinsèque de ces modèles et maintenir sous contrôle l'usage de la mémoire et le temps d'exécution de la recherche d'une preuve.

Bien que l'approche matroïdale ait déjà servi pour effectuer la mécanisation de la preuve du théorème de Desargues en utilisant uniquement les rangs [MNS09, MNS12], nous comparons plus précisément l'automatisation des preuves dans notre contexte géométrique grâce aux deux formalisations. Pour cela, nous analysons non seulement les performances de chaque approche pour établir des preuves automatiques, mais aussi la facilité de la mise en place du schéma de preuve et sa généralisation lors du passage au modèle fini selon la dimension supérieure. Finalement, à titre indicatif, nous comparons notre approche avec des prouveurs automatiques de la librairie *tptp* [Sut10].

1 Introduction aux modèles finis

Cette partie vise à donner les principales définitions et propriétés sur les modèles finis que nous étudions dans ce chapitre. Tous les détails peuvent être trouvés dans [Bat97, BW11, Dem12, RK70, Ros17].

1.1 Groupe, corps, espace vectoriel, corps fini

Dans cette sous-section, nous rappelons les différentes structures algébriques qui sont nécessaires pour la définition d'un corps fini.

1.1.1 Groupe

Commençons par rappeler la notion de groupe nécessaire pour la définition d'un corps :

Définition (Groupe)**.** *Un groupe est un ensemble $\mathbb{G}$ avec une opération binaire « $\bullet$ » (c'est à dire une fonction de $\mathbb{G} \times \mathbb{G}$ dans $\mathbb{G}$) satisfaisant les propriétés suivantes :*

(i) L'opération est associative, $\forall a,\ b,\ c \in \mathbb{G},\ a \bullet (b \bullet c) = (a \bullet b) \bullet c$;

(ii) Il existe un élément neutre pour l'opération, $\exists e \in \mathbb{G}$, $\forall a \in \mathbb{G}$, $a \bullet e = e \bullet a = a$;

(iii) Tout élément a *de* $\mathbb{G}$ *possède un inverse* a^{-1}, $a \bullet a^{-1} = a^{-1} \bullet a = e$.

Un groupe $(\mathbb{G}, \bullet)$ est commutatif (abélien) si $\forall a, b \in \mathbb{G}$, $a \bullet b = b \bullet a$. Quelques exemples de groupes abéliens connus sont : $(\mathbb{R}, +)$, $(\mathbb{Q}, +)$, $(\mathbb{C}, +)$ ou encore $(\mathbb{R} - \{0\}, \times)$.

1.1.2 Corps

Nous définissons ensuite la notion de corps :

Définition (Corps)**.** *Un corps est un ensemble* $\mathbb{F}$ *muni de deux opérations « + » et « × » satisfaisant les propriétés suivantes :*

(i) $\mathbb{F}$ *est un groupe abélien avec la loi de composition interne « + » et l'élément neutre* 0 ;

(ii) Les éléments non nuls de $\mathbb{F}$ *forment un groupe abélien avec la loi de composition interne « × » et l'élément neutre* 1 ;

(iii) La multiplication « × » est distributive pour l'addition, autrement dit $\forall$ a, b, $c \in \mathbb{F}$, $a \times (b + c) = a \times b + a \times c$.

Le nombre d'éléments dans $\mathbb{F}$ définit ce qu'on appelle l'ordre du corps $\mathbb{F}$. Les corps les plus connus sont infinis tels que les nombres réels $\mathbb{R}$, les rationnels $\mathbb{Q}$ ou les nombres complexes $\mathbb{C}$. Rajoutons qu'il est possible de considérer un corps où la multiplication n'est pas nécessairement commutative, ces corps sont généralement appelés corps gauche ou anneau de division ("skew field").

Un sous-corps de $\mathbb{F}$ est un ensemble de $\mathbb{F}$ qui est lui-même un corps muni des opérations d'addition et de multiplication. $\mathbb{Q}$ est un sous-corps de $\mathbb{R}$ qui est lui-même un sous-corps de $\mathbb{C}$.

1.1.3 Espace vectoriel sur un corps

Puis nous apportons la définition d'un espace vectoriel construit sur un corps :

Définition (Espace vectoriel sur un corps)**.** *Un espace vectoriel sur un corps commutatif* $\mathbb{F}$ *est un ensemble* $\mathbb{V}$*, dont les éléments sont appelés vecteurs, muni de deux lois :*

(i) une loi de composition interne « + » appelée addition ou somme vectorielle ;

(ii) une loi de composition externe à gauche « × » appelée multiplication par un scalaire ;

telles que les propriétés suivantes soient vérifiées pour tous les éléments $X, Y, Z \in \mathbb{V}$ *et tous les scalaires* $\alpha, \beta \in \mathbb{F}$ *:*

- *Commutativité de l'addition vectorielle :* $X + Y = Y + X$;
- *Associativité de l'addition vectorielle :* $(X + Y) + Z = X + (Y + Z)$;
- *Vecteur nul comme élément neutre :* $X + 0 = 0 + X$;
- *Vecteur opposé :* $X + (-X) = 0$;
- *Associativité du produit scalaire :* $\alpha(\beta X) = (\beta\alpha)X$;

- *Distributivité de la somme scalaire :* $(\alpha + \beta)X = \alpha X + \beta X$;
- *Distributivité de l'addition vectorielle :* $\alpha(X + Y) = \alpha X + \alpha Y$;
- *L'élément neutre multiplicatif du corps* $\mathbb{F}$ *:* $1 \times X = X$.

1.1.4 Corps fini

Un corps fini est un corps contenant un nombre fini d'éléments. À isomorphisme près, un corps fini est entièrement déterminé par son cardinal, qui est toujours une puissance d'un nombre premier, ce nombre premier étant sa caractéristique. Les corps finis sont utilisés en théorie algébrique des nombres mais aussi en théorie des codes ou en cryptographie [Ben12] avec le développement de l'informatique. Ces corps finis sont aussi appelés corps de Galois, d'après Évariste Galois dont les travaux furent publiés en 1830 [Gal08].

Définition (Corps fini ou corps de Galois)**.** *Un corps fini noté* $GF(q)$ *ou* $\mathbb{F}_q$ *possède* q *éléments où* q *est une puissance d'un nombre premier. Ce corps fini d'ordre* q *satisfait les propositions et propriétés suivantes :*

(i) *Pour tout* $x \in GF(q)$*, nous avons* $x^q = x$;

(ii) *Soit* p *un nombre premier. Le corps* $GF(p)$ *est l'ensemble* $\mathbb{Z}/p\mathbb{Z}$ *muni de l'addition et de la multiplication standard ;*

(iii) *Le corps fini* $GF(p^h)$ *avec* $h \in \mathbb{Z}^+$ *peut être construit de la manière suivante. Soit* $f \in GF(p)[x]$ *un polynôme de degré* h*, irréductible sur* $GF(p)$*. L'anneau quotient* $GF(p)[x]/f(x)$ *forme un corps muni de l'addition et la multiplication standards puisque* f *est irréductible. Ce corps est noté* $GF(p^h)$ *et possède* p^h *élément ;*

(iv) *Le nombre premier* p *de* $GF(p^h)$ *est appelé la caractéristique du corps ;*

(v) $GF(p^r)$ *est un sous-corps de* $GF(p^h)$ *si et seulement si* r *divise* h ;

(vi) $GF(p^h)$ *est un espace vectoriel de dimension* h *sur* $GF(p)$;

(vii) *Les éléments du corps de Galois* $GF(p^h)$ *sont :*

$$\begin{aligned} GF(p^h) = {} & (0,\ 1,\ 2,\ \ldots,\ p-1) \cup \\ & (p,\ p+1,\ p+2,\ \ldots,\ p+p-1) \cup \\ & (p^2,\ p^2+1,\ \ldots,\ p^2+p-1,\ p^2+p,\ p^2+p+1,\ \ldots,\ p^2+p+p-1) \cup \\ & \ldots \cup \\ & (p^{h\text{-}1},\ p^{h\text{-}1}+1,\ \ldots,\ p^{h\text{-}1}+p-1,\ p^{h\text{-}1}+p^{h\text{-}2},\ p^{h\text{-}1}+p^{h\text{-}2}+1,\ \ldots,\ p^{h\text{-}1}+p^{h\text{-}2}+\ldots+p+p-1) \end{aligned}$$

Le degré de chaque élément est au plus $h - 1$.

Le corps Galois $GF(5) = (0,\ 1\ ,\ 2,\ 3,\ 4)$ est composé de 5 éléments où chacun d'eux a un polynôme caractéristique de degré 0 (une constante).

Tandis que le corps de Galois $GF(2^3) = (0,\ 1,\ 2,\ 2+1\ ,\ 2^2,\ 2^2+1,\ 2^2+2,\ 2^2+2+1)$ est composé de 8 éléments (0, 1, 2, 3, 4, 5, 6, 7) avec des polynômes caratéristiques de degré 2 au maximum.

1.2 Espace projectif fini

Bien qu'il existe de nombreux modèles que l'on peut appeler géométries finies, nous portons notre attention sur ceux provenant des espaces finis affines et projectifs. De tels espaces sont constructibles via l'algèbre, c'est à dire en commençant par un espace vectoriel $V(n+1, q)$ de dimension $n+1$ sur un corps fini $GF(q)$. L'espace projectif $pg(n, q)$ est la géométrie dont les points, les droites, les plans, ..., les hyperplans sont les sous-espaces de $V(n+1, q)$ de dimension 1, 2, 3, ..., n. Un hyperplan est un sous-espace de codimension 1, c'est à dire que sa dimension est plus petite d'une unité par rapport à la dimension de l'espace tout entier. La dimension d'un sous-espace de $pg(n, q)$ est un de moins que le rang d'un sous-espace de $V(n+1, q)$. Notons que les espaces construits de cette manière sont tous desarguésiens.

Une question naturelle s'ensuit : est-ce que toutes les géométries projectives finies sont isomorphes à un espace projectif $pg(n, q)$?

La réponse est « oui » lorsque $d \geq 3$; ce résultat découle de deux théorèmes importants de la géométrie et de l'algèbre. Le premier est issu de la géométrie projective : c'est le fameux théorème de Desargues présenté dans le Chapitre I.1. Ce théorème est toujours vrai pour un espace projectif défini algébriquement à partir d'un corps commutatif ou non [Kod14]. En dimension 3 ou plus, tout espace projectif fini provient nécessairement d'un espace vectoriel sur un corps fini et est donc désarguésien. Le second théorème est énoncé par Wedderburn en 1905 [MW05], il indique que « La multiplication dans un corps fini est nécessairement commutative », en d'autres termes « Tout corps fini est commutatif ».

La réponse est « non » en dimension 2 ; le plus petit plan projectif ne découlant pas d'un corps et qui est donc non desarguésien appartient à une famille de plans appelée « plans de Hughes ». C'est un plan d'ordre 9 composé de 91 points et 91 droites [LKT91, RK70].

Définition. *Un plan projectif d'ordre q est une géométrie qui satisfait les axiomes de la Table I.1.2 et qui contient q points. Un tel plan projectif d'ordre q vérifie la liste non exhaustive de propositions et propriétés suivantes :*

(i) Les plans projectifs ont q^2+q+1 points et q^2+q+1 droites. Chaque point réside sur $q+1$ droites et chaque droite contient exactement $q+1$ points ;

(ii) Principe de dualité : Pour tout énoncé sur les plans projectifs finis qui est un théorème, l'énoncé dual obtenu en interchangeant "point" par "droite" et en permutant "point sur une droite" par "droite passant par un point" est aussi un théorème ;

(iii) Il existe un plan projectif d'ordre p^h avec p premier et h un entier positif. Il est possible de construire méthodiquement avec des coordonnées ces plans projectifs en considérant les sous-espaces d'un espace vectoriel de dimension 3 sur un corps fini $GF(q)$;

(iv) Il n'y a pas de plan projectif connu pour un ordre quelconque qui ne soit pas une puissance d'un nombre premier. Le plus petit cas qui est un problème ouvert est l'ordre 12. Il a été prouvé qu'il n'existe aucun plan projectif d'ordre 6 [Bos38, Tar00] ou 10 [Lam96, LTS89]. La conjecture suivante reste ouverte : "L'ordre d'un plan projectif fini est toujours la puissance d'un nombre premier".

(v) Il existe quatre plans projectifs non isomorphiques d'ordre 9, trois d'entre eux (plan de Halls et Hughes [Hal43, RK70]) sont non désarguésiens ;

(vi) La table II.1.1 résume les faits connus à propos de l'existence des plans projectifs d'ordre q pour $1 \leq q \leq 12$;

(vii) L'existence d'un plan projectif d'ordre q est équivalent à l'existence d'un plan affine d'ordre q.

Ordre	2	3	4	5	6	7	8	9	10	11	12
Nombre de plans projectifs	1	1	1	1	0	1	1	4	0	≥ 1	?
Nombre total de points/droites	7	13	21	31	0	57	73	91	0	133	?

TABLE II.1.1 – Nombre de plans projectifs connus d'ordre q.

À toutes ces propriétés sur les plans finis s'ajoutent des formules générales non spécifiques à la dimension permettant de déterminer précisément le nombre de sous-espaces vectoriels pour chacune des dimensions d ($1 \leq d \leq n$) d'un plan projectif d'ordre q modulo différentes contraintes (passage par un point ou appartenance à un hyperplan ...) [Dem12]. Le théorème suivant qui est le plus général donne le nombre de sous-espaces appartenant à une dimension spécifique d'un espace projectif fini.

Théorème II.1.1. *Le nombre de sous-espaces de dimension d de $pg(n, q)$ tel que $1 \leq d \leq n$ est déterminé par le produit suivant :*

$$\prod_{i=0}^{d} \frac{q^{n+1-i}-1}{q^{i+1}-1}$$

Pour illustrer ces définitions et propriétés, nous donnons quelques figures de modèles finis qui sont analysés dans ce chapitre. Nous obtenons la structure d'incidence de la Figure II.1.1 en fixant $n = q = 2$. Ce plan fini $pg(2, 2)$ possède un groupe de 168 automorphismes[1]. Dans la figure II.1.2 nous considérons une des 5616 configurations possibles du plan projectif fini d'ordre 3. Finalement, la Figure II.1.3 représente le plus petit modèle fini de dimension 3 : $pg(3, 2)$ contenant 15 points, 35 droites et 15 plans.

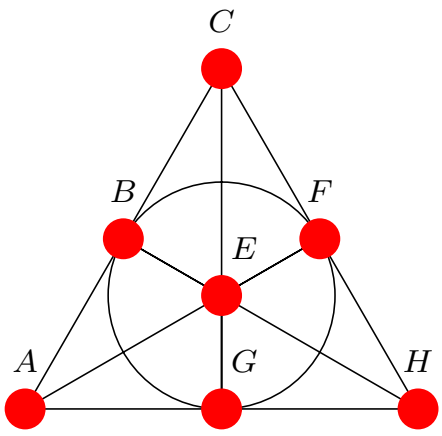


FIGURE II.1.1 – Plan projectif fini $pg(2, 2)$ ou de Fano : 7 points et 7 droites.

1. Étude des petits plans projectifs d'ordre connu : http ://ericmoorhouse.org/pub/planes/

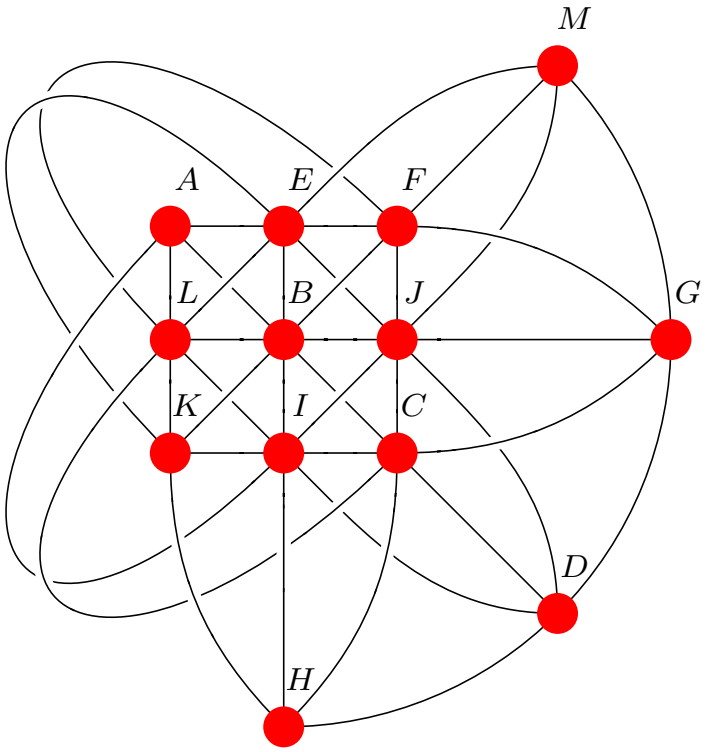


FIGURE II.1.2 – Plan projectif fini $pg(2, 3)$: 13 points et 13 droites ($AEFG$, $CELM$, ...).

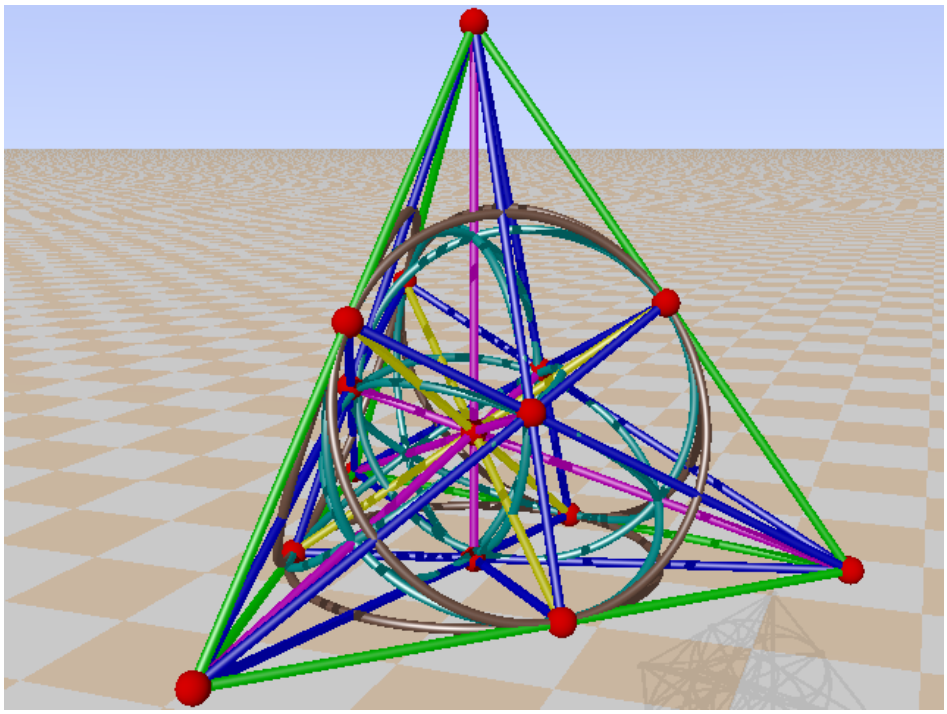


FIGURE II.1.3 – Plan projectif fini $pg(3, 2)$ illustré par Frans Marcelis.

En gardant à l'esprit tous ces résultats théoriques, nous nous concentrons sur la formalisation de ces modèles finis uniquement avec la caractérisation géométrique qui est purement axiomatique. Autrement dit, nous oublions la définition algébrique de ces modèles ; nous n'utilisons à aucun moment des coordonnées homogènes.

2 Génération des modèles finis

Les modèles finis $pg(n, q)$ fournissent un banc d'essais approprié pour tester nos stratégies de mécanisation des preuves en Coq. En effet, ces derniers sont bien connus et faciles à engendrer, ils proposent un cadre où la combinatoire est élevée permettant d'effectuer des tests de stress qualitatifs. Mais avant de pouvoir automatiser le raisonnement dans ce cadre, nous devons d'abord générer ces modèles. Pour cela, il faut spécifier ces modèles dans leur intégralité en énumérant l'ensemble de tous les points, l'ensemble de toutes les droites et l'ensemble de toutes les incidences.

2.1 Recherche des modèles

La description de chacun de ces modèles diffère selon principalement l'approche axiomatique mais aussi selon la dimension et l'ordre. D'un côté en géométrie synthétique, nous caractérisons les points, les droites et la relation d'incidence indiquant quels sont les points appartenant à chacune des droites. De l'autre côté avec l'approche matroïdale, nous énumérons tous les plats décrivant le modèle : l'ensemble des points, l'ensemble des droites et l'ensemble des plans. En donnant un maximum d'informations dès la génération de l'espace fini, nous évitons la recherche des informations supplémentaires lors des preuves, car l'énumération de tous les objets contient déjà toute la description du modèle.

Par exemple, pour engender le modèle $pg(2, 3)$ comportant 13 points et 13 droites du point de vue des matroïdes, il est nécessaire de définir le rang de 339 ensembles : 13 ensembles pour les points, 78 ensembles pour les points distincts, 13 ensembles pour les droites et 234 triplets de points ordonnés représentant uniquement des parties du plan. De cette manière, le système n'a donc pas besoin de vérifier qu'un triplet de points n'appartient pas à une des droites du plan pour finalement déduire que ce triplet détermine un plan. Néanmoins, cette création du modèle fini devient trop laborieuse pour être réalisée manuellement, surtout si les modèles sont encore plus gros.

La recherche de ces modèles finis et la génération de leur spécification sont effectuées par un programme externe en langage C [KRB84]. Ce programme permet d'engendrer une structure candidate à être un plan projectif d'ordre n avant de fabriquer un fichier *Gallina* décrivant ce modèle. Nous distinguons la manière dont nous créons ces modèles en fonction de la dimension. Pour les plans finis $pg(2, q)$, un petit algorithme de recherche brute-force a été réalisé. Il permet de construire tous les plans finis jusqu'à l'ordre 9 inclus (excepté bien évidemment le plan d'ordre 6 dont la preuve de non existence est disponible dans la littérature [Bos38]). Pour les espaces finis de dimension 3, la spécification des modèles s'appuie sur des tables d'incidence disponibles dans la littérature.

2.2 Construction des modèles finis

Nous détaillons dans la suite de cette sous-section la création des modèles finis en fonction de la dimension.

2.2.1 Plan fini

Pour construire ces plans finis sans faire usage d'une méthode algébrique, nous définissons algorithmiquement la table qui lie les points et les droites. En nous inspirant de ces travaux [Lam96,LKT91], un algorithme simple mais non complet, reprenant certains résultats provenant des carrés latins [2], permet de reconstruire des plans projectifs d'ordre q. Le principe consiste à créer une table où les lignes contiennent les points du modèle fini et les colonnes représentent les droites de notre plan fini. Dans chaque ligne, les points sont stockés dans l'ordre croissant de leur indice. Nous obtenons la Table II.1.2 à l'initialisation pour le calcul du plan projectif d'ordre 2 contenant 7 points (entre 0 et 6) et 7 droites (une pour chaque colonne). Ce tableau ne représente pas pour l'instant une configuration valide du modèle fini.

Ligne 1	0	1	2	3	4	5	6
Ligne 2	0	1	2	3	4	5	6
Ligne 3	0	1	2	3	4	5	6

TABLE II.1.2 – Initialisation de la table de construction du plan fini.

Le principe de l'algorithme consiste à décaler chaque série de points jusqu'à obtenir une configuration où la règle suivante est respectée : chaque colonne doit contenir une droite qui est unique, en d'autres termes deux points distincts sont associés l'un à l'autre une seule et unique fois. Pour obtenir une telle configuration, il est nécessaire d'effectuer un décalage d'une ligne $l+1$ lorsqu'elle rentre en conflit avec l'une des l premières lignes. Si tous les décalages de la ligne $l+1$ posent un problème, l'algorithme revient à l'étape précédente et tente un décalage supplémentaire de la ligne l.

Sur l'exemple de la Table II.1.2, la première ligne n'est jamais considérée en conflit, elle est fixe. La deuxième ligne rentre en conflit avec la première, nous décalons de droite à gauche cette dernière pour obtenir la configuration II.1.3. Aucun conflit n'apparaît entre les deux lignes, l'algorithme considère que la configuration des deux premières lignes est valide temporairement.

Il passe donc à la ligne suivante qui est la troisième et dernière ligne de notre plan projectif fini $pg(2,\,2)$. L'algorithme décale une première fois la troisième ligne, elle est nécessairement en conflit avec la deuxième, leur positionnement est identique II.1.4. L'algorithme continue avec un second décalage de la dernière ligne montré par la Table II.1.5. L'unicité des droites est alors remise en cause, les points 1 et 2 ne peuvent pas appartenir à la fois à la droite 012 et 123. L'algorithme doit donc effectuer une rotation supplémentaire présentée par la Table II.1.6. La configuration obtenue est valide, deux colonnes quelconques n'ont pas deux points en commun et chaque point apparaît exactement dans 3 droites, le plan d'ordre 2 a été créé avec les droites 013, 124, 235, 346, 045 156 et 026.

Ligne 1	0	1	2	3	4	5	6
Ligne 2	1	2	3	4	5	6	0
Ligne 3	.	.	.	.	.	.	.

TABLE II.1.3 – Premier décalage de la ligne 2.

2. En combinatoire, un carré latin est un tableau carré $n \times n$ rempli avec n symboles distincts. Chacun de ces symboles apparaît exactement une fois par ligne et une fois par colonne.

Ligne 1	0	1	2	3	4	5	6
Ligne 2	1	2	3	4	5	6	0
Ligne 3	1	2	3	4	5	6	0

TABLE II.1.4 – Premier décalage de la ligne 3.

Ligne 1	0	1	2	3	4	5	6
Ligne 2	1	2	3	4	5	6	0
Ligne 3	2	3	4	5	6	0	1

TABLE II.1.5 – Deuxième décalage de la ligne 3.

Ligne 1	0	1	2	3	4	5	6
Ligne 2	1	2	3	4	5	6	0
Ligne 3	3	4	5	6	0	1	2

TABLE II.1.6 – Troisième décalage de la ligne 3.

Il est possible de laisser l'algorithme continuer sa recherche pour construire d'autres plans qui sont isomorphes au premier. Nous résumons dans l'Algorithme II.1.1 la recherche d'un plan fini d'ordre q.

Algorithme II.1.1 : Recherche d'un plan fini d'ordre q.

Entrée(s) : L'ordre du plan projectif q
Sortie(s) : Tableau contenant les droites du plan projectif fini d'ordre q

```
1  Allocation et initialisation du tableau contenant les points et les droites
2  pour chaque ligne l de 2 à q faire
3      si tous les décalages de l n'ont pas encore été testés faire
4          Décalage de la ligne l
5          si la ligne l n'est pas en conflit avec une ligne précédente faire
6              Passage à la ligne l + 1
7          fin si
8      sinon
9          Revenir à la ligne l − 1
10         si l est égal à 1 faire
11             Arrêt de l'algorithme : échec
12         fin si
13         Réinitialiser à la configuration initiale toutes les lignes > l
14     fin si
15 fin pour chaque
```

2.2.2 Espace fini

La construction des espaces projectifs finis de dimension 3 est bien plus difficile à réaliser de manière algorithmique. L'espace projectif d'ordre 3 contient presque autant de droites qu'un plan projectif d'ordre 11. À notre connaissance, la littérature ne propose pas de méthode simple pour construire de tels espaces sans passer par une méthode algébrique. Pour construire automatiquement les modèles finis $pg(3, 2)$ et $pg(3, 3)$, nous concevons un algorithme qui, à partir des tables d'incidence entre points et droites des articles suivants [Bru11, Pri99], produit automatiquement

le modèle dans son intégralité. La vérification formelle du modèle $pg(3, 3)$, construit à partir de la relation d'incidence provenant de l'article [Pri99], a permis de montrer que cette table publiée comporte plusieurs erreurs. Nous avons localisé et corrigé ces erreurs en utilisant la propriété qui indique le nombre de points que doit contenir une droite.

2.3 Pré-validation des plans finis

Une fois la construction du plan projectif achevée, avant de fabriquer un fichier au format *Gallina*, quelques vérifications de certains axiomes projectifs sont réalisées sur la table d'incidence associée au plan. La construction de la matrice d'incidence à partir du tableau contenant les droites du plan projectif est triviale et immédiate. La Table II.1.7 contient la matrice d'incidence associée au plan projectif d'ordre 2 construit précédemment.

	L0	L1	L2	L3	L4	L5	L6
P0	1	0	0	0	1	0	1
P1	1	1	0	0	0	1	0
P2	0	1	1	0	0	0	1
P3	1	0	1	1	0	0	0
P4	0	1	0	1	1	0	0
P5	0	0	1	0	1	1	0
P6	0	0	0	1	0	1	1

TABLE II.1.7 – Exemple de matrice d'incidence du plan projectif d'ordre 2.

Sur cette matrice d'incidence représentant un plan projectif d'ordre q, les propriétés suivantes sont au minimum vérifiées :

- Chaque ligne possède $q + 1$ points
- Chaque point appartient à $q + 1$ droites
- Il existe toujours un unique point commun entre deux droites distinctes

L'algorithme offre la possibilité de vérifier chacun des axiomes du plan projectif ainsi que la propriété de Desargues (voir l'Annexe C). Cette vérification n'est cependant pas rendue systématique sachant que l'objectif consiste à exporter le modèle construit afin qu'il soit validé formellement dans le système Coq.

2.4 Export des modèles en langage *Gallina*

La fabrication automatique des fichiers en langage *Gallina* contenant les différents plans et espaces finis suit un patron permettant de plus facilement réexploiter tous les outils et structures mis en place pour réaliser les différentes preuves. Tous les modèles ainsi générés sont quasiment identiques pour l'ordre et la dimension. Cependant, le patron diffère largement en fonction de l'approche géométrique que nous employons.

2.4.1 Modèle en géométrie synthétique

La génération d'un modèle, décrits avec cette axiomatisation, contient deux types inductifs pour les points et les droites ainsi qu'un prédicat booléen pour la relation d'incidence (voir Table

II.1.8). Cette description contient tous les éléments pour prouver que c'est un modèle de la géométrie projective respectant de plus la propriété de Desargues.

```
(* Type inductif pour les points *)
Inductive ind_Point : Set := A | B | C | ... | K | L | M.

(* Type inductif pour les droites *)
Inductive ind_line : Set := ABCD | AEFG | AIJM | AHKL | BEHI | BGJL
| BFKM | CELM | CFHJ | CGIK | DEJK | DGHM | DFIL.

(* Prédicat booléen pour la relation d'incidence *)
Définition Incid_bool (P:Point) (l:Line) : bool := match P with
  | A => match l with
           | ABCD | AEFG | AIJM | AHKL => true
           | _ => false
         end
  [...]
end.

Definition Incid : Point -> Line -> Prop := fun P L => (Incid_bool P L = true).
```

TABLE II.1.8 – Application du patron au modèle fini $pg(2, 3)$ en géométrie synthétique.

Tous les modèles finis de plan $pg(2, q)$ suivent exactement le même patron avec plus ou moins de points et de droites impliqués dans la spécification. Pour la génération des modèles de dimension 3, il n'est pas nécessaire d'énoncer la notion de plan en géométrie synthétique puisque les axiomes sont exprimés uniquement à partir de points et de droites (voir l'Annexe A).

2.4.2 Modèle exprimé à l'aide des rangs

Pour l'approche matroïdale de la géométrie, la génération d'un modèle contient uniquement des points. Cet ensemble de points est ensuite utilisé pour exprimer l'ensemble des plats composant le modèle fini : les plats composés d'un unique point, les plats spécifiant que toute paire de points distincts du modèle forme une droite, les plats représentant les droites et les plats définissant les plans (voir Table II.1.9). Pour formaliser les espaces finis, il est indispensable de considérer l'ajout des plats permettant de capturer l'espace tout entier.

```
(* Paramètre définissant tous les points du modèle *)
Parameter A B C D E F G H I J K L M : Point.

(* Décidabilité sur les points du modèle *)
Parameter is_only_13_pts : forall P, {P=A}+{P=B}+{P=C}+[...]+{P=K}+{P=L}+{P=M}.

(* Plats pour les points *)
Parameter rk_points : rk(A :: nil) = 1 /\ rk(B :: nil) = 1 /\ rk(C :: nil) = 1 /\
[...] /\ rk(M :: nil) = 1.

(* Plats pour les paires de points distincts *)
Parameter rk_distinct_points :
rk(A :: B :: nil) = 2 /\ rk(A :: C :: nil) = 2 /\ rk(A :: D :: nil) = 2 /\
[...] /\ rk(L :: M :: nil) = 2.

(* Plats pour les droites *)
Parameter rk_lines :
rk (A :: B :: C :: D :: nil) = 2 /\ rk (A :: E :: F :: G :: nil) = 2 /\
rk (A :: I :: J :: M :: nil) = 2 /\ rk (A :: H :: K :: L :: nil) = 2 /\
rk (B :: E :: H :: I :: nil) = 2 /\ rk (B :: G :: J :: L :: nil) = 2 /\
rk (B :: F :: K :: M :: nil) = 2 /\ rk (D :: E :: J :: K :: nil) = 2 /\
rk (C :: E :: L :: M :: nil) = 2 /\ rk (C :: F :: H :: J :: nil) = 2 /\
rk (D :: G :: H :: M :: nil) = 2 /\ rk (D :: F :: I :: L :: nil) = 2 /\
rk (C :: G :: I :: K :: nil) = 2.

(* Plats pour les plans *)
Parameter rk_planes :
rk (A :: B :: E :: nil) = 3 /\
rk (A :: B :: F :: nil) = 3 /\
rk (A :: B :: G :: nil) = 3 /\
[...] /\ rk (K :: L :: M :: nil) = 3.
end.
```

TABLE II.1.9 – Application du patron au modèle fini $pg(2, 3)$ en utilisant les rangs.

Il est possible de concevoir les modèles de la géométrie synthétique dans un format plus proche de ceux issus de la théorie des matroïdes en décomposant le prédicat booléen d'incidence en deux ensembles. Le premier regroupant toutes les incidences d'un point à une droite, le second rassemblant tous les points non incidents à une droite. Cette décomposition se rapproche plus de la séparation en plats de l'approche matroïdale mais elle est moins efficace lors des preuves lorsqu'il s'agit de mettre en évidence une contradiction. En effet, il est nécessaire d'introduire toutes les hypothèses dans le contexte au lieu de simplement tester le retour du prédicat booléen.

Une fois tous ces modèles exportés dans la syntaxe *Gallina*, nous étudions l'automatisation selon les deux formalisations équivalentes de la géométrie d'incidence projective. Pour cela, nous nous intéressons à la démonstration prouvant que ces géométries finies sont effectivement des modèles de la géométrie d'incidence projective en vérifiant les différents axiomes et que la propriété de Desargues est respectée par les plans finis $pg(n, q)$. Il est possible de vérifier d'autres théorèmes dans ce cadre telle que la propriété de Pappus, mais cette dernière n'apporterait rien de plus par rapport à Desargues. Rappelons qu'en dimension 3, la propriété de Desargues devient un théorème que l'on peut prouver avec les deux approches de la géométrie [Kod14, Kus90, MNS12]. Par conséquent tout espace fini de dimension 3 ou plus vérifie cette propriété.

3 Vérification formelle des modèles et preuve de la propriété de Desargues

Nous utilisons les modèles projectifs finis pour étudier d'une part l'automatisation de preuves de propriétés géométriques et d'autre part la gestion de la complexité dans des démonstrations dont la combinatoire est très élevée. Parallèlement à cette analyse, les deux approches de la géométrie d'incidence projective sont comparées principalement pour leur capacité et leur efficacité à prouver des propriétés mais aussi la facilité avec laquelle les procédés d'automatisation peuvent être mis en place et réutilisés dans les modèles de dimension et d'ordre supérieurs.

Nous démontrons, et ce n'est pas si simple de le faire en pratique, que les axiomes de la géométrie synthétique sont vérifiés pour les plans $pg(2, 2)$, $pg(2, 3)$, $pg(2, 4)$, $pg(2, 5)$, $pg(2, 7)$ et les espaces $pg(3, 2)$, $pg(3, 3)$. De la même manière, nous montrons que les axiomes issus de la théorie des rangs sont respectés pour $pg(2, 2)$, $pg(2, 3)$. Nous détaillons à travers ces exemples quelques méthodes pour gérer la complexité lors de preuves de grande taille dans l'assistant de preuve Coq.

Gérer la complexité dans le cadre des assistants de preuve est quelque chose qui est bien étudiée lors de la formalisation des preuves mais qui n'est pas assez détaillée dans les publications. En effet, la plupart des formalisations dans le domaine se concentre sur la vérification d'un résultat mathématique sans mettre en avant les performances en temps d'exécution et en mémoire. L'objectif principal est d'établir la preuve, si bien que la lisibilité, la segmentation de la preuve, les performances globales ou la facilité à formuler et prouver un résultat sont des facteurs moins considérés. De nombreux outils très efficaces sont intégrés aux assistants de preuve [BTT15, Chl13, GL02, TG15], pour faciliter le développement des preuves, améliorer les temps d'exécution et diminuer l'occupation mémoire. Néanmoins, il arrive des situations où, même si la démonstration est finement découpée et optimisée, dominer la complexité de la preuve reste un challenge et l'objectif principal est alors mis en défaut.

Ce chapitre permet d'observer certaines de ces situations limites où les facteurs tels que la quantité d'hypothèses, le choix des tactiques et le nombre de configurations à vérifier rendent le problème difficile à résoudre pour l'utilisateur. Ces trois facteurs peuvent être controlés afin de limiter le plus possible leur impact sur la complexité des preuves considérées. Nous identifions pour cela plusieurs critères à travers les différents exemples de modèles ; ces critères peuvent influencer drastiquement la complexité dans les preuves. L'optimisation simple mais pertinente de ces critères généraux permet d'obtenir efficacement des preuves dans un contexte hautement combinatoire. La mise en place de chacun de ces critères permettant d'améliorer l'automatisation des preuves est dans la plupart des cas réalisée à la main en suivant notre intuition.

3.1 Gestion de la complexité

Toutes les optimisations présentées dans cette partie sont indépendantes les unes des autres et permettent de démontrer des lemmes où l'explosion combinatoire du nombre de cas à prouver est omniprésente.

3.1.1 Analyse de cas

Le premier critère très impactant qu'il est primordial d'étudier est l'explosion du nombre de cas à analyser. Pour prouver un énoncé géométrique en géométrie finie, la démarche classique consiste à vérifier toutes les configurations possibles de ce théorème, c'est-à-dire qu'il faut effectuer une analyse de cas à la fois sur les points et les droites du modèle. Le plus souvent, une approche de force brute mène à une multitude de cas à traiter rendant la preuve impossible. Illustrons ce problème d'analyse de cas sur la propriété *Uniqueness* **(A3P2)** de la géométrie

d'incidence projective (voir II.1.10) :

```
(* Propriété d'unicité en géométrie synthétique *)
Lemma uniqueness : forall A B : Point, forall l m : Line,
  Incid A l -> Incid B l -> Incid A m -> Incid B m -> A=B \/ l=m.
```

TABLE II.1.10 – Rappel de la propriété d'unicité de la géométrie synthétique.

En considérant le plus petit plan fini $pg(2, 2)$ contenant 7 points et 7 droites, une analyse de cas basique aboutit à $7^4 = 2\,401$ cas à traiter. En passant au plan projectif d'ordre 3 contenant 13 points et 13 droites, nous devons gérer $13^4 = 28\,051$ cas. Ces situations ne sont pas encore critiques et de telles preuves sont encore facilement réalisées. Cela devient néanmoins plus compliqué en traitant par exemple un plan d'ordre 5 contenant 31 points et droites dans lequel 923 521 cas sont à étudier. De manière générale, soit q l'ordre du plan projectif, alors nous devons analyser $(q^2 + q + 1)^4$ combinaisons possibles, ces preuves aboutissent seulement pour q petit. Cette analyse pour la force brute utilisée pour étudier toutes les configurations doit être optimisée le plus possible en utilisant les hypothèses du contexte et des techniques d'automatisation.

3.1.2 Formulation et choix de la théorie

Un deuxième facteur influençant fortement la complexité est la manière dont les énoncés sont formulés. Ces questions sont bien connues et étudiées dans la théorie de la complexité spécialement dans les problèmes SAT [MMZ+01, NOT06]. Des critères tels que la taille des clauses, le nombre de propositions et l'ordre des propositions a un impact significatif sur le temps de calcul d'une preuve. Pour illustrer cette importance dans notre cadre, considérons deux définitions de l'existence de l'intersection dans le plan entre deux droites (voir Table II.1.11), la première exprimée en géométrie synthétique, la seconde définie à partir des rangs.

```
(* Propriété d'existence de l'intersection dans le plan en géométrie synthétique *)
Lemma point_existence : forall (l1 l2 : Line),
  exists A : Point, Incid A l1 /\ Incid A l2.
```

```
(* Propriété d'existence de l'intersection dans le plan avec les rangs *)
Lemma rk_inter : forall A B C D : Point,
  exists J, rk(triple A B J) = 2 /\ rk(triple C D J) = 2.
```

TABLE II.1.11 – Différence de formulation de la propriété d'existence d'une intersection entre les deux approches de la géométrie d'incidence projective.

En considérant le plan fini d'ordre 3, l'analyse de cas dans la première description génère $13^2 = 169$ configurations avant de fournir un témoin pour le quantificateur existentiel alors que dans la deuxième formulation nous faisons face à $13^4 = 28\,051$ cas. Il serait nécessaire dans cette situation de créer une méthode de résolution de la formule existentielle cent fois plus rapide avec l'approche matroïdale pour obtenir un temps d'exécution similaire à celui de la géométrie synthétique. Nous observons ici que le choix de la géométrie synthétique semble plus prometteur que la théorie des matroïdes puisque l'expression de la même définition nécessite

deux quantificateurs universels en moins. Choisir la théorie la plus adaptée pour formuler un énoncé permet de rendre certaines preuves traitables.

Ce choix ne résout pas tous les problèmes de complexité. La meilleure façon de traiter proprement l'explosion combinatoire du nombre de cas causée par une succession d'analyses de cas est d'élaguer l'arbre de preuve le plus tôt possible.

3.1.3 Élagage de l'arbre de preuve

Considérons à nouveau la propriété *Uniqueness* **(A3P2)** de la géométrie d'incidence projective et sa preuve dans le plan projectif $pg(2, 3)$ dans la Table II.1.12. Une analyse de cas naïve sans élagage ou gestion des quantificateurs aboutit aux 28 051 cas expliqués précédemment. La preuve se décompose en deux parties : tout d'abord une induction est effectuée sur chacun des quantificateurs universels pour considérer les 13^4 configurations avant de résoudre tous ces buts par la mise en évidence d'une contradiction ou d'une reflexivité entre les hypothèses et la partie gauche ou droite de la disjonction du but de la propriété d'unicité. Une exécution force brute de cette preuve sur la machine standard [3] prend approximativement 40 secondes dans cette situation.

```
(* Preuve de l'unicité en géométrie synthétique *)
Lemma uniqueness : forall A B : Point, forall l m : Line,
  Incid A l -> Incid B l -> Incid A m -> Incid B m -> A=B \/ l=m.

Proof.
  induction A;induction B;induction l;induction m;
  try discriminate; try (left;reflexivity);try (right;reflexivity).
Qed.
```

TABLE II.1.12 – Exemple d'élagage dans la preuve de la propriété d'unicité de la géométrie synthétique.

Des stratégies plus élaborées et liées au contexte sont nécessaires pour assurer que la preuve reste praticable pour un ordre ou une dimension supérieur. La variable A est liée à l dans l'hypothèse `Incid A l`. Il est alors possible d'élaguer l'arbre de preuve dès la première induction sur la droite l quand le point A n'est pas incident à la droite l. En réordonnant ainsi les quantificateurs, nous tirons avantage des hypothèses de l'énoncé géométrique. Toutes les hypothèses de cet énoncé sont symétriques et possèdent le même pouvoir d'élagage. Plus précisément le nombre de buts qui peuvent être éliminés après l'analyse de cas sur A suivi de l sera identique à l'ordonnancement B puis m ou m puis B. Une autre amélioration notable consiste à valider directement la partie gauche des buts après l'induction sur B quand l'égalité $A = B$ (autrement dit la partie gauche de la disjonction) est vérifiée. En effet, il n'est pas utile de continuer l'analyse de cas sur la variable m si le but peut déjà être validé. On évite ainsi la duplication d'un but en 13^x cas avec x le nombre d'analyse de cas restantes à effectuer si celui-ci peut être validé ou éliminé par contradiction. En appliquant ces deux ajustements dans la Table II.1.13, nous construisons la preuve de l'axiome d'unicité en moins d'une seconde avec au plus 400 buts à prouver à chaque niveau de la démonstration.

3. Spécificités de cette machine : Intel(R) Core(TM) i5-4460 CPU @3.20GHz avec 16Go de mémoire

```
(* Preuve de l'unicité en géométrie synthétique *)
Lemma uniqueness : forall A B : Point, forall l m : Line,
  Incid A l -> Incid B l -> Incid A m -> Incid B m -> A=B \/ l=m.

Proof.
  induction A;induction l;try discriminate;
  induction B;try discriminate;try (left;reflexivity);
  induction m;try discriminate;try (right;reflexivity).
Qed.
```

TABLE II.1.13 – Exemple d'élagage amélioré dans la preuve de la propriété d'unicité de la géométrie synthétique.

3.1.4 Hypothèses les plus restrictives

La sous-section précédente montre que l'ordonnancement des quantificateurs en fonction des hypothèses de l'énoncé peut avoir un impact fort sur la taille de l'arbre de preuve. Autrement dit, l'ordre dans lequel l'analyse de cas est effectuée doit dépendre du pouvoir d'élagage des différentes hypothèses. L'idée consiste à considérer le plus tôt possible les hypothèses qui sont les plus restrictives pour élaguer au maximum l'arbre de preuve du plus grand nombre de cas possibles. Considérons les deux hypothèses suivantes $A \neq B$ et `Incid A l` dans le modèle fini $pg(2, 3)$. En effectuant une analyse de cas sur les deux variables de chacune des hypothèses, nous obtenons que la première hypothèse permet d'éliminer 13 cas sur 169 alors que dans le second contexte il est possible de mettre en évidence une contradiction dans 117 cas sur les 169. On comprend l'intérêt de prendre l'hypothèse d'incidence en priorité pour éliminer rapidement des buts. S'il est facile de mesurer si une hypothèse est plus élagante qu'une autre (elles peuvent avoir le même pouvoir d'élagage), il est bien plus difficile d'appliquer ce principe en considérant des paquets d'hypothèses qui ont une arité non identique et des variables qui sont différentes. On doit dans ce cas essayer de planifier l'orthogonalité de ces différentes hypothèses pour élaguer l'arbre de preuve.

3.1.5 Existence de témoin

Certaines propriétés ne sont pas uniquement exprimées à partir de quantifications universelles, elles contiennent aussi une ou plusieurs quantifications existentielles. Pour résoudre ces dernières, il faut fournir un témoin d'existence qui est l'un des objets (point ou droite) déjà construit du modèle. Il est possible de trouver naïvement ce témoin en testant tous les objets du modèle. La quantification est alors traitée comme une quantification universelle supplémentaire complexifiant ainsi le calcul de la preuve, ce qui n'est pas désirable. Nous allons montrer comment il est possible de construire un témoin plus efficacement en utilisant le calcul d'une fonction. Considérons l'exemple suivant en dimension 3 qui est la propriété d'*Upper-Dimension* **(A6P3)** ; celle-ci indique qu'il existe toujours une quatrième droite coupant trois autres droites non nécessairement coplanaires (voir II.1.14).

```
(* Propriété de dimension supérieure en géométrie synthétique *)
Lemma upper_dimension : forall l1 l2 l3 : Line,
  ~ l1 = l2 /\ ~ l1 = l3 /\ ~ 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).
```

TABLE II.1.14 – Rappel de la propriété d'*Upper-Dimension* en géométrie synthétique.

Pour construire le témoin d'existence de la quatrième droite `exists l4 : Line`, nous utilisons le résultat d'une fonction intermédiaire qui calcule automatiquement la droite témoin intersectant les trois droites fournies en paramètre. Cette fonction II.1.15 est construite dès la génération du modèle à partir des informations de ce dernier. Nous réduisons ainsi la recherche d'un témoin d'existence à un simple calcul de fonction sans complexifier l'analyse de cas. À titre d'information dans le plus petit espace projectif $pg(3, 2)$, une telle fonction possède 42 875 entrées.

```
(* Fonction calculant la droite qui coupe trois droites quelconques données *)
Definition f_upper_dimension (l1:Line) (l2:Line) (l3:Line) :=
match l3 with
 | L0 => match l2 with
        | L0 => match l1 with
                  | L0 => (L0,(P0,P0,P0))
                  ...
                end
        end
end.
```

TABLE II.1.15 – Fonction qui calcule une droite témoin pour la propriété d'*Upper-Dimension* en géométrie synthétique.

3.1.6 Preuves comme des programmes

Les techniques précédentes deviennent encore plus importantes lorsque nous traitons des espaces projectifs finis de dimension 3. Plaçons nous dans l'espace fini $pg(3, 2)$ contenant 15 points et 35 droites et prouvons dans ce modèle la propriété de *Pasch* **(A2P3)** indiquant que si deux droites sont coplanaires alors elles se coupent nécessairement dans le plan (voir II.1.16).

```
(* Propriété d'existence de l'intersection entre deux droites coplanaires *)
Lemma pasch : forall A B C D : Point, forall lAB lCD lAC lBD : Line,
  ~ A = B /\ ~ A = C /\ ~ A = D /\
  ~ B = C /\ ~ B = D /\ ~ C = D ->
  Incid A lAB /\ Incid B lAB ->
  Incid C lCD /\ Incid D lCD ->
  Incid A lAC /\ Incid C lAC ->
  Incid B lBD /\ Incid D lBD ->
  (exists I : Point, (Incid I lAB /\ Incid I lCD)) ->
  (exists J : Point, (Incid J lAC /\ Incid J lBD)).
```

TABLE II.1.16 – Rappel de la propriété de *Pasch* en géométrie synthétique.

Une analyse de cas force brute sur un tel énoncé mène à $15^4 \times 35^4 = 75\ 969\ 140\ 625$ configurations à traiter. Les simplifications pour limiter la taille de l'arbre de preuve, et l'ordre dans lequel les inductions sont effectuées ne sont plus suffisantes pour obtenir une preuve praticable. Pour simplifier la combinatoire dans cette situation, nous profitons de la correspondance de Curry-Howard [4] consistant à considérer les preuves comme des programmes.

L'idée consiste à éliminer l'analyse de cas sur les droites en utilisant l'information générée par l'analyse de cas sur les points que l'on effectue préalablement. Nous souhaitons construire un lemme intermédiaire qui fournit la droite effective passant par deux points distincts donnés. Pour élaborer ce lemme, il est possible d'utiliser l'approche précédente en construisant dès la génération du modèle une fonction qui calcule cette droite unique. L'autre possibilité retenue, qui exploite la correspondance de Curry-Howard, consiste à remarquer que cette fonction est une version simplifiée indépendante de la propriété *Line-Existence* **(A1P3)** qui peut être utilisée directement comme un programme. En effet, l'assistant de preuve Coq permet de construire des fonctions complètement spécifiées à partir d'une preuve.

Dans la Table II.1.17, nous définissons donc la fonction `line_from_points` qui extrait à partir de la preuve de la propriété *Line-Existence* la droite passant par les deux points distincts T et Z. Cette fonction sert ensuite dans la construction et la preuve du lemme `points_line`. Grâce à ce lemme, il est maintenant possible d'extraire x en utilisant une tactique lors de la preuve de la propriété de *Pasch*. De cette manière, nous obtenons les droites lAB, lCD, lAC et lBD sans analyse de cas, et aucun but supplémentaire n'est généré. Nous réduisons ainsi le nombre de configurations à vérifier à seulement $15^4 = 50\ 625$ cas, avant d'effectuer l'élimination de l'hypothèse existentielle `exists I : Point, (Incid I lAB / Incid I lCD)`.

4. Il existe un lien profond entre déduction naturelle et le lambda-calcul typé. Cette correspondance joue un rôle clé en logique en établissant un pont entre la théorie de la démonstration et l'informatique théorique. Dans cette approche, démontrer un théorème est identique à écrire un programme ; énoncer un théorème revient à examiner la spécification partielle d'un programme avec son typage.

```
(* Fonction calculant la droite unique passant par deux points *)
Definition line_from_points (Z : Point * Point) :=
  proj1_sig (line_existence (fst Z) (snd Z)).

(* Preuve de la propriété points_line à partir de la fonction line_from_points *)
Lemma points_line : forall T Z : Point, forall x : Line,
  Incid T x -> Incid Z x -> ~ T = Z -> x = (line_from_points(T, Z)).
```

TABLE II.1.17 – Utilisation de la correspondance de Curry-Howard en géométrie synthétique.

3.1.7 Pseudo-recherche en profondeur

Dans des arbres de preuves où la ramification est trop importante, quand les optimisations précédentes ne sont plus suffisantes (parce que la consommation mémoire est trop grande), nous adaptons l'algorithme classique de parcours en largeur des buts en Coq. Considérons la preuve suivante `tac1;tac2;tac3;`, par défaut le système Coq applique la tactique 1 sur tous les buts courants, puis la tactique 2 sur tous les buts générés par la tactique 1 et enfin la tactique 3 s'exécute sur tous les buts issus de l'application de la tactique 2. Naturellement, le système expose tous les buts courants après l'application de chacune des tactiques et sauvegarde le contexte de chacun d'eux à chaque étape de la démonstration.

L'idée consiste à ne pas construire toutes les ramifications de l'arbre de preuve directement, en essayant de résoudre séquentiellement chacune des branches principales de l'arbre de preuve avant de considérer la suivante. La méthode de résolution d'une branche est ainsi répétée sur chacune des branches dès leur construction. Pour faire cela, nous exploitons le mécanisme d'associativité à droite pour réaliser une pseudo-recherche en profondeur afin de limiter le nombre de cas à considérer à chaque niveau de la démonstration `tac1;(tac2;tac3)`. Le système applique la tactique 2 puis 3 sur le premier but créé par la tactique 1 avant de passer au but suivant.

3.1.8 Relation d'ordre sur les objets

Dans la plupart des propriétés qui sont prouvées, les points impliqués doivent être nécessairement distincts. Afin de diminuer un peu plus le nombre de cas à étudier, nous ajoutons une relation d'ordre qui est construite dès la génération du modèle en associant un entier à chacun des points du modèle (un procédé similaire est appliqué pour les droites). Nous modifions aussi légèrement les énoncés des propriétés en ajoutant des hypothèses d'ordre sur les points permettant d'éliminer directement les cas où l'ordre entre les points n'est pas respecté. Nous exploitons de cette manière la symétrie des énoncés.

Nous prouvons ainsi la propriété de *Pasch* **(A2P3)** en utilisant un lemme intermédiaire intégrant cette relation d'ordre sur les points. Pour cela, deux hypothèses contenant le prédicat `leP`, qui indique que le premier point doit toujours avoir un indice plus petit que le second, sont ajoutées au lemme intermédiaire de la Table II.1.18. Nous observons la symétrie de cet énoncé en remarquant que s'il est vrai pour $A = a$, $B = b$, ... $D = d$ alors il est aussi vrai pour $A = b$, $B = a$, ... $D = d$. Une fois la démonstration de ce lemme finie, nous prouvons la propriété générale de *Pasch* en utilisant les symétries.

En reprenant le modèle $pg(3, 2)$ possédant 15 points, le nombre de cas par défaut qui est 15^4 peut être ainsi réduit à $15^2 \times 14^2$. La vérification d'une hypothèse d'ordre est triviale et immédiate ; elle est le résultat d'une fonction booléenne précisant si l'ordre entre les deux entiers est respecté. Si l'ordre n'est pas respecté, le but courant est directement éliminé.

```
(* Propriété d'existence de l'intersection entre deux droites coplanaires *)
Lemma line_existence_order : forall A B C D : Point, forall lAB lCD lAC lBD : Line,
  leP A B -> leP C D ->
  [...].
```

TABLE II.1.18 – Exemple de relation d'ordre dans la propriété de *Pasch* en géométrie synthétique.

3.1.9 Ingénierie de la preuve

Le dernier aspect purement technique auquel nous nous intéressons est le génie logiciel dans l'assistant de preuve Coq. Toutes les tactiques ou mot-clés définis dans cette partie sont détaillés dans le manuel d'utilisation Coq et le livre de référence [Coq02]. Une fois que toutes les simplifications sont réalisées en élaguant l'arbre dès que nécessaire, en minimisant le nombre de cas à traiter ou en créant les témoins pour les quantificateurs existentiels, il reste à définir la séquence de tactiques [Del00] qui va résoudre tous les buts avec le plus d'efficacité possible. Bien souvent, il existe de multiples combinaisons de primitives Coq qui permettent de mener à son terme une preuve. La question est de trouver un arrangement de tactiques suffisamment puissant pour résoudre tous les buts qui sont fortement similaires, mais il faut aussi que celui-ci soit peu coûteux en temps et en mémoire. Pour y parvenir, nous énonçons quelques principes généraux à suivre en élaborant cette combinaison de tactiques.

L'idée principale consiste à fournir au système le plus d'informations possibles afin de simplifier le mécanisme d'unification [Zil14]. Lorsqu'une tactique est conçue par nos propres soins, nous connaissons la classe de problèmes qu'elle est susceptible de résoudre et les informations dont elle aura besoin pour y arriver. En effet, dès qu'il est possible de donner des paramètres à la tactique ou d'apporter une indication sur les hypothèses que le système doit rechercher dans le contexte, le système peut conclure bien plus vite.

Considérons l'exemple suivant : si pour simplifier le but courant le système doit trouver une hypothèse indiquant que deux points ne sont pas égaux, alors il est bien plus évident pour Coq de trouver cette hypothèse si les points qui doivent être trouvés sont passés en paramètre et que le motif de l'hypothèse est déjà décrit au sein de la tactique. Parfois il n'est pas toujours faisable d'être aussi précis, la tactique doit rester assez générale pour simplifier tous les buts ayant un motif similaire. Il faut faire attention à ne pas trop alourdir en informations une tactique afin que cette dernière soit performante dans la résolution de sa classe de problèmes (voir Table I.2.17 et I.2.18).

Dans ce même sens, l'utilisateur doit éviter les tactiques de hauts niveaux déjà intégrées au système qui permettent de simplifier une large classe de problèmes plus grande que la configuration que l'on cherche à résoudre. Ces dernières priorisent toujours le succès de la tactique par rapport à son coût.

Cependant développer des tactiques en Coq pour effectuer une automatisation massive est une tâche ardue. Plus elles doivent être efficaces, plus cette tâche est difficile. Le profiler *Ltac* [TG15] apporte une aide précieuse pour localiser les simplifications les plus coûteuses. Un utilisateur expert de Coq peut être habitué à fortement utiliser des primitives tel que `rewrite`, `intuition` ou `omega`. Ces tactiques permettent de résoudre une majorité de problèmes sans nécessairement s'inquiéter de l'impact sur les performances. C'est pourquoi, il est important de revenir aux tactiques les plus élémentaires du système Coq couplées à des lemmes intermédiaires.

Nous comparons l'application de deux tactiques classiques du systèmes Coq appliquant des substitutions en cas d'égalité dans un ensemble d'hypothèses grâce au profiler *Ltac* dans la Table

II.1.19. Cet exemple de retour du profiler en remplaçant la tactique `intuition` par la commande `subst` montre qu'il est possible d'obtenir le même résultat en divisant le temps global d'exécution par un facteur 50. Ce résultat s'explique simplement par le fait que la tactique `intuition` possède bien plus de manières de clôturer un but courant. Le profiler *Ltac* devient rapidement un outil incontournable pour mesurer le temps d'exécution moyen des tactiques qui sont élaborées tout en estimant l'impact que peut avoir un simple changement dans le design d'une tactique sur les performances globales.

(* Tacticque intuition *)	local	total	calls	max
-case_clear_1 ----------	X%	100.0%	1	3.120s
--<Coq.Init.Tauto.with_uniform_flags> -	X%	99.5%	7	0.456s
--t_tauto_intuit ----------	X%	99.5%	7	0.456s

(* Tacticque subst *)	local	total	calls	max
-case_clear_2 ----------	X%	100.0%	1	0.060s
--subst ----------	X%	40.0%	7	0.012s

TABLE II.1.19 – Exemple de retour du profiler *Ltac* en appliquant deux tactiques sur le même contexte.

Un autre aspect non négligeable à considérer au sein des preuves est le changement de contexte. Par défaut, le système refuse de passer au but courant suivant tant que ce dernier n'est pas prouvé. On a observé qu'il est bénéfique d'élaguer l'arbre de preuve le plus tôt possible et d'éliminer rapidement certains buts triviaux. Dans cette optique, l'utilisateur peut tester une séquence de tactiques sur tous les buts grâce au mot clé `try`. Lorsque cette séquence échoue, le système change de contexte et essaye d'appliquer cette dernière sur le but suivant. Le système doit revenir tôt ou tard dans le contexte du premier but qui n'a pas encore été prouvé. Pour éliminer ce côut de transition entre les différents buts, il est avantageux de résoudre le but dès la première fois où il est rencontré grâce aux mots clés `first` et `solve`. La tactique `solve` indique au système que si l'arrangement de tactiques passé en paramètre ne résout pas le but, le système ne doit pas continuer l'exécution de la preuve. Le mot clé `first` quant à lui permet d'indiquer que le système doit appliquer chacune des séquences de tactiques entre crochets dans l'ordre séparés par des pipes jusqu'à la résolution du but. Si aucune de ces combinaisons n'est concluante, le système s'arrête et renvoie une erreur.

En complément, nous travaillons avec la tactique `abstract` qui permet de prouver un but comme si celui-ci était un lemme séparé afin de diminuer la taille de la preuve courante en mémoire. Nous simplifions ainsi la structure globale du terme de preuve qui peut devenir trop conséquente et nous facilitons la vérification de ce dernier à la fin de la preuve. Finalement, pour paralléliser la preuve dès que l'occupation mémoire le permet, nous dupliquons les instances de Coq grâce au mot clé `par` en parallélisant la résolution des buts.

3.2 Automatisation de la preuve de Desargues

Il est bien connu que les plans projectifs $pg(2, q)$ sont désarguésiens étant donné leur construction via un corps. Nous prouvons que ces plans finis d'ordre 2 et 3 sont désarguésiens avec les deux approches de la géométrie.

Pour automatiser et prouver la propriété de Desargues dans des modèles finis, toutes les techniques présentées dans les sous-sections précédentes deviennent encore plus essentielles. La propriété de Desargues (voir l'Annexe C) s'exprime à partir de 10 points et 10 droites. Les trois derniers points et la dernière droite sont construits automatiquement à partir du reste de la configuration. Une analyse de cas dans le plan projectif $pg(2, 3)$ sur les 7 premiers points produit 62 748 517 configurations à traiter sans élagage. L'approche s'appuyant sur les rangs est bien plus efficace dans ce cadre puisqu'elle permet d'éliminer directement les quantificateurs universels sur les droites sans devoir extraire les différentes droites en s'aidant de l'information sur les points. La preuve de cette propriété est néanmoins possible à condition de correctement élaguer l'arbre de preuve, de méthodiquement découper la preuve et d'exploiter la géométrie.

Pour simplifier la démonstration de la propriété de Desargues, nous tirons largement profit des nombreuses symétries du problème mais aussi des conditions de non-dégénérescence pour éliminer les cas non désirables afin de limiter l'explosion combinatoire.

En premier lieu, nous nous servons de la symétrie du problème vis à vis du centre de perspective. En fixant ce centre avec un des points du plan fini et en prouvant que la permutation des points dans le modèle fini reste encore un modèle, il est possible de vérifier la propriété de Desargues peu importe le centre de perspective qui a été choisi. Nous démontrons ainsi que la propriété de Desargues est vraie avec le centre fixé sur un premier point du modèle. Nous éliminons de cette manière un quantificateur universel de l'analyse de cas. Les autres points du modèle comme centre de perspective sont ensuite prouvés grâce aux différentes permutations du modèle en utilisant un foncteur échangeant les différents points.

La deuxième symétrie dont nous nous aidons pour décomposer le problème provient de la permutation des droites qui sont concourantes au niveau du centre de perspective. Soit A le centre de perspective ; il est possible de spécifier dès l'énoncé des droites distinctes contenant ce point A et permettant de former les deux triangles qui sont en perspectives. Ces droites ainsi contraintes deviennent des hypothèses très restrictives avec un pouvoir d'élagage conséquent. Par la suite, nous montrons que la permutation de ces droites satisfait toujours la propriété.

Finalement, il est important de bien considérer les conditions de non-dégénérescences de notre énoncé géométrique. Dans le cas de la propriété de Desargues, il est possible de traiter une propriété plus générale où les deux triangles peuvent partager jusqu'à deux points. Cette propriété aboutit à une contradiction dans la spécification de la droite $\alpha\beta\gamma$ (plusieurs droites sont confondues). En restreignant la propriété au cas où les triangles ont au maximum un point en commun, nous éliminons approximativement un tiers des buts à prouver à chaque niveau de la démonstration.

Toutes ces optimisations en incluant les techniques décrites précédemment permettent de rendre les preuves traitables tout en améliorant nettement les performances. La première preuve globale de Desargues qui a été conçue pour le modèle $pg(2, 2)$ nécessitait 11 minutes pour être vérifier ; à la fin de cette thèse en utilisant la même configuration une telle preuve n'a besoin que de 30 secondes.

3.3 Résultats

Nous avons vérifié que des espaces finis $pg(n, q)$ de dimension 2 et 3 sont réellement des modèles de la géométrie projective et nous prouvons que ces modèles sont aussi désarguésiens. Ces résultats sont obtenus en utilisant les deux formalisations de la géométrie que nous comparons : la géométrie synthétique et l'approche par les rangs. Globalement, ce développement représente 440 000 lignes de spécifications engendrés dans la majorité automatiquement et 1 500

lignes de preuves pour formaliser les 7 modèles[5]. Tous les résultats sont résumés dans la Table II.1.20. Pour chaque modèle, nous présentons le nombre de lignes de spécifications et de pas de preuves ainsi que le temps d'exécution requis pour compiler ce dernier sur une machine standard.

	Formalisation de la géométrie d'incidence projective					
	avec la géométrie synthétique			en utilisant les rangs		
	Spéc.	Preuve	T. E.	Spéc.	Preuve	T. E.
$pg(2, 2)$ est un modèle	108	44	2s	110	42	16s
$pg(2, 3)$ est un modèle	150	44	7s	297	77	2055s
$pg(2, 4)$ est un modèle	206	44	35s	E. C.		
$pg(2, 5)$ est un modèle	276	44	90s			
$pg(2, 7)$ est un modèle	458	44	7337s			
$pg(2, 8)$ est un modèle	E. C.					
$pg(3, 2)$ est un modèle	10490	192	1440s			
$pg(3, 3)$ est un modèle	420130	250	7623s			
`Desargues` vérifié pour $pg(2, 2)$	188	205	37s	297	162	26s
`Desargues` vérifié pour $pg(2, 3)$	E. C.			2089	386	10700s
`Desargues` vérifié pour $pg(2, 4)$	E. C.					
...						
`Desargues` vérifié pour $pg(3, 3)$						

TABLE II.1.20 – Tests de performance pour plusieurs preuves en géométrie finie réalisés sur notre machine standard. E. C. signifiant « Explosion Combinatoire ».

Rappelons que ces modèles de la géométrie finie sont tous formalisés en suivant un patron quasiment identique qui est adapté selon la dimension et l'ordre. Les propriétés qui sont prouvées dans ce cadre suivent exactement la même décomposition et sont optimisées en suivant des procédés identiques quelque soit la formalisation de la géométrie qui est employée.

L'analyse de ce tableau de résultats suggère que la géométrie synthétique est plus efficace que la formalisation s'appuyant sur la notion de rang pour prouver que les espaces finis $pg(n, q)$ de dimension 2 et 3 sont des modèles. Ce résultat s'explique facilement par le critère mentionné précédemment : « la formulation et choix de la théorie ». L'expression de l'axiome *Point-Existence* en géométrie d'incidence plane ne nécessite que deux quantificateurs universels en géométrie synthétique alors qu'il en faut quatre avec de la théorie des matroïdes (voir l'Annexe A et l'Annexe B). Cette différence en nombre de quantificateurs dans l'expression de la propriété fait augmenter significativement la combinatoire lors de la preuve avec les rangs dans les différents modèles $pg(n, q)$ jusqu'à entraîner une explosion combinatoire dès le modèle $pg(2, 4)$. Cette différence se répète à nouveau pour l'axiome *Upper-Dimension* en géométrie d'incidence spatiale (3 quantificateurs universels contre 6) favorisant à nouveau l'utilisation de la géométrie synthétique et provoquant dès $pg(3, 2)$ une explosion combinatoire avec les rangs. Si nous éliminions des modèles en fonction de la dimension, la preuve des deux propriétés *Point-Existence* et *Upper-Dimension*, nous observerions dans le tableau de résultat un léger avantage dans les temps d'exécution pour l'approche matroïdale.

Pour la preuve de la propriété de Desargues, la Table II.1.20 montre que les rangs permettent

5. Pour tester les fonctionnalités de l'extension ssreflect [GM10, MT17] (calcul booléen), nous prouvons aussi que les espaces finis de dimension 3 sont également des modèles de la géométrie d'incidence projective en utilisant cette bibliothèque. Nous choisissons de ne pas détailler les parties techniques issues de ce développement dans ce manuscrit. Notons que l'architecture globale de notre bibliothèque visible dans l'Annexe G inclut néanmoins ces résultats.

d'obtenir un meilleur résultat que la géométrie synthétique. Cette propriété ne nécessite que 10 points avec la théorie des matroïdes pour être exprimé contre 10 points et 10 droites en géométrie classique. Même si les 10 droites peuvent être éliminées de l'analyse de cas en utilisant une fonction déterminant la droite à partir des points, la preuve est plus difficile à établir. En utilisant les symétries du problème et les conditions de non-dégénérescence, nous prouvons que la propriété de Desargues est vraie pour les modèles d'ordre 2 et d'ordre 3 (uniquement avec les rangs). L'explosion combinatoire du nombre de cas à traiter survient très vite dans les deux cas. Nous ne traitons pas la démonstration de la propriété de Desargues en dimension 3 sachant que la preuve générale est disponible dans [MNS09, MNS12].

Nous observons à travers ces résultats que les deux approches sont complémentaires en considérant uniquement l'objectif de la prouvabilité. La théorie la plus performante est celle qui nécessite le moins de quantificateurs pour exprimer le même énoncé géométrique. Pour les propriétés simples avec très peu de points, l'approche synthétique est à son avantage puisqu'elle peut caractériser une droite ou un plan sans passer par la décomposition en points. Lorsque l'énoncé devient plus complexe et que de nombreux points sont impliqués, l'approche matroïdale semble plus prometteuse. Il est possible de se dispenser de la notion de droite et de plan étant donné que les différents ensembles de points du matroïde permettent de prendre en compte ces deux notions.

3.4 Comparaison avec les prouveurs SMT

Les prouveurs « *Satisfiability Modulo Theories* » (SMT) [Fon18, MMZ+01, NOT06] sont des outils permettant de prouver automatiquement des théorèmes exprimés dans un sous-ensemble très expressif de la logique du premier ordre. Ces prouveurs sont des logiciels pointus qui sont en constante mutation pour s'adapter aux nouvelles procédures de décision et aux nouvelles théories. L'évolution très rapide de ces systèmes fait qu'il est dur de vérifier leur bon fonctionnement entraînant des problèmes de confiance dans les décisions qui sont émises. Pour pallier cet inconvénient, de nombreux prouveurs SMT exportent, en plus de la décision, une trace (un témoin de preuve) qui peut êtré vérifiée par un outil externe [AFG+11b, AFG+11a, BP11, FMM+06]. Ce témoin de preuve peut être analysé pour certifier que la décision du prouveur est correcte mais cette preuve est bien souvent incompréhensible et complètement illisible pour un être humain.

Les assistants de preuves sont généralement conçus à partir d'un petit noyau soigneusement élaboré, uniquement composé de quelques centaines de lignes de code qui sont considérées comme sûres. Toute preuve construite dans un de ces assistants interactifs doit être certifiée grâce à ce ce noyau. Pour augmenter grandement la confiance dans les verdicts obtenus par les solveurs SMT, nous pouvons donc nous appuyer sur la vérification des traces construites par ces derniers dans les assistants de preuves. Un prouveur automatique externe qui ne fonctionne pas correctement ou un conduit buggué entre l'assistant de preuves interactif et le solveur a seulement pour conséquence dans le pire des scénarios de produire un échec dans la vérification du certificat. Il est donc impossible de certifier un théorème faux de cette manière. Les deux travaux pionniers concrétisant cette approche pour les solveurs SMT sont les conduits entre *HOL-Light* et *CVC Lite* [Har96, MBG06] et *haRVey* (prédécesseur de *veriT* [BdODF09]) et *Isabelle* [NPW02]. Cependant, l'intégration de ces outils de déductions automatiques en arrière-plan des assistants de preuves nécessite des ajustements importants pour qu'ils puissent coopérer : taille et format des traces, compromis entre trace complète et certificat compressé, gestion des différences théoriques comme la logique sous-jacente ou encore l'interface à déployer entre les deux outils. Des publications plus récentes s'intéressent à l'incorporation d'un ensemble de prouveurs automatiques et solveurs SMT ainsi que du conduit dans les assistants de preuves. Nous pouvons citer non ex-

haustivement pour Coq les plugins *SMTCoq* [EMT+17] et *CoqHammer*[6] [CK18] et pour *Isabelle* l'outil *Sledgehammer* [BP11] qui est directement intégré.

Afin de mieux évaluer nos tests de performances précédents, nous étudions la capacité de certains prouveurs SAT/SMT de la librairie *tptp* [Sut10] à établir que ces géométries finies sont des modèles de la géométrie d'incidence projective et que ces espaces finis respectent la propriété de Desargues en utilisant les systèmes d'axiomes de la géométrie synthétique. Cette librairie permet de tester la résolution d'une conjecture à partir d'axiomes avec 75 prouveurs en utilisant un langage commun s'adaptant au format de chacun des prouveurs. Nous résumons dans la table II.1.21 les décisions globales de l'ensemble des prouveurs sur les deux conjectures étudiées : modèles de la géométrie d'incidence projective et la propriété de Desargues.

	Succès	Échec	Inapproprié
$pg(2,\ 2)$ est un modèle	24	32	19
$pg(2,\ 3)$ est un modèle	23	33	19
$pg(2,\ 4)$ est un modèle	21	35	19
$pg(2,\ 5)$ est un modèle	21	35	19
`Desargues` vérifié pour $pg(2,\ 2)$	6	50	19
`Desargues` vérifié pour $pg(2,\ 3)$	2	54	19
`Desargues` vérifié pour $pg(2,\ 4)$	3	53	19
`Desargues` vérifié pour $pg(2,\ 5)$	2	54	19

TABLE II.1.21 – Résumé des décisions de l'ensemble des prouveurs SAT/SMT de la librairie *tptp* pour deux conjectures géométriques.

Ces deux conjectures sont étudiées basiquement en suivant le même principe que pour les espaces finis. Nous énonçons d'abord les différents espaces finis par extension en décrivant l'ensemble des points, l'ensemble des droites ainsi que toutes les incidences de point à une droite. Puis nous vérifions que ces descriptions permettent de valider chacune des conjectures.

Les prouveurs classifiés dans la catégorie « Inapproprié » ne disposent pas d'un langage conçu pour analyser ce type de conjecture. Tous les autres prouveurs aboutissent soit à un succès dans le temps imparti (15min), soit à un échec parce que le système dépasse le temps ou n'est pas capable de trouver une solution pour le problème traité. Plus de 50% des prouveurs ne permettent pas de démontrer la conjecture recherchée alors qu'ils sont en mesure d'étudier sa prouvabilité. On peut observer que la vérification de la propriété de Desargues est quelque chose de complexe qui est accessible uniquement à quelques prouveurs SMT (*Paradox* [CS03], *Vampire* [KV13] et *Z3* [dMB08, dMKA+15]) à partir de l'ordre 3. Notons que notre expertise dans l'utilisation de la librairie *tptp* est relativement limitée et qu'avec de meilleures connaissances de chaque solveur SMT il est sans doute possible d'obtenir des résultats bien plus satisfaisants. En considérant uniquement les prouveurs dont la décision est un succès pour la conjecture concernée, nous synthétisons dans la Table II.1.22 le temps moyen, le temps médian et le meilleur temps obtenus. Précisons que le meilleur temps d'exécution lors de la vérification de la propriété de Desargues dans le modèle $pg(2,\ 3)$ est un mystère sachant que la méthode pour formuler le plan fini reste strictement identique.

6. Le plugin CoqHammer est un outil prometteur très récent qui n'a pas pu être manipulé dans le cadre de nos travaux sur les géométries finies.

	Temps moyen	Temps médian	Meilleur temps
$pg(2, 2)$ est un modèle	11.75s	0.03s	0.01s
$pg(2, 3)$ est un modèle	14.73s	0.12s	0.01s
$pg(2, 4)$ est un modèle	22.31s	0.15s	0.01s
$pg(2, 5)$ est un modèle	41.11s	0.69s	0.01s
`Desargues` vérifié pour $pg(2, 2)$	237.76s	88.7s	0.33s
`Desargues` vérifié pour $pg(2, 3)$	130.725s	130.725s	36.14s
`Desargues` vérifié pour $pg(2, 4)$	50.14s	3.79s	2.71s
`Desargues` vérifié pour $pg(2, 5)$	286.965s	279.325s	17.95s

TABLE II.1.22 – Tests de performance pour les solveurs SAT/SMT dont la décision est un succès pour deux conjectures géométriques.

Les résultats montrent que les solveurs SAT/SMT sont plus efficaces pour établir que les plans finis sont des modèles de la géométrie d'incidence projective que notre approche à travers l'assistant de preuve Coq. Bien que quelques prouveurs démontrent ce résultat instantanément jusqu'au plan fini d'ordre 5, on peut constater qu'en moyenne les prouveurs ont un ordre de grandeur similaire à l'assistant de preuve Coq à quelques secondes près.

Pour la propriété de Desargues, les quelques prouveurs qui réussissent dans le temps imparti ont besoin de plus d'une centaine de secondes en moyenne pour vérifier cette dernière. Les moyennes et les médianes deviennent peu représentatives à partir du plan d'ordre 3 puisqu'il n'y a que deux prouveurs capables de démontrer cette propriété. Le prouveur Paradox [CS03] est le plus performant dans la librairie *tptp*, il a besoin uniquement de quelques secondes pour valider un résultat complexe nécessitant plusieurs heures en Coq.

Nous comparons les performances des solveurs SMT en décomposant la démonstration de la propriété de Desargues grâce aux symétries décrites précédemment. De cette manière le système voit les symétries comme des conjectures intermédiaires qui peuvent être utilisées comme de nouvelles propriétés une fois qu'elles sont démontrées. Cette division bien qu'efficace au sein de l'assistant de preuve Coq n'est pas très efficiente dans le cas des solveurs SMT. Ces derniers ne sont pas capables de manipuler des lemmes intermédiaires démontrés aussi complexes pour prouver la conjecture suivante. Globalement, pour tous les prouveurs SMT cette décomposition ne fait que rallonger le temps d'exécution pour obtenir une décision.

Sans aucune surprise, les solveurs SMT sont des outils bien plus efficaces pour effectuer des preuves en géométrie finie, lorsque ces derniers sont capables de résoudre notre classe de problème. L'automatisation complète de preuves de formules n'est cependant pas toujours possible dans tous les cas. L'expertise humaine est bien souvent nécessaire pour pouvoir découper une démonstration. L'idée consiste alors à associer la puissance des solveurs SMT aux assistants de preuves. Nous discutons cette perspective dans la conclusion de partie qui suit.

Conclusion : partie II

Bilan

Dans cette partie, nous avons présenté une étude de cas sur l'aide à la preuve au sein de l'assistant de preuve Coq dans le contexte spécifique des géométries finies. Nous commençons par définir le cadre théorique permettant de construire les espaces finis en fonction de l'ordre et de la dimension. Nous nous intéressons ensuite à la construction automatique de modèles de la géométrie finie que nous définissons par extension en énumérant l'ensemble de tous les objets qui les composent : l'ensemble des points, l'ensemble des droites, l'ensemble des plans, l'ensemble des incidences, etc. Une fois ces modèles engendrés et importés dans l'assistant de preuve Coq, nous étudions l'automatisation des démonstrations de ces configurations géométriques spécifiques.

En utilisant deux formalisations équivalentes de la géométrie présentées dans la Partie I, nous prouvons que ces espaces finis sont réellement des modèles de la géométrie d'incidence projective et que ces derniers vérifient de plus la propriété de Desargues. Ces démonstrations sont explorées à travers : les plans finis $pg(2,\,2)$, $pg(2,\,3)$, $pg(2,\,4)$, $pg(2,\,5)$ et $pg(2,\,7)$; mais aussi $pg(3,\,2)$ et $pg(3,\,3)$. À mesure que les espaces finis grandissent, la complexité augmente. Nous devons affiner notre méthodologie d'aide à la preuve pour maîtriser du mieux possible le temps d'exécution et l'utilisation de la mémoire.

Pour cela, nous identifions trois facteurs majeurs : le nombre de buts à traiter, le nombre d'hypothèses dans le contexte et l'imbrication des tactiques choisies. Dans le cas des géométries finies, le nombre d'hypothèses que l'on utilise est un facteur qui ne peut pas être réellement contrôlé. En effet, les modèles sont décrits par extension dans leur intégralité, toutes les informations énumérées permettent de les caractériser et sont utiles lors des démonstrations. C'est pourquoi, l'ensemble des critères que nous présentons dans ce chapitre pour limiter la complexité afin de rendre les preuves praticables sont répartis sur la gestion des deux autres facteurs. Le premier groupe d'optimisations s'intéresse à l'élagage de l'arbre de preuve en utilisant les hypothèses les plus restrictives tout en considérant attentivement la formulation de l'énoncé géométrique. La deuxième catégorie quant à elle se concentre sur le génie logiciel orienté Coq avec l'élaboration de tactiques efficaces utilisant les preuves comme des programmes. Grâce à toutes ces optimisations, il est possible d'envisager le traitement de preuves conséquentes comportant des millions de buts et contenant plusieurs centaines d'hypothèses comme la propriété de Desargues. Mentionnons que les principes introduits dans cette partie sont suffisamment généraux pour être utilisés dans un autre cadre que les géométries finies. Tout le génie logiciel orienté Coq est notamment largement réemployé dans la Partie III.

Un autre aspect qui est analysé ici est la comparaison des deux formalisations lors de l'automatisation des preuves. Uniquement en termes de performances, les deux approches sont complémentaires. Nous avons pu voir à travers le critère du choix de la théorie qu'un énoncé géométrique peut être spécifié à partir de plus ou moins de quantificateurs. L'approche géométrique utilisant le moins de quantificateurs pour décrire une configuration géométrique à prouver est de manière

générale plus efficace lorsqu'il s'agit d'automatiser la démonstration sous-jacente. La géométrie synthétique est donc à son avantage dans les petits énoncés quand elle peut introduire directement une droite ou un plan au lieu de définir plusieurs points. Ce bénéfice s'inverse dans les énoncés de grande taille où la définition des points englobent déjà chacun de ces objets et que la géométrie d'incidence projective classique doit définir chacune des intersections entre les différents objets pour pouvoir construire le problème géométrique. Dans ce cas, il est préférable de tirer profit de la formalisation sur les rangs.

Les deux formalisations de la géométrie d'incidence projective sont équivalentes du point de vue de la mise en place des procédés d'automatisation et leur maintien lors du passage à la dimension supérieure ou l'ordre suivant. Toutes les tactiques conçues sont facilement réemployées dans chacun des espaces finis étudiés et ne nécessitent presqu'aucun ajustement. Nous estimons cependant qu'en considérant un cadre plus général, l'approche matroïdale semble plus prometteuse. La plupart des tactiques développées pour les géométries finies en utilisant les rangs sont plus générales dans leur possibilité de réutilisation et moins adaptées aux géométries finies dans leur design. Pour ce qui est de la lisibilité des preuves, les deux approches permettent d'obtenir des démonstrations très courtes traitant tous les buts au prix d'une génération automatique des modèles, des différentes fonctions d'ordre et d'existence de témoin.

Les performances de l'assistant de preuve Coq sont par la suite évaluées à travers une petite comparaison avec les solveurs SAT/SMT disponibles dans la librairie *tptp* [Sut10]. Les benchmarks obtenus pour ces prouveurs automatiques sont sans surprise meilleurs que les résultats de notre assistant de preuve interactif. Cependant sur les 75 solveurs mis à disposition, très peu sont capables de prouver la propriété de Desargues ; parmi ces derniers uniquement deux solveurs se distinguent (*Paradox* [CS03] et *Vampire* [KV13]) en prouvant la propriété en quelques secondes. Ce temps d'exécution doit être néanmoins pondéré par le fait qu'il est encore nécessaire de valider la décision en analysant la trace qui est éventuellement produite par le solveur.

Perspectives

Une première piste à poursuivre naturellement est l'analyse du temps de vérification des traces produites par les solveurs SMT dans les assistants de preuves qui n'est pas considéré ici. Cependant, cette tâche n'est pas triviale puisqu'à notre connaissance, les deux solveurs principalement concernés lors de la démonstration de la propriété de Desargues n'ont pas encore été combinés à l'assistant de preuve Coq. Nous envisageons d'intégrer le plugin *SMTCoq* pour étudier les traces qui sont construites lors des démonstrations que les modèles finis sont bien des modèles de la géométrie d'incidence projective. Une autre solution plus récente à considérer est l'incorporation du plugin *CoqHammer*.

Parallèlement à cette intégration, la modélisation de ces espaces finis dans l'assistant de preuve *Isabelle* combiné à l'apport de l'outil *Sledgehammer* permettrait d'inclure le solveur *Vampire*. Nous pourrions ainsi parfaitement évaluer la validation du certificat produit par ce prouveur automatique en démontrant la propriété de Desargues. En complément, le développement de ces modèles dans *Isabelle* compléterait notre étude de cas en apportant une comparaison à tous les niveaux entre les deux assistants de preuves : évaluation des facteurs et des critères, différence de formalisation, analyse des benchmarks obtenus.

Toutes les optimisations réalisées uniquement à l'aide de l'assistant de preuve Coq n'ont pas permis d'approcher suffisamment les nombreux problèmes ouverts largement étudiés en géométrie finie. Avec les solveurs SMT qui deviennent des outils puissants et incontournables, il est possible d'envisager la formalisation des plans finis d'ordre 9 de Hugues & Hall [Hal43, RK70] qui sont non désarguésiens. Une autre perspective intéressante est la preuve de non existence d'un plan fini $pg(2, 6)$ et $pg(2, 10)$ [Bos38, LTS89]. Par ailleurs, nous pensons qu'il est possible de

simplifier la description par extension de nos modèles en reconstruisant l'information manquante à partir de l'approche par "spreads & packings" [PW98]. Sans rentrer dans les détails, cette solution alternative permet d'étudier le partionnement de l'ensemble des points sur l'ensemble de droites disjointes.

La piste que nous privilégions en premier lieu et que nous décrivons dans la Partie III de ce manuscrit est l'automatisation des démonstrations en géométrie d'incidence dans un cadre plus général que la géométrie finie. Dans ce but, nous retirons la contrainte qui consiste à décrire la configuration géométrique par extension en énumérant chacun des ensembles dès sa construction. De cette manière, le système ne se contente pas de manipuler les informations qui sont directement à disposition. Ce dernier doit être capable de déduire de nouvelles informations dans un environnement qui n'est pas complètement défini en croisant les différentes hypothèses. Nous réintégrons ainsi la clôture des hypothèses présentée dans la Partie I. Pour limiter les complications supplémentaires qui sont engendrées par l'ajout de la phase de déductions au système, nous nous inspirons du conduit entre solveur SMT et assistant de preuve introduit peu avant. Nous concevons un prouveur en langage C [KRB84] dont la tâche est de produire un certificat contenant le cheminement de la preuve. Ce certificat est ensuite exporté en *Gallina* pour être vérifié et validé par l'assistant de preuve Coq. Nous détaillons le fonctionnement complet de ce prouveur généralisé s'appuyant sur l'approche matroïdale dans la Partie III qui suit.

Troisième partie

Vers un prouveur généralisé de configuration géométrique d'incidence

CHAPITRE III.1

Pipeline du prouveur de configuration géométrique d'incidence

"The reward for being a good problem solver is to be heaped with more and more difficult problems to solve."

R. Buckminster Fuller (1895–1983)

Résumé

Après avoir étudié les performances de l'assistant de preuve Coq pour formaliser des démonstrations dans ces géométries finies définies in extenso, nous nous intéressons à l'automatisation des preuves en géométrie d'incidence projective dans un cadre plus général. Nous présentons dans ce chapitre un prouveur généralisé permettant de résoudre des problèmes d'incidence en géométrie affine et projective. Cet outil d'aide à la preuve permet de suggérer des pistes, de mécaniser et de vérifier automatiquement les raisonnements les plus simples de notre théorie laissant comme tâche à l'utilisateur de guider la preuve en créant de nouveaux points dans l'énoncé géométrique grâce aux axiomes incluant des quantificateurs existentiels.

Nous débutons par la construction d'un prototype pour valider l'automatisation du raisonnement matroïdal sur des théorèmes simples de la géométrie d'incidence (section 1 et 2). Pour permettre de démontrer des théorèmes géométriques plus ardus, nous évaluons par la suite les performances en temps et en mémoire de ce prouveur afin d'optimiser les différentes étapes du pipeline (section 3). Parmi ces optimisations, nous détaillons en particulier une heuristique de coloration améliorant significativement les performances lors de la saturation des hypothèses. Finalement, nous exposons un mécanisme automatique de scission des preuves permettant de considérer des preuves conséquentes (section 4).

Contenu

Ce chapitre présente notre prouveur généralisé de problèmes géométriques d'incidence applicable à la fois en géométrie affine et projective. Celui-ci permet d'engendrer automatiquement une preuve à partir d'un énoncé géométrique que l'on vérifie grâce à l'assistant de preuve Coq. L'algorithme sous-jacent résout l'énoncé en utilisant des règles de réécriture traduisant les axiomes sur les matroïdes. Il permet de résoudre des configurations géométriques complexes tel que le conjugué harmonique et le théorème de Dandelin-Gallucci.

L'étude de cas du chapitre précédent nous a permis de mettre en évidence que certaines preuves géométriques d'incidence sont difficiles à vérifier. Les trois sources majeures de difficulté sont le nombre de buts, la quantité d'hypothèses ainsi que le choix et l'imbrication des tactiques. Pour effectuer la description d'un modèle en géométrie finie, il est nécessaire de décrire ce dernier par extension en énumérant tous les ensembles d'objets.

Nous nous plaçons dans cette partie dans un cadre où le contexte doit être complété en ajoutant des points et des informations supplémentaires pour prouver des théorèmes géométriques. Nous ne disposons pas cette fois-ci de toute la description de la configuration géométrique dès le commencement de la preuve. Notre prouveur devra ainsi d'abord déduire à partir des hypothèses un maximum d'informations sur le problème géométrique pour le résoudre. Cette phase de déduction est une complication supplémentaire trop importante pour envisager une résolution complète et performante directement dans l'assistant de preuve Coq. Nous séparons donc les tâches du prouveur en deux parties : une phase de déduction que nous appelons saturation de la configuration géométrique réalisée dans un langage externe de programmation (langage C [KRB84]) et une phase de vérification de la solution produite par ce langage externe effectuée par Coq.

Dans ce chapitre, nous présentons tout d'abord le fonctionnement global de notre prouveur en décrivant la chaîne de traitements dans son ensemble. Nous détaillons ensuite la mise en place d'un premier prototype permettant de prouver des théorèmes simples en géométrie d'incidence projective. Puis, nous évaluons à travers une étude des performances, les optimisations nécessaires pour prouver des théorèmes de plus en plus complexes que nous présentons avec un catalogue dans le Chapitre III.2.

1 Principe du prouveur par saturation

Nous examinons en premier lieu le principe général de l'algorithme du prouveur avec son pipeline et les règles utilisées jusqu'à saturation par ce dernier. Dans un second temps, nous donnons une description intuitive du bon fonctionnement de l'algorithme en analysant sa terminaison et sa correction. Finalement, nous enrichissons notre prouveur en y introduisant la règle de Pappus.

1.1 Présentation

L'algorithme général du prouveur commence par traduire l'énoncé géométrique et ses hypothèses en un treillis d'inclusion avec des données incomplètes sur les rangs en termes de rang minimum et rang maximum. Un algorithme de saturation est ensuite appliqué pour déterminer toutes les déductions qu'il est possible d'effectuer dans ce treillis en manipulant uniquement les propriétés de la Table I.2.6 (voir l'Annexe B). Pour cela, ces propriétés sont transformées en règles capables de réduire les intervalles de rang possibles. Le prouveur applique ces règles tour à tour sur tous les ensembles de points de la configuration géométrique pour préciser l'intervalle entre le rang minimum et maximum : lorsqu'une nouvelle déduction a lieu, le système fait évoluer le treillis. Le prouveur continue de calculer de nouveaux rangs jusqu'à l'obtention du résultat

recherché ou lorsqu'il n'existe plus aucune déduction à effectuer dans ce contexte. Parallèlement, le prouveur sauvegarde l'application de chacune de ces règles pour pouvoir reconstruire le cheminement des déductions permettant d'établir le nouveau rang d'un ensemble de points. Ce chemin est ensuite extrait sous la forme d'un certificat qui est vérifié et validé par l'assistant de preuve Coq. Nous résumons ce processus de création de preuve dans le pipeline du prouveur de la Figure III.1.1. L'étape de saturation et la sauvegarde du cheminement sont traitées simultanément.

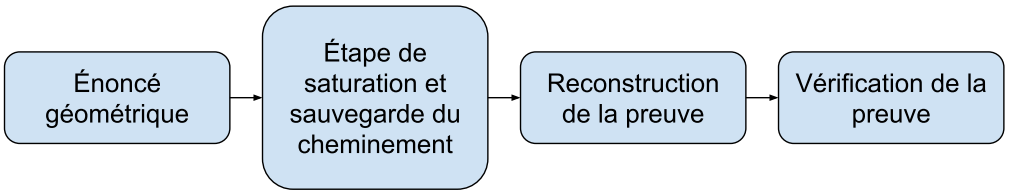


FIGURE III.1.1 – Pipeline du prouveur par saturation.

Nous faisons remarquer au lecteur que ce prouveur permet de traiter des configurations à la fois en géométrie affine et en géométrie projective. Nous apportons à travers ce prouveur un raisonnement qui est en incidence pure. En effet, en considérant uniquement les propriétés matroïdales qui sont indépendantes des axiomes caractérisant l'aspect projectif de notre géométrie, il est possible d'effectuer un ensemble de déductions qui restent vraies à la fois en géométrie affine et projective.

1.2 Création de points

Déjà mentionné dans le Chapitre I.1, la contrainte forte de ce prouveur est le choix de ne pas créer automatiquement de nouveaux objets et en particulier des points. Cette création ne peut être effectuée sans aide ou guidage de la part d'un mathématicien ou d'une intelligence artificielle [Sch19, WCA+14] afin d'éviter une explosion combinatoire ou une boucle infinie.

Il est néanmoins possible de réaliser une saturation sur une configuration géométrique contenant plus d'objets que nécessaire. L'importance de certains de ces objets peut être mesurée en utilisant un découpage géométrique. Si le système privé d'un point aboutit au résultat recherché, le point en question n'est d'aucune utilité pour la résolution de ce théorème.

Cette hypothèse forte s'inscrit parfaitement dans le cadre d'une aide à la preuve avancée qui n'est pas complètement automatique. Nous assistons l'utilisateur dans la tâche fastidieuse qu'est la saturation des hypothèses. Ce même utilisateur garde le contrôle sur la création des objets et le découpage de l'énoncé géométrique en lemmes.

1.3 Règles de réécriture

Comme nous l'avons identifié dès le Chapitre I.2, la majorité des déductions avec l'approche combinatoire de la géométrie d'incidence est effectuée en utilisant les propriétés de la fonction rk de la Table I.2.6 (voir l'Annexe B). Les autres axiomes I.2.9 permettent uniquement de créer de nouveaux objets ou de borner la dimension. Nous éliminons ainsi des règles utilisées jusqu'à saturation tous les axiomes contenant un quantificateur existentiel. Tous les points servant à la résolution du problème doivent donc être créés au préalable pour pouvoir être utilisés dans l'algorithme.

Nous établissons finalement 8 propriétés qui sont une transformation des inégalités présentes dans les axiomes **(A2R2-R3)** **(A3R2-R3)** (4 propriétés sont définies pour chaque axiome). Ces

propriétés s'appuient sur ces deux axiomes pour faire de la réduction d'intervalles, c'est à dire la maximisation du rang minimum et la minimisation du rang maximum. Soient X et Y deux ensembles de points quelconques de l'ensemble de tous les points E, les ensembles X, Y, $X \cup Y$ et $X \cap Y$ doivent vérifier les propriétés décrites dans la Table III.1.1 [1].

(PS1) $X \subseteq Y \subseteq E, rkMin(Y) \geq rkMin(X)$

(PS2) $Y \subseteq X \subseteq E, rkMin(X) \geq rkMin(Y)$

(PS3) $X \subseteq Y \subseteq E, rkMax(X) \leq rkMax(Y)$

(PS4) $Y \subseteq X \subseteq E, rkMax(Y) \leq rkMax(X)$

(PS5) $X, Y \subseteq E, rkMax(X \cup Y) \leq rkMax(X) + rkMax(Y) - rkMin(X \cap Y)$

(PS6) $X, Y \subseteq E, rkMax(X \cap Y) \leq rkMax(X) + rkMax(Y) - rkMin(X \cup Y)$

(PS7) $X, Y \subseteq E, rkMin(X) \geq rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(Y)$

(PS8) $X, Y \subseteq E, rkMin(Y) \geq rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(X)$

TABLE III.1.1 – Propriétés utilisées jusqu'à saturation.

À partir de ces propriétés, nous construisons des règles de réécriture appliquées par le prouveur permettant de mettre à jour l'intervalle des rangs des différents ensembles de points. Chaque règle de réécriture de la Table III.1.2 est associée respectivement à la propriété du même numéro de la Table III.1.1.

1. Les propriétés PS1 et PS2 ou PS3 et PS4 sont identiques lorsqu'on considère une quantification universelle sur les variables X et Y

(RS1) **si** $X \subseteq Y$ et $rkMin(X) > rkMin(Y)$ **alors** $rkMin(Y) \leftarrow rkMin(X)$

(RS2) **si** $Y \subseteq X$ et $rkMin(Y) > rkMin(X)$ **alors** $rkMin(X) \leftarrow rkMin(Y)$

(RS3) **si** $X \subseteq Y$ et $rkMax(Y) < rkMax(X)$ **alors** $rkMax(X) \leftarrow rkMax(Y)$

(RS4) **si** $Y \subseteq X$ et $rkMax(X) < rkMax(Y)$ **alors** $rkMax(Y) \leftarrow rkMax(X)$

(RS5) **si** $rkMax(X) + rkMax(Y) - rkMin(X \cap Y) < rkMax(X \cup Y)$
alors $rkMax(X \cup Y) \leftarrow (rkMax(X) + rkMax(Y) - rkMin(X \cap Y))$

(RS6) **si** $rkMax(X) + rkMax(Y) - rkMin(X \cup Y) < rkMax(X \cap Y)$
alors $rkMax(X \cap Y) \leftarrow (rkMax(X) + rkMax(Y) - rkMin(X \cup Y))$

(RS7) **si** $rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(Y) > rkMin(X)$
alors $rkMin(X) \leftarrow (rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(Y))$

(RS8) **si** $rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(X) > rkMin(Y)$
alors $rkMin(Y) \leftarrow (rkMin(X \cap Y) + rkMin(X \cup Y) - rkMax(X))$

TABLE III.1.2 – Règles de réécriture.

La description de ces règles permet de construire une ébauche du fonctionnement de l'Algorithme par saturation III.1.1 dont le but est de mettre à jour l'intervalle de rangs des différentes parties X et Y.

Algorithme III.1.1 : Algorithme de saturation.

Entrée(s) : Ensemble des parties
Sortie(s) : Ensemble des parties mis à jour par les règles (RS1) à (RS8)

```
1 tant que on peut extraire deux parties X Y et appliquer une règle faire
2     Appliquer la règle sur X et Y
3 fin tant que
```

1.4 Terminaison

Lorsque l'on ne peut plus réduire l'intervalle de valeurs (entre le rang minimum et le rang maximum) de chaque ensemble de points, la saturation est terminée et il ne reste dans ce cas aucune déduction possible à réaliser permettant de faire avancer la résolution si le résultat recherché n'a pas encore été trouvé. L'algorithme vient de détecter qu'aucune modification n'a eu lieu sur les rangs minimum ou maximum lors du dernier parcours de toutes les paires d'ensembles, le système est saturé et toutes les informations déductibles de ce contexte ont été calculées. L'algorithme n'a plus qu'à reconstruire la liste des déductions sauvegardées qui ont été nécessaires pour arriver au système saturé ou au résultat recherché. Finalement, l'assistant de preuve Coq vérifie en un temps fini la liste des pas de preuves générés pour valider ou invalider la démonstration extraite.

1.5 Correction et validation

Pour ajouter ou modifier les hypothèses, l'algorithme doit appliquer les règles de réécriture dérivant immédiatement des axiomes sur les matroïdes. Si une règle modifie le rang minimum

ou maximum d'un ensemble de points, l'application de l'axiome correspondant est notée. Une fois que toutes les déductions possibles ont été effectuées à partir de l'énoncé géométrique, l'algorithme produit un code *Gallina* traduisant les règles de réécriture qui ont été appliquées. Ce code est ensuite importé dans l'assistant de preuve Coq pour être vérifié et validé. Si l'énoncé peut être prouvé en géométrie d'incidence et qu'il est correctement spécifié et complet, nous obtenons une démonstration valide de l'énoncé géométrique. Si l'énoncé est correctement spécifié mais incomplet pour finir la démonstration, nous obtenons une démonstration partielle contenant toutes les déductions qu'il est possible de faire dans ce cadre. Si l'énoncé est incorrect, l'algorithme aboutit à une incohérence lors des déductions ($rkMin > rkMax$ pour un ensemble de points) et se termine par un échec.

1.6 Extension des règles avec la propriété de Pappus

Ce cadre peut être spécialisé en considérant de nouveaux axiomes qui ne doivent pas contenir de quantificateur existentiel. Dans la suite, pour étudier une plus grande variété de théorèmes en géométrie d'incidence projective, nous incorporons la propriété de Pappus dans le prouveur. Cet ajout permet aussi de mesurer l'évolution du prouveur en additionnant des propriétés géométriques qui ne sont pas directement issues des propriétés matroïdales. Cette propriété est ainsi traduite sous forme de règle et incluse dans la boucle des règles de réécriture à utiliser. Si une configuration de Pappus modulo permutation est identifiée et que le résultat (colinéarité de 3 points d'intersection) n'a pas encore été trouvé, le prouveur indique que les points du résultat sont alignés.

2 Implantation du prouveur

Cette section détaille la mise en place du prototype du prouveur en langage C avec les différentes étapes de l'algorithme ainsi que les structures de données qui sont manipulées. Sachant que la saturation possède une complexité exponentielle à la fois en temps et en espace, il n'est pas envisageable de déployer ce prouveur directement dans l'assistant de preuve Coq.

2.1 Initialisation de l'algorithme

Au lancement de l'algorithme, la seule donnée en entrée dont nous disposons est la description faite à la main de la configuration géométrique avec le nombre de points impliqués. Nous initialisons à partir de ce nombre le prouveur en construisant l'ensemble des parties de l'ensemble de tous les points automatiquement. Chaque partie représente un sous-ensemble de points de la configuration géométrique.

Nous encodons de manière automatique chacune de ces parties par un mot de 32 bits où les 28 premiers bits de poids faible représentent des points syntaxiquement distincts de la configuration géométrique. Pour distinguer les points entre eux, nous utilisons habituellement l'ordre lexicographique pour les 26 premiers bits en ajoutant 2 caractères supplémentaires 'a' et 'b' pour représenter le $27^{\text{ème}}$ et $28^{\text{ème}}$ point. Si un point appartient à l'ensemble, le nième bit en partant du bit de poids faible correspondant à la position de la lettre dans l'ordre lexicographique est positionné à 1. Dans l'exemple III.1.2, les bits 1, 2, 5 et 8 sont initialisés à 1 pour représenter dans l'ordre respectivement les points A, B, E, H. Les 4 derniers bits servent à représenter le rang minimum (2 bits) et le rang maximum (2 bits) associé à cet ensemble de points. Le rang peut donc prendre 4 valeurs différentes de 1 à 4 (nous excluons l'ensemble vide) codées par les entiers de 0 à 3 comme illustré dans la Figure III.1.2.

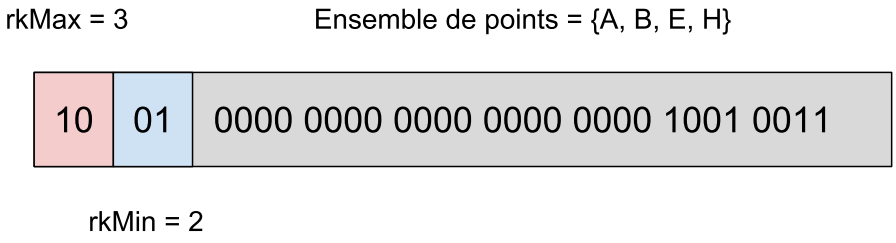


FIGURE III.1.2 – Représentation mémoire d'un ensemble de points avec ses rangs.

Cette structure peut facilement être généralisée avec une architecture 64 bits grâce à la modularité du programme. Notons que la structure pour stocker un ensemble de points avec ses rangs présentée ici n'est pas la plus efficiente possible au niveau de l'occupation mémoire pour le cas général où l'on considère systématiquement l'ensemble de toutes les parties. Nous conservons néanmoins cette structure pour des cas où l'on souhaite effectuer une saturation qui ne s'applique qu'à un sous ensemble de l'ensemble des parties. De plus, la duplication de certaines informations dans l'élaboration de ce prototype permet de plus facilement retrouver l'historique de l'application des règles de réécriture lors de la saturation. Nous détaillons l'occupation mémoire de cette structure à la section 3 où nous montrons également que ça n'est pas une limitation.

Par défaut, nous bornons le rang de chacune des parties X de E avec un rang minimum de 1 (le rang 0 de l'ensemble vide étant exclu) et un rang maximum égal à $min\{|X|, 4\}$ grâce aux axiomes **(A1R2-R3)** et **(A9R3)**. Il s'ensuit l'ajout des informations complémentaires sur les rangs apportées par les hypothèses de l'énoncé géométrique qui sont le plus souvent des rangs exacts. Dans la suite, nous notons **(RS0)** l'application de cette règle pour initialiser les rangs d'un ensemble de points. De nouveaux résultats seront déduits pendant la saturation à partir de ces hypothèses. Nous résumons dans l'Algorithme III.1.2 l'étape d'initialisation.

Algorithme III.1.2 : Étape d'initialisation.

Entrée(s) : Description manuelle de l'énoncé géométrique avec tous les points
Sortie(s) : Ensemble des parties initialisé avec l'application de la règle (RS0)

1 Allocation de l'ensemble des parties à partir de l'ensemble des points E de la configuration géométrique
2 Initialisation des rangs de chacune des parties X de E
3 - Rang minimum initialisé à 1
4 - Rang maximum initialisé à $min\{|X|,4\}$
5 Mise à jour automatique des rangs concernés grâce aux hypothèses de l'énoncé géométrique

2.2 Boucle de saturation

Après l'étape d'initialisation, l'étape de saturation consiste à appliquer les règles à l'ensemble des parties. L'idée générale consiste à garder continuellement l'ensemble des parties dans un état localement correct. Si l'une de ces règles est applicable, on met à jour le rang minimum ou maximum de la partie concernée. Prenons l'exemple suivant, soient $X = \{A, B\}$ et $Y = \{A, B, C\}$ avec l'hypothèse que $rkMin(X) \geq 2$. L'initialisation des ensembles dans le prouveur permet de dire que $rkMin(Y) \geq 1$ et $rkMax(Y) \leq 3$. La propriété **(PS1)** n'est pas vérifiée par l'ensemble X et Y puisque le rang minimum de Y est inférieur au rang minimum de X alors que $X \subseteq Y$.

Le rang minimum de Y est alors mis à jour en indiquant qu'il est au moins supérieur à celui de X en appliquant la règle **(RS1)**.

Pour effectuer l'étape de saturation dans son intégralité, l'Algorithme III.1.3 doit appliquer toutes les règles de réécriture sur toutes les paires d'ensembles de points possibles. Si une règle est déclenchable, l'algorithme l'active et met à jour le minimum ou le maximum de la partie concernée. Pour maintenir la correction de l'algorithme, nous modifions uniquement : un rang minimum en l'augmentant ($\geq$) et un rang maximum en le diminuant ($\leq$) tout en vérifiant que $RkMin \geq RkMax$. Lorsque le rang minimum et maximum d'une des parties sont égaux, nous avons trouvé le rang exact pour cette partie et il ne devrait plus être modifié. S'il advient que le rang $RkMin$ soit strictement supérieur au rang $RkMax$, alors nous sommes en présence d'un contexte incohérent avec des hypothèses contradictoires et l'algorithme se termine par un échec. Dans le cas contraire, l'algorithme continue d'appliquer les règles de réécriture sur les parties, jusqu'à ce que plus aucune modification n'ait lieu lors du dernier parcours complet de toutes les parties. Dès lors la saturation du problème est complète, toutes les déductions possibles à partir des propriétés matroïdales dans ce contexte ont été effectuées.

Algorithme III.1.3 : Étape de saturation.

Entrée(s) : Ensemble des parties initialisé
Sortie(s) : Ensemble des parties mis à jour par les règles (RS1) à (RS8)

```
1  tant que modification au dernier passage faire
2      pour chaque partie X de E faire
3          pour chaque partie Y de E tel que X ≠ Y faire
4              pour chaque règle de réécriture faire
5                  si la règle est activable faire
6                      Appliquer la règle correspondante réduisant l'intervalle des rangs
7                      de la partie concernée en vérifiant que RkMin < RkMax
8                  fin si
9              fin pour chaque
10         fin pour chaque
11     fin pour chaque
12 fin tant que
```

2.3 Mémorisation des déductions

En parallèle de l'étape de saturation, le solveur sauvegarde le cheminement des déductions afin de pouvoir reconstruire ultérieurement une preuve. La production d'une preuve contenant l'intégralité des déductions effectuées lors de la saturation n'est pas pertinente puisqu'une majorité d'entre elles ne sont pas utiles pour établir le ou les résultats recherchés. De plus, nous ne connaissons pas à l'avance les déductions qui sont nécessaires pour conclure la preuve. Notons que ce cheminement n'est pas unique, il dépend de l'ordre dans lequel les règles sont appliquées sur les différentes parties et de l'ordre dans lequel les parties sont sélectionnées.

Dans ce but, un graphe orienté acyclique[2], appelé graphe de déductions (GD), est construit. Chaque noeud de ce graphe représente une partie à laquelle on associe ses rangs courants $RkMin$ et $RkMax$ ainsi que la règle qui a été appliquée pour construire ce noeud. Au départ, nous fabriquons un noeud pour chacune des parties avec ses rangs initialisés grâce à l'application de la règle **(RS0)**. Lorsqu'une des règles fait évoluer l'un des rangs de l'ensemble des parties,

2. En théorie, ce graphe est un hypergraphe où les arêtes sont orientées et étiquetées par les règles de réécritures appliquées.

un nouveau noeud est rajouté au graphe des déductions. Il contient la partie modifiée et ses nouveaux rangs mis à jour en appliquant l'une des règles de la Table III.1.2. Pour rattacher ce noeud au graphe des déductions, nous utilisons le contexte de la règle que l'on vient d'appliquer. La partie est ainsi liée par des arcs indiquant la parenté à toutes les parties qui ont permis d'établir la modification. De cette manière, il est possible de retracer l'évolution de l'intervalle entre le rang minimum et maximum au sein d'une même partie en parcourant le noeud parent possédant la même partie.

Pour illustrer la construction de ce graphe de déductions (GD), nous présentons dans la Figure III.1.3, un exemple d'application de la règle **(RS5)**. Pour des questions de lisibilité, le numéro de la règle appliquée sera positionné systématiquement en dessous du noeud. Ce schéma explique que, si l'intersection d'un plan et d'un point engendre un point alors l'union de ces deux derniers génèrent au maximum un plan (le $rkMax$ de l'union est décrémenté d'une unité). Toutes les déductions suivantes opérées par le système prendront en considération ce résultat si nécessaire. L'information contenue dans le noeud $X \cup Y$ représentant soit un plan, soit un espace est désormais inutile. Une même partie peut apparaître au maximum quatre fois avant de posséder son rang exact final. En effet, la convergence du rang minimum vers le rang maximum est réalisée dans le pire des cas en 3 étapes. Il est alors possible d'estimer l'espace mémoire maximum utilisé pour représenter l'intégralité du graphe de déductions.

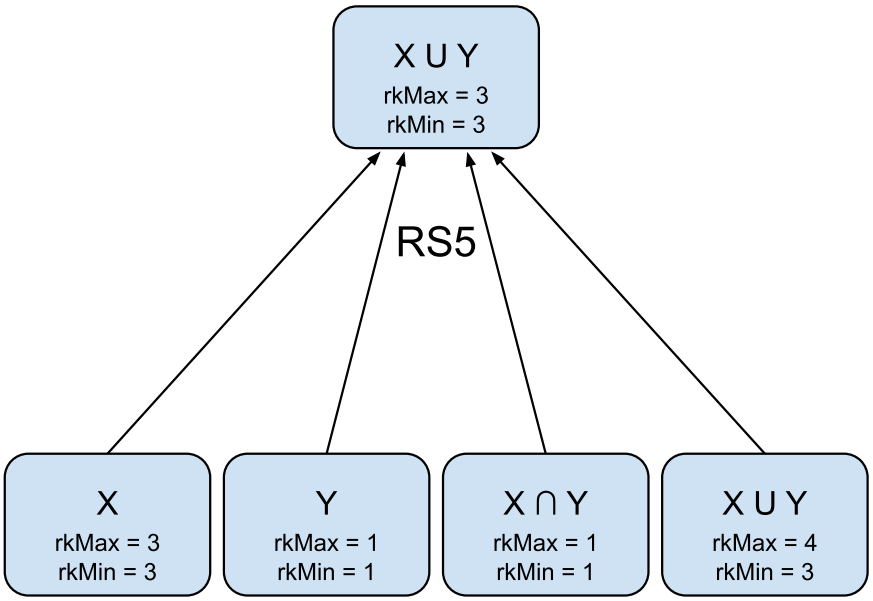


FIGURE III.1.3 – Représentation d'une partie du GD avec l'application de la règle RS5.

Afin d'illustrer le fonctionnement complet de la construction du graphe de déductions, nous présentons dans la Table III.1.3 et la Figure III.1.4 un exemple d'énoncé géométrique simplifié que l'on souhaite saturer. Considérons un plan ABD où l'on construit sur la droite AD un point distinct C. Dans ce cas, il est envisageable de déduire que l'espace engendré par les points ABC est un plan.

```
(* Exemple de lemme à 4 points que l'on souhaite saturer *)
Lemma example : forall A B C D : Point,
rk(A, B, D) = 3 ->
rk(A, C, D) = 2 ->
rk(A, C) = 2 ->
rk(C, D) = 2 ->
rk(A, B, C) = 3.
```

TABLE III.1.3 – Exemple d'énoncé géométrique à saturer.

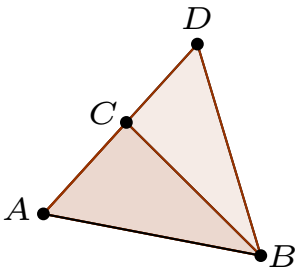


FIGURE III.1.4 – Illustration de la configuration géométrique de la Table III.1.3.

Avant de saturer cet énoncé géométrique, nous commençons par créer le graphe de déductions représentant la configuration géométrique initiale (Figure III.1.5). Étant donné que l'énoncé possède 4 points, la couche initiale du graphe de déductions comporte $2^4 - 1$ noeuds où chaque noeud représente une des parties avec son rang maximum à gauche et minimum à droite.

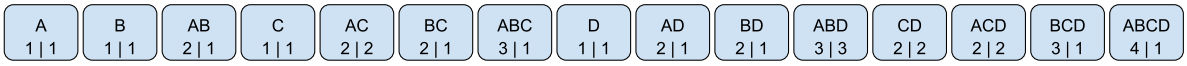


FIGURE III.1.5 – GD initialisé associé à la configuration géométrique de la Table III.1.3.

Puis, nous réalisons une première étape de saturation sur l'ensemble des parties en appliquant chacune des règles dans l'ordre fixe suivant : RS1 RS3 RS2 RS4 RS5 RS7 RS6 RS8. Nous discutons l'intêret de choisir un ordre précis pour les règles dans la partie sur l'optimisation de l'algorithme. Nous obtenons un graphe de déductions partiel (Figure III.1.6) représentant l'ensemble des modifications effectuées à la fin du premier passage illustré par les noeuds en orange. La numérotation en rouge juxtaposée indique l'ordre dans lequel les déductions ont été trouvées.

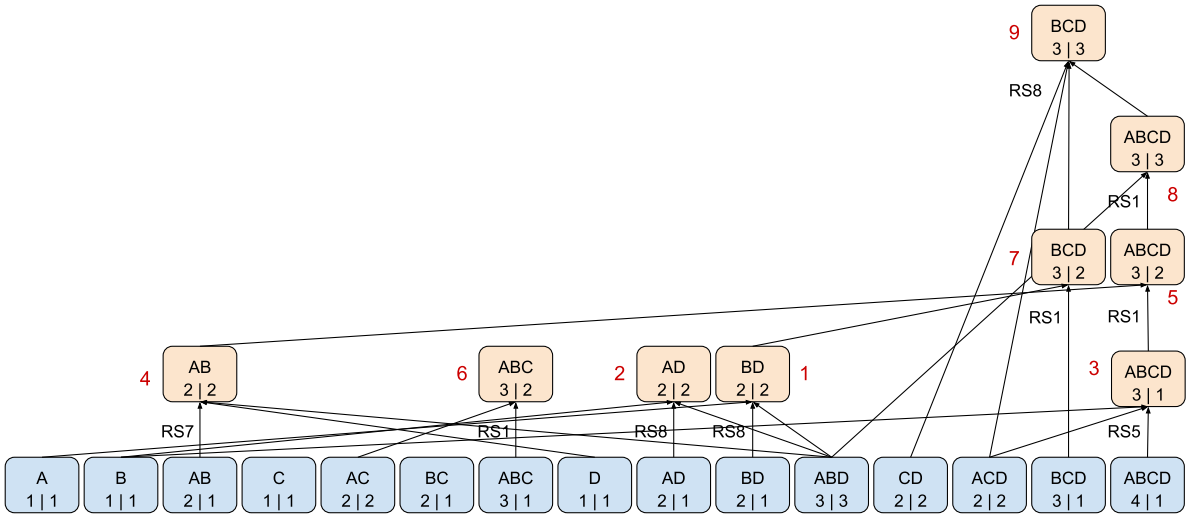


FIGURE III.1.6 – GD partiellement saturé associé à la configuration géométrique de la Table III.1.3.

Finalement, l'algorithme relance un deuxième parcours pour compléter la saturation. Le graphe de déductions (GD) résultant est complété par les noeuds de coloration rouge dans la Figure III.1.7 et permet d'obtenir la conclusion recherchée à savoir que la partie ABC représente strictement un plan. Nous laissons ici volontairement l'algorithme terminer la saturation en trouvant une déduction supplémentaire résumée dans le noeud 11 sachant que le résultat a déjà été trouvé précédemment. Dans cet exemple, le rang minimum et maximum de toutes les parties après saturation sont égaux. Ce n'est pas toujours le cas, cela dépend fortement des contraintes initiales de notre configuration géométrique. En effet, déterminer le rang exact d'une partie, dépend de manière générale de la constriction du problème géométrique. Si le problème est sous-contraint, le rang exact de chaque partie n'est pas déductible. Cependant si le problème est bien-contraint ou sur-contraint, cela ne garantit pas que chaque rang exact peut être évalué. Une partie peut ainsi prendre plusieurs rangs tout en satisfaisant l'ensemble des hypothèses de l'énoncé géométrique, cette variation dans le rang d'un ensemble de point permet de considérer tous les cas dégénérés.

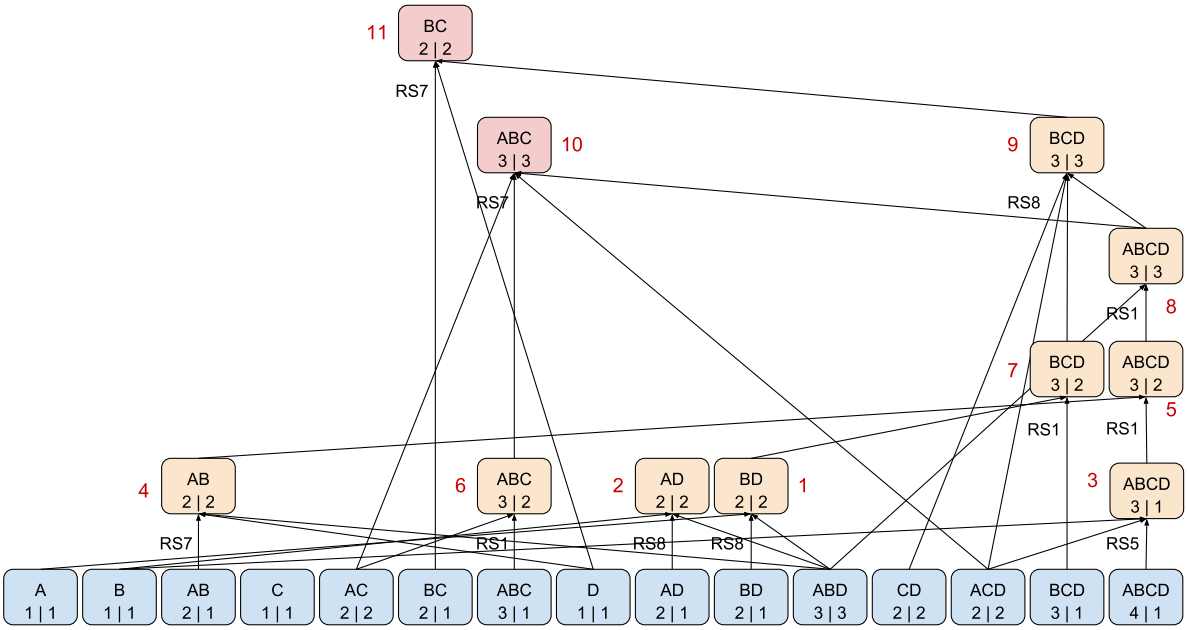


FIGURE III.1.7 – GD complètement saturé associé à la configuration géométrique de la Table III.1.3.

Nous modifions l'Algorithme III.1.4 de saturation en conséquence pour intégrer la construction en parallèle du graphe de déductions (GD).

Algorithme III.1.4 : Étape de saturation et construction du GD.

Entrée(s) : GD initialisé
Sortie(s) : GD mis à jour

```
1  tant que modification au dernier passage faire
2      pour chaque partie X de E faire
3          pour chaque partie Y de E tel que X ≠ Y faire
4              pour chaque règle de réécriture faire
5                  si la règle est activable faire
6                      Mise à jour du rang minimum ou maximum de la partie concernée
7                      en construisant un noeud dans le GD mémorisant l'application de
8                      la règle et lien avec des pointeurs vers les parties parentes
9                  fin si
10             fin pour chaque
11         fin pour chaque
12     fin pour chaque
13 fin tant que
```

2.4 Fenêtre des derniers noeuds calculés

Pour maintenir à jour le graphe ou rajouter un noeud lors de l'étape de saturation, le temps d'accès aux différentes parties est minimisé. En effet, étant donné que le graphe peut contenir jusqu'à $2^n \times 4$ noeuds où n est le nombre de points, nous voulons éviter tous les parcours inutiles de ce dernier. Pour cela, nous construisons une table séquentielle contenant un pointeur vers la dernière modification de chacune des parties. Dans la Figure III.1.8, la fenêtre est représentée

par un tableau de pointeurs vers chaque noeud en violet. Lorsqu'on applique une règle, si l'ensemble modifié représente l'union ou l'intersection ensembliste, il suffit de faire l'opération bits à bits correspondante pour obtenir la partie résultante. Ensuite, l'algorithme extrait la valeur numérique représentée par la chaîne de bits de la partie calculée. Cette valeur numérique permet d'accéder en temps constant dans la table séquentielle au pointeur sur le noeud contenant la dernière modification de la partie concernée. Ce tableau séquentiel de pointeurs représente une fenêtre d'accès aux dernières modifications de toutes les parties de notre configuration géométrique[3]. Pour mieux comprendre cette structure, nous illustrons dans la Figure III.1.9 le chemin d'accès à une partie. À chaque création de noeud, le pointeur correspondant dans la table est modifié par un pointeur sur ce nouveau noeud. L'application de toutes les règles suivantes prendra uniquement en compte les derniers noeuds qui ont été modifiés. Cet accès direct donne la possibilité de récupérer les rangs de chacune des parties efficacement lors de d'application des règles pendant la phase de saturation. De plus, cette structure permet d'établir aisément la relation père-fils lors de la création d'un nouveau noeud. Précisons que cette fenêtre est initialisée par un pointeur sur chacune des parties du graphe de déductions dans leur état initial (configuration géométrique initial). L'Algorithme de saturation III.1.5 est modifié pour intégrer la mise à jour de cette fenêtre des derniers noeuds calculés.

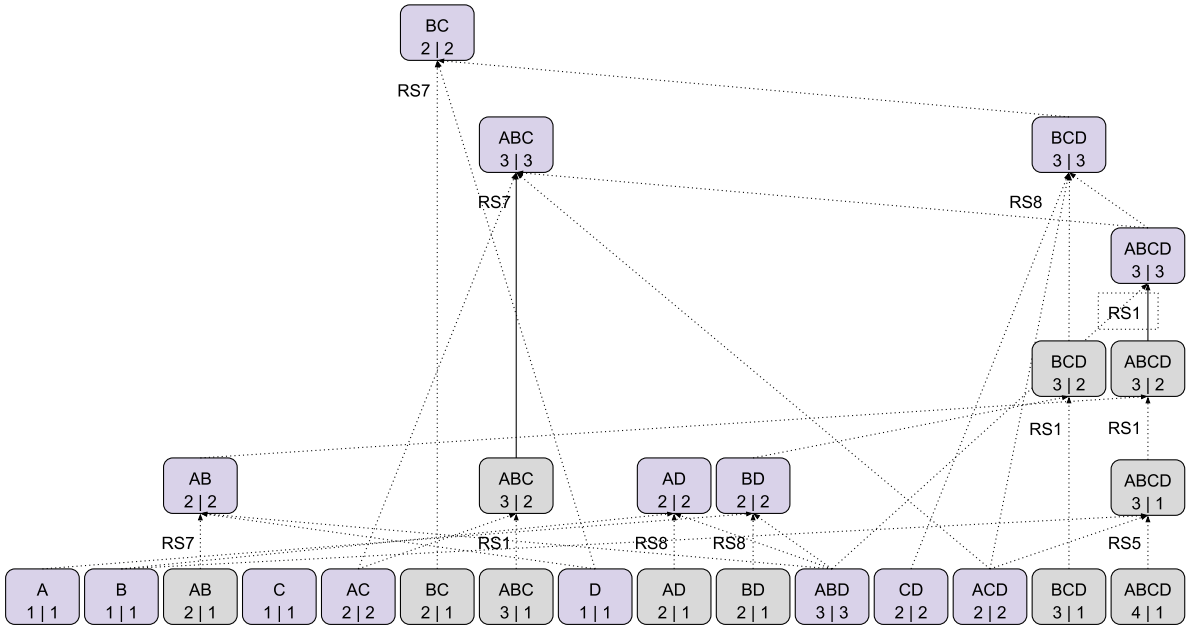


FIGURE III.1.8 – Fenêtre des derniers noeuds calculés associée à la configuration géométrique de la Table III.1.3.

3. Cette table représente l'état courant de la fonction rang sur l'ensemble des parties et peut être vue comme une table des piles

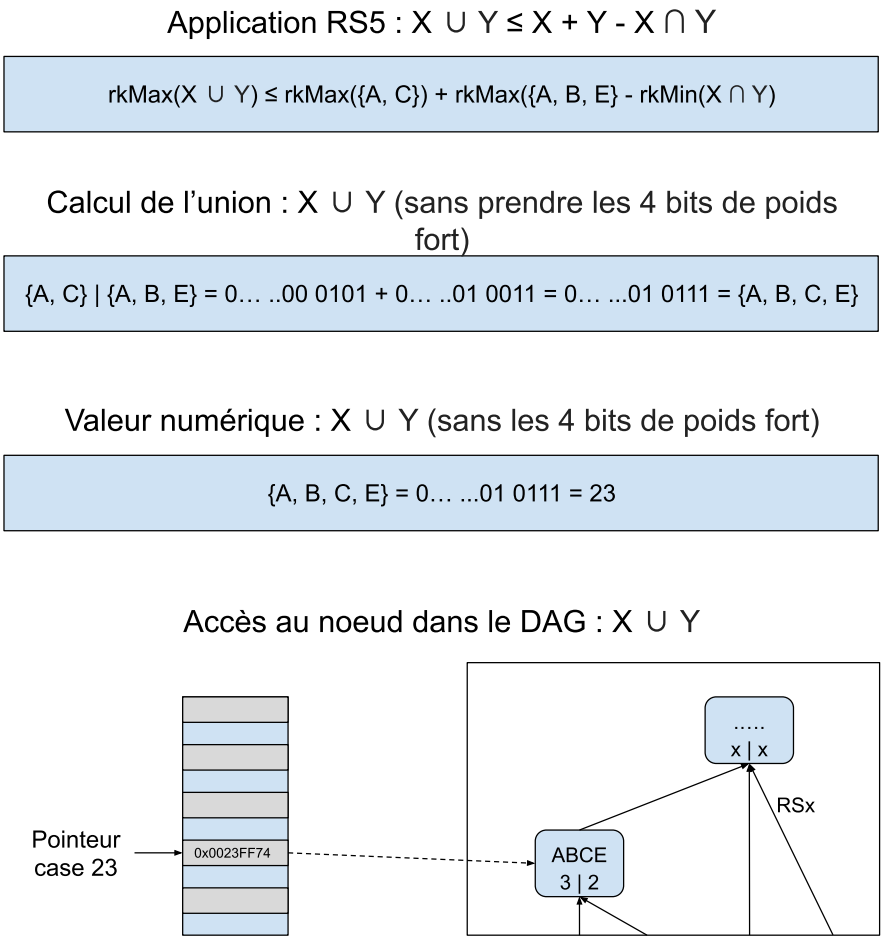


FIGURE III.1.9 – Cheminement pour accéder aux dernières informations de la partie $ABCE$.

Algorithme III.1.5 : Étape de saturation, construction du GD et mise à jour de la fenêtre

Entrée(s) : GD initialisé, fenêtre des derniers noeuds calculés
Sortie(s) : GD mis à jour, fenêtre mise à jour

```
1  tant que modification au dernier passage faire
2      pour chaque partie X de E faire
3          pour chaque partie Y de E tel que X ≠ Y faire
4              pour chaque règle de réécriture faire
5                  si la règle est activable faire
6                      Mise à jour du rang minimum ou maximum de la partie concernée
7                      en construisant un noeud dans le GD mémorisant l'application de
8                      la règle et lien avec des pointeurs vers les parties parentes
9                      Mise à jour du pointeur associé à ce noeud dans la fenêtre
10                 fin si
11             fin pour chaque
12         fin pour chaque
13     fin pour chaque
14 fin tant que
```

2.5 Reconstruction de la preuve et procédé de marquage

La troisième étape consiste à reconstruire la preuve du/des résultat(s) recherché(s) grâce au graphe de déductions construit lors de l'étape précédente. Cette reconstruction est réalisée grâce à un parcours récursif postfixe en partant du noeud dont on souhaite prouver le rang, ou le noeud prouvant la contradiction de la figure initiale. Il est en effet nécessaire pour tout noeud du graphe orienté acyclique de construire au préalable la preuve de chacun de ses fils avant de finalement pouvoir générer le morceau de preuve qui correspond à l'application de la règle aboutissant à ce noeud. Rappelons que ce graphe ne possède par définition aucun cycle, le parcours postfixe est donc comparable à celui d'un arbre en considérant le graphe orienté inverse ou transposé. En reprenant l'exemple de graphe de déductions précédent, nous illustrons dans la Figure III.1.10 le parcours postfixe de la reconstruction de la preuve que la partie ABC représente strictement un plan. Les noeuds de coloration verte retrace le parcours postfixe et la numérotation bleue identifie l'ordre dans lequel les preuves de noeuds sont construites afin d'être réutilisées lors des déductions suivantes.

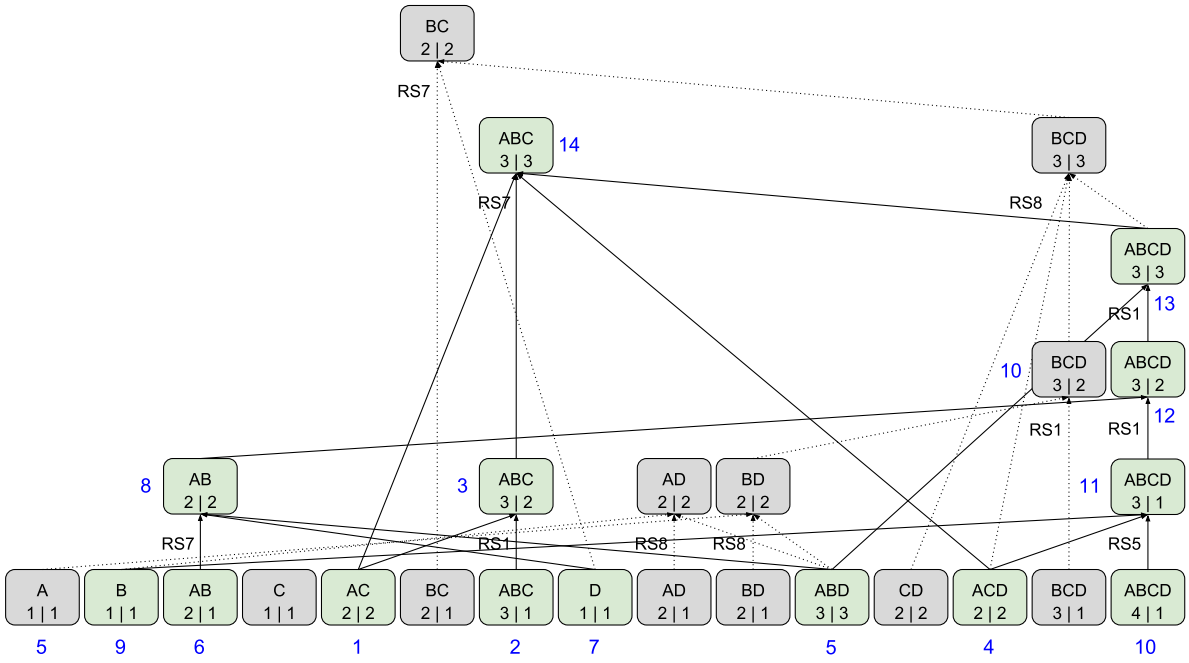


FIGURE III.1.10 – Cheminement de la Reconstruction de la preuve dans le GD associé à la configuration géométrique de la Table III.1.3.

On peut observer que tous les noeuds du graphe de déductions ne sont pas nécessaires à la reconstruction, seuls ceux nécessaires à la preuve sont parcourus. En analysant plus précisément le parcours, on découvre que certains noeuds sont utilisés dans plusieurs preuves et ne doivent pas être parcouru plusieurs fois. À cette fin, nous rajoutons un système de marquage permettant de connaître le statut courant d'un noeud. Ce marquage peut prendre les cinq valeurs suivantes :

- 0 : le noeud n'appartient pas à la reconstruction de la preuve.
- 1 : le noeud fait partie de la reconstruction mais n'a pas encore été parcouru.
- 2 : le noeud a déjà été parcouru mais reste en attente des preuves apportées par chacun de ses fils.

- 3 : la preuve de ce noeud a déjà été reconstruite dans la preuve courante.
- 4 : la preuve de ce noeud a déjà été reconstruite dans un autre lemme.

Le marqueur 0 élimine directement ce noeud de la phase de reconstruction pour ce lemme. Le marqueur 1 est utilisé lors d'une phase de prémarquage pour parcourir l'intégralité des noeuds qui sont nécessaires à la construction de ce lemme. L'ajout d'une phase de prémarquage est obligatoire lorsqu'on souhaite reconstruire la preuve d'un théorème complexe en effectuant un découpage en lemmes intermédiaires. Le graphe de déductions (GD) est alors découpé en plusieurs couches qui sont prouvées successivement. Nous examinerons et détaillerons cette reconstruction par couche dans la section 4. Le marqueur 2 indique que la preuve de ce noeud sera bientôt reconstruite. Le marqueur 3 indique que le noeud a déjà été parcouru et reconstruit dans la démonstration qui est en cours de reconstruction, le résultat de ce noeud est donc déjà connu dans le contexte. Sa descendance reste néanmoins utile puisqu'elle peut éventuellement intervenir dans d'autres reconstructions. Finalement le marqueur 4 intervient lui aussi dans la reconstruction par couche et permet de distinguer les noeuds qui ont déjà été reconstruits précédemment dans d'autres démonstrations. Ces démonstrations forment maintenant des lemmes intermédiaires indépendants que l'on peut réemployer directement dans d'autres reconstructions.

Ensuite, pour effectuer la reconstruction de la preuve, l'algorithme traduit chaque règle appliquée en un code Coq générique. Ce code est englobé dans un bloc regroupant l'intégralité des pas de preuves nécessaires pour établir le rang minimum et maximum du noeud reconstruit. Le bloc vérifie que la preuve des hypothèses dont il a besoin a déjà été effectuée ou que les hypothèses existent dans le contexte avant de prouver le résultat. De manière simplifiée, ce code prend la forme décrite dans la Table III.1.4.

```
(* Bloc précédent *)
...

(* Bloc du résultat à établir *)
assert(Hx : rk(P1 :: P2 :: P3 :: nil) >= 3).
{
        (* Vérification des hypothèses *)
        ...
        (* Préparation de la simplification de l'intersection et de l'union *)
        ...
        (* Application de la règle concernée *)
        assert(HT := rule_2 ...);apply HT. (* application de RS7 *)
}
(* Elimination d'hypothèses *)
try clear Hxxx.

(* Bloc suivant *)
...
```

TABLE III.1.4 – Illustration d'un bloc correspondant à l'application d'une règle.

L'Algorithme récursif III.1.6 résume la phase de reconstruction pour un noeud. Le prémarquage à 1 ou le marquage à 4 lors d'une reconstruction par couche sont effectués au préalable dans la fonction principale.

Algorithme III.1.6 : Étape de reconstruction.

Entrée(s) : Noeud à reconstruire
Sortie(s) : Fichier texte contenant la preuve du noeud

```
si le noeud possède une descendance faire
    pour chaque fils faire
        si le marquage du fils est égal à 1 faire
            Marquage du fils à 2
            Algorithme III.1.6 sur ce fils
        fin si
    fin pour chaque
fin si
Marquage du noeud à 3
Reconstruction du code Coq associée à l'application de la règle stockée dans ce noeud
```

2.6 Validation par l'assistant de preuve Coq

La preuve par bloc générée à partir du graphe de déductions saturé est ensuite importée dans l'assistant de preuve Coq pour être vérifiée et validée par ce dernier. L'algorithme de reconstruction exporte la preuve sous forme de code *Gallina* générique contenant : la génération de l'énoncé, l'introduction des hypothèses, la preuve en elle-même et la conclusion. La reconstruction de la preuve implique le respect de certaines conventions facilitant la génération du certificat : instauration d'une nomenclature sur les hypothèses locales et globales, gestion des hypothèses, gestion des lemmes intermédiaires et contrôle sur les tactiques manipulées. Certaines de ces conventions vont aussi permettre d'optimiser le temps d'exécution pour vérifier la preuve et limiter l'occupation mémoire.

L'utilisation des blocs en Coq permet de gérer les hypothèses de la même façon que les portées dans bon nombre de langage de programmation. Toutes les hypothèses locales introduites dans le bloc sont perdues dès que le résultat recherché dans ce bloc est établi. Le système conserve uniquement l'hypothèse globale déclarée avant l'ouverture du bloc. Pour les hypothèses locales dans les blocs, il est alors possible d'utiliser la même nomenclature à chaque fois sans problèmes d'interférences. Pour les hypothèses globales, nous nous servons d'une nomenclature globale incluant les points mis en jeu, le rang minimum et/ou maximum concerné. Nous obtenons ainsi des exemples d'hypothèses tels que `HP3P6P9m2` ou `HP4P8P9M3` qui décrivent respectivement le contenu suivant : $rk(P3, P6, P9) \geq 2$ et $rk(P4, P8, P9) \leq 3$. Les symboles `m` et `M` signifient dans ce contexte respectivement minimum et maximum.

Cette nomenclature va permettre de gérer les hypothèses en évitant que le contexte soit surchargé inutilement. Dès qu'une hypothèse initiale ou calculée n'est plus utilisée, elle est éliminée de la preuve. À tout instant, si une hypothèse est encore présente dans le contexte, alors elle doit encore servir à établir un résultat futur. Nous détaillons ce mécanisme de gestion des hypothèses dans la partie Optimisation 4. Cette gestion permet à Coq de grandement améliorer ses performances lorsqu'il doit rechercher une hypothèse dans le contexte.

Pour optimiser encore les performances, il est souhaitable de découper les preuves en lemmes intermédiaires qui doivent être référencés et inclus en tant qu'hypothèse globale dès que le système en a besoin. Ces lemmes sont automatiquement gérés par la phase de reconstruction en utilisant le marquage à 4. Ce découpage permet de diminuer la taille de la preuve, de diminuer

le nombre d'hypothèses traitées, et surtout la taille du terme de preuve. Ces lemmes une fois prouvés servent de résultats intermédiaires pouvant être réutilisés dans d'autres preuves.

Finalement, la validation des blocs par le système Coq doit être la plus efficace possible. Nous réutilisons pour cela les résultats majeurs établis et observés dans le Chapitre II.1. L'idée générale consiste à manipuler les tactiques les plus adaptées et les plus simples pour résoudre une tâche spécifique. Nous diminuons au maximum la charge de travail du système Coq en fournissant un maximum d'informations aux tactiques pour éviter le plus possible la routine d'unification effectuée par le système Coq. Nous déportons ainsi au mieux la difficulté et le temps de calcul dans le certificat produit en C. Nos expérimentations, détaillées dans la suite, nous montrent que les performances mémoires du système Coq sont comme en géométrie finie le principal goulot d'étranglements. Toutes ces optimisations permettent de vérifier des preuves pouvant aller jusqu'à plusieurs dizaines de milliers de lignes et comportant des centaines d'hypothèses.

3 Mesure de performances

Nous abordons maintenant les aspects liés à la complexité pour découvrir les goulots d'étranglements dans la chaîne de traitements. Cette étude permet ensuite de mieux envisager les optimisations à mettre en place pour résoudre des configurations géométriques complexes.

3.1 Complexité en temps du prouveur

Rappelons que le prouveur fonctionne à partir d'un ensemble fini de règles s'appliquant sur les paires de l'ensemble des parties de la configuration géométrique. Le temps d'exécution d'un tel algorithme dépend principalement du nombre de points de la configuration géométrique. En effet, le nombre de parties à étudier dans le problème vaut 2^n où n désigne le nombre de points. En étudiant toutes les paires différentes de l'ensemble des parties, nous devons analyser $\frac{2^n \times (2^n - 1)}{2}$ combinaisons. Cette analyse doit être répétée m fois jusqu'à l'obtention du résultat recherché ou la saturation complète du problème. En résumé, la complexité en temps de l'étape de saturation du prouveur s'exprime de la façon suivante : $m \times (2^{2n-1} - 2^{n-1})$.

En comparaison, la complexité en temps de l'étape d'initialisation (allocation et initialisation des structures) et de l'étape de reconstruction (parcours de graphe et écriture dans un fichier) sont négligeables. Finalement, la complexité en temps de la construction d'une preuve est : $m \times p^2$ où m est le nombre d'étapes jusqu'à la saturation et p le nombre de parties réellement considérées.

À titre d'exemple, cet algorithme exponentiel permet de saturer des configurations géométrique allant jusqu'à 20 points (1 048 576 de parties) en quelques heures sur la machine standard [4]. En cas de succès, nous obtenons à la fin un graphe saturé où il est possible d'extraire la preuve des rangs de chacune des parties.

3.2 Complexité en mémoire du prouveur

Pour fonctionner le prouveur s'appuie sur deux structures : le graphe des parties et la fenêtre des derniers noeuds calculés. Nous stockons dans chaque noeud du graphe un mot binaire représentant la partie et ses rangs, deux entiers pour le marquage et la règle appliquée, ainsi que deux listes de pointeurs pour les noeuds précédents et suivants. Un noeud représente en mémoire 22-x octets : 2 octets pour le mot binaire, 8 octets pour les deux entiers, 12-16 octets pour les 2-4 noeuds fils, et 0-x octets pour les noeuds parents. Le nombre de noeuds parents varie : soit le

4. Spécificités de cette machine : Intel(R) Core(TM) i5-4460 CPU @3.20GHz avec 16Go de mémoire

noeud n'a servi dans aucune règle, soit le noeud a servi dans une seule règle (donnant sa propre évolution par exemple), soit le noeud est utilisé dans plusieurs voire tous les calculs de parties du graphe. Dans ce dernier cas, le noeud peut avoir au maximum $3 \times p - 2$ noeuds parents où p est le nombre de parties. Sachant que le nombre de fils par noeud est limité, on peut établir que chaque noeud possède 4 parents en moyenne, c'est à dire 16 octets. Si un noeud possède plus de parents que cette moyenne, cela impliquera que d'autres noeuds ont nécessairement moins de parents. On estime de cette manière qu'un noeud en mémoire occupe en moyenne 42 octets.

Le graphe possède au minimum un noeud pour chacune des parties et dans le pire des cas 4 noeuds pour chaque évolution de la même partie jusqu'à saturation. Le graphe représente ainsi une taille maximale en mémoire de $42 \times p \times 4$ octets avec p le nombre de parties.

Pour la fenêtre, nous stockons dans un tableau de taille p un pointeur vers chacune des parties. La fenêtre représente en mémoire $p \times 4$ octets avec p le nombre de parties.

Sur un énoncé géométrique à 20 points, il faudrait au maximum 176 Mo pour stocker l'intégralité du graphe (172Mo pour le graphe et 4Mo pour la fenêtre). La consommation en mémoire du prouveur n'est donc pas une limite pour notre algorithme. Il est cependant possible d'optimiser la structure de données pour minimiser l'occupation mémoire.

3.3 Complexité en temps de la vérification du certificat

La complexité en temps de la vérification du certificat par l'assistant de preuve Coq dépend de trois facteurs : la taille de la preuve, le nombre d'hypothèses et la complexité des tactiques qui sont employées. Sachant qu'il est difficile de calculer rigoureusement la complexité en temps et en mémoire de la vérification d'une preuve de la même manière qu'un algorithme, nous donnons une intuition de ces complexités avec des ordres de grandeur.

En considérant qu'un maximum d'information est fourni au système et que les tactiques utilisés sont efficaces, nous estimons le temps d'exécution d'un bloc en Coq correspondant à l'application d'une règle entre 0.05 sec à 0.2 sec. Le nombre de bloc par preuve varie en fonction de la difficulté de la preuve et du nombre de points impliqués. En utilisant nos exemples, la taille d'une preuve à s points peut varier entre $s \times 10$ à $s \times 10000$ lignes. Le nombre de bloc obtenu dans une preuve à x lignes est $\frac{x}{10}$ blocs. Et finalement, le temps d'exécution d'une preuve à s points dans le pire des cas est de $\frac{s \times 10\ 000}{10} \times 0.2$ sec.

En étudiant un énoncé géométrique à 20 points, nous obtenons dans le pire des cas une preuve de 200 000 lignes contenant 20 000 blocs où le temps de vérification est de 4 000 sec. Dans tous les cas, ce temps d'exécution reste négligeable par rapport au temps de calcul de la saturation.

3.4 Complexité en mémoire de la vérification du certificat

La complexité mémoire d'une preuve Coq dépend fortement du nombre d'hypothèses, du nombre de buts courants et surtout de la taille du terme de preuve. Plus la preuve est longue, plus le système doit mémoriser le cheminement nécessaire pour arriver jusqu'à cette étape de la démonstration. À notre connaissance, il n'existe pas de travaux qui traitent de l'occupation mémoire dans les preuves. Pour contourner ce problème de performance en mémoire, la meilleure solution consiste à segmenter les preuves en utilisant le génie logiciel orienté Coq. Dans notre cas, la difficulté ne réside pas réellement dans l'application des règles dérivées des axiomes mais dans la gestion à grande échelle de tous ces blocs. Pour quantifier cette occupation, nous pouvons uniquement nous baser sur nos expérimentations.

Par exemple, un énoncé géométrique à 15 points possédant 15 000 lignes peut occuper plus de 6Go de mémoire si on ne supprime pas les hypothèses devenues inutiles. Une démonstration de 15 000 lignes correspond à 1500 blocs introduisant 1 500 nouvelles hypothèses dans le contexte. Si les hypothèses sont supprimées dès que possible, le même énoncé se limite à l'utilisation de 2Go de mémoire. Pour minimiser encore plus l'usage de la mémoire, il est nécessaire d'effectuer un découpage de la démonstrastion en lemmes afin de restreindre la taille du terme de preuve.

3.5 Conclusion sur les complexités

Toutes ces complexités n'ont pas le même degré d'importance. Le facteur le plus limitant à considérer actuellement est le temps d'exécution du prouveur afin de produire une solution. Le cheminement de la preuve n'étant pas connu à l'avance, nous ne pouvons pas prédire l'ordre de parcours idéal des parties et des règles. Une recherche force brute est donc effectuée sur l'intégralité des parties et des règles jusqu'à l'obtention du résultat recherché ou jusqu'à ce que la saturation soit complète. Quelques améliorations et heuristiques sont néanmoins présentées et testées dans la section suivante pour en évaluer les performances sur le temps d'exécution et la taille du certificat.

Le deuxième aspect à considérer est l'occupation mémoire lors de vérification du certificat. Une bonne gestion des hypothèses permet de considérer la vérification de preuves bien plus conséquentes sans que l'on se soucie de la mémoire. Cependant, lorsque les preuves dépassent 10 000 lignes, elles ne peuvent plus être traitées en une seule fois et nécessitent un découpage en lemmes intermédiaires. Ce découpage permet de réduire la taille du terme de preuve et d'augmenter la modularité dans les grandes preuves. Cette gestion de la mémoire est complètement automatique et est détaillée dans la section suivante.

4 Optimisations

Afin d'optimiser le temps d'exécution tout en contrôlant la taille de la preuve produite, nous détaillons plusieurs améliorations sur le parcours des règles et des parties. Toutes ces optimisations ne sont pas concluantes mais permettent de mieux cerner le fonctionnement interne du processus de saturation d'un énoncé géométrique. Une fois l'énoncé saturé, nous décrivons quelques simplifications qui sont effectuées pour aider le plus possible l'assistant de preuve dans sa validation de la preuve.

4.1 Parcours linéaire

La première amélioration que nous analysons s'intéresse au parcours de l'ensemble des parties. Avec un algorithme force brute où il est nécessaire d'analyser toutes les paires de parties, la meilleure méthode reste un parcours linéaire comme nous le rappelons dans l'Algorithme III.1.7.

Algorithme III.1.7 : Parcours de l'ensemble des paires de l'ensemble des parties.

```
1 pour chaque partie X de E entre 1 et 2^n faire
2     pour chaque partie Y de E entre X et 2^n faire
3         ...
4     fin pour chaque
5 fin pour chaque
```

Nous garantissons ainsi qu'aucune déduction n'a été oubliée. De manière naturelle, nous considérons l'heuristique suivante : le parcours croissant des parties en commençant avec les

premiers points de la construction donne un meilleur résultat. Les derniers points qui ont été introduits possèdent en général très peu d'informations et ne sont pas les points les plus propices à la propagation de l'information lors de la saturation. En parcourant de manière croissante (respectivement décroissante) l'ensemble des parties, nous pouvons simplifier l'ensemble des règles en éliminant les règles RS2 et RS4 (respectivement RS1 et RS3) de la Table III.1.2. Étant donné que les parties sont triées dans un ordre croissant de 1 à 2^n selon leur mot binaire, la propriété suivante est toujours vérifiée en appliquant l'algorithme III.1.7 sur les parties X et Y :

Proprieté III.1.1. $\forall\ X\ Y,\ motbinaire(X) < motbinaire(Y) \Rightarrow Y \not\subset X$

Cette propriété permet de simplifier l'ordonnancement des règles en ne considérant plus que l'application de 6 règles dans le cas d'un parcours linéaire. Ce parcours peut être soit croissant, soit décroissant.

4.2 Ordre des règles

L'optimisation suivante provient de l'examen de l'ordonnancement lors de l'application des règles de réécriture. Le résultat obtenu établit que cet ordre a un impact sur le cheminement de la preuve mais n'influe aucunement sur la complexité en temps de l'algorithme. Bien souvent plusieurs démonstrations distinctes peuvent être produites pour résoudre un même problème géométrique, cependant l'idée principale de ces preuves n'est que très rarement modifiée (à condition que cette preuve ne soit pas trop simple). Cette idée se retrouve lorsque nous effectuons la correspondance entre la démonstration manuscrite et la preuve engendrée automatiquement. Dans les deux cas, la déduction du rang de quelques ensembles de points clés permet de mener à son terme la preuve. Les modifications provoquées lorsqu'on change l'ordre des règles interviennent dans le calcul de certains résultats intermédiaires qui sont déduits de plusieurs manières.

Pour illustrer cette situation, considérons deux ensembles $X = \{1, 3\}$ et $Y = \{1, 2, 3\}$. Le rang de l'ensemble X évolue en appliquant quatre règles : soit la règle de sous-modularité modifiant directement l'ensemble X (RS7), soit la règle de sous-modularité sur l'intersection entre X et Y qui n'est autre que l'ensemble X (RS6), soit la règle d'inclusion de X dans Y (RS1 ou RS3). En fonction de la preuve à résoudre, nous pouvons gagner ou perdre un facteur de 1% à 5% sur le nombre de lignes. Il n'est pas possible de prédire à l'avance quel ordre donnera le meilleur résultat en termes de nombre de lignes. La Table III.1.5 donne un aperçu de la variation de la taille de la preuve en fonction de l'ordonnancement. On peut constater que l'écart en nombre de lignes entre deux ordonnancements est très anecdotique et que chaque exemple a un ordre optimal différent. Nous justifions peu après dans la sous-section 4.1 la disparition des règles RS3 et RS4.

	Exemple 1 (10 pts)	Exemple 2 (14 pts)	Exemple 3 (15 pts)	Exemple 4 (19 pts)
RS1|RS2|RS5|RS6|RS7|RS8	651	6271	7 495	78116
RS6|RS2|RS8|RS1|RS5|RS7	679	6172	7 316	79422
RS7|RS8|RS1|RS2|RS5|RS6	667	6193	7 522	78113
RS5|RS6|RS7|RS8|RS1|RS2	679	6146	7 437	78318

TABLE III.1.5 – Taille de la preuve en nombre de lignes en fonction de l'ordre des règles.

En observant plus précisément l'utilisation des règles lors de la saturation sur les exemples précédents de la Table III.1.5, on constate que quelque soit l'ordonnancement le nombre de règles de non-décroissance appliquées sur tout l'énoncé géométrique est nettement supérieur au nombre

de règles de sous-modularité (voir Table III.1.6).

	Exemple 1 (10 pts)	Exemple 2 (14 pts)	Exemple 3 (15 pts)	Exemple 4 (19 pts)
Règles de non décroissance	1 413	35 949	62 443	945 180
Règles de sous-modularité	322	12 226	31 195	625 623
Total des règles appliquées	1 735	48 175	93 638	1 570 803

TABLE III.1.6 – Comparaison du nombre de règles appliquées globalement entre les deux principales propriétés matroïdales.

Nous considérons donc l'heuristique suivante où il est plus probable que les règles RS1 à RS4 modifient le rang de l'ensemble X ou Y que celles issues de la sous-modularité. De cette manière, l'application des règles RS5 à RS8 profite déjà des modifications. C'est pourquoi, nous posons dans la suite l'ordre fixe suivant : RS1 (RS3) RS2 (RS4) RS5 RS7 RS6 RS8 appliquant ainsi les règles de non-décroissance avant les règles de sous-modularité.

4.3 Règle de Pappus

L'introduction de l'axiome de Pappus parmi les règles à vérifier permet d'élargir le nombre de théorèmes d'incidence que l'on peut prouver. Ce théorème sert principalement à débloquer des configurations géométriques où les règles sur les matroïdes ne sont plus suffisantes. Néanmoins, cette règle bien plus complexe ne doit pas être traitée de la même manière que les règles matroïdales. En effet, ce théorème nécessite de contrôler 8 alignements parmi 9 points modulo 6 permutations possibles afin de valider la bonne application du théorème. Une telle vérification est bien trop coûteuse et elle échoue dans la majorité des cas. Dans les rares situations où la règle s'applique, la conclusion est souvent déjà connue. Dans cette optique, nous séparons la règle de Pappus pour uniquement l'appliquer sur un problème où la saturation par les règles matroïdales a échoué. Si une ou plusieurs configurations de Pappus permettent de découvrir des résultats non référencés dans le graphe, nous ajoutons ces informations au graphe avant de relancer une étape de saturation. Ce processus est répété jusqu'à l'obtention du résultat ou que la règle de Pappus ne soit plus applicable.

La Table III.1.7 compare les temps d'exécution en fonction du traitement de la règle de Pappus sur différentes configurations géométriques. Notons que l'exemple 4 a obligatoirement besoin de la propriété de Pappus pour être complété. L'exécution de cet exemple sans Pappus est donc non pertinente.

	Exemple 1 (10 pts)	Exemple 2 (14 pts)	Exemple 3 (15 pts)	Exemple 4 (19 pts)
Sans Pappus	0.113s	43s	2m33s	X
Avec Pappus pendant chaque saturation	0.151s	1m01s	3m40s	43h27m11s
Avec Pappus uniquement à la fin de chaque saturation	0.120s	47s	2m42s	36h34m48s
Gain en %	-21%	-23%	-26%	-22%

TABLE III.1.7 – Temps d'exécution moyen en fonction de l'intégration de la règle de Pappus.

Nous observons que le traitement de la propriété de Pappus uniquement en fin de saturation apporte un gain non négligeable de 25% sur le temps d'exécution en comparaison avec une vérification systématique à chaque étape. La différence en temps d'exécution entre la saturation « Sans Pappus » et la saturation « Avec Pappus uniquement à la fin » sur les trois premiers exemples apparaît lorsque l'algorithme n'est pas arrêté après l'obtention du résultat recherché. L'algorithme cherche la présence éventuelle d'une configuration de Pappus dans l'énoncé géométrique bien que cette propriété ne soit pas utile pour conclure la démonstration courante. Nous estimons ainsi le temps que l'algorithme met pour vérifier cette propriété et nous assurons que la saturation soit complète pour reconstruire plus rapidement d'autres résultats dans ce contexte. Nous modifions en conséquence l'Algorithme III.1.5 de saturation par l'Algorithme III.1.8 pour inclure cette optimisation.

Algorithme III.1.8 : Étape de saturation.

```
1  tant que modification en appliquant la règle de Pappus faire
2      tant que modification au dernier passage faire
3          pour chaque partie X de E faire
4              pour chaque partie Y de E tel que X ≠ Y faire
5                  pour chaque règle de réécriture faire
6                      ...
7                  fin pour chaque
8              fin pour chaque
9          fin pour chaque
10     fin tant que
11     si une configuration de Pappus est identifiée faire
12         Mise à jour du rang minimum ou maximum de la partie concernée
13     fin si
14 fin tant que
```

4.4 Heuristique de coloration

Il est possible d'optimiser ce parcours linéaire en considérant une heuristique utilisant uniquement les parties dont le rang a été modifié au dernier passage dans la boucle de saturation. L'idée consiste à marquer les parties dont le rang est modifié au passage n par un entier de valeur n indiquant la dernière fois que la partie a évolué. Cet entier agit comme un marqueur de coloration. À l'étape $n+1$, si l'entier est égal ou supérieur à n, le noeud est noir et doit être utilisé dans la saturation. Au contraire, si le marqueur est strictement inférieur à n, le noeud conserve sa couleur blanche et n'est pas nécessairement manipulé. L'algorithme estime que pour faire avancer la saturation, au moins une des deux parties de la paire sélectionnée X et Y, leur union $X \cup Y$ ou leur intersection $X \cap Y$ doit avoir un entier égal à n. Si aucune de ces 4 parties n'a été modifiée récemment, il est impossible de déduire de nouveaux résultats dans ce contexte. Toutes les déductions ont déjà été réalisées précédemment sur ces noeuds et l'algorithme peut tout de suite passer à la paire de parties suivante. À ce fonctionnement, nous rajoutons le traitement particulier des hypothèses initiales qui sont toujours considérées comme des noeuds noirs permanents. Sachant qu'elles peuvent intervenir dans toutes les étapes de saturation, elles ne doivent jamais changer de couleur.

La Figure III.1.11 résume quelques configurations que l'heuristique de coloration peut rencontrer. Le cas n°1 ne doit pas être traité puisqu'aucune des 4 parties ne possède un marqueur de coloration noire. Le cas n°2 représente une configuration où toutes les parties ont évolué depuis l'étape précédente. L'heuristique n'a pas besoin d'analyser le marqueur de coloration de

chacune des parties, elle se contente d'une évaluation paresseuse : si une des parties possède un marqueur de coloration noire, plus besoin d'observer la couleur des autres parties. Cette évaluation paresseuse est à nouveau utilisée dans le cas n ° 3 où seule la partie X est noire. Le dernier cas doit être étudié en entier jusqu'à l'analyse du marqueur de la partie $X \cap Y$.

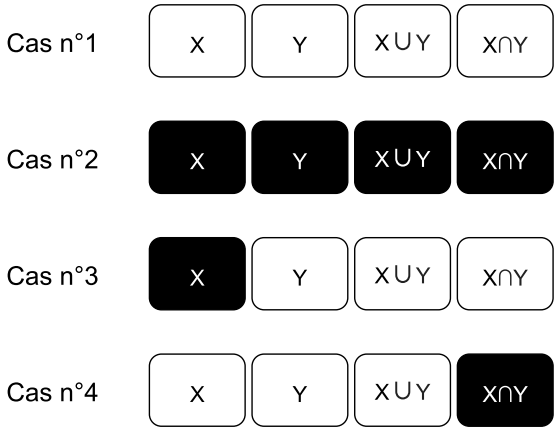


FIGURE III.1.11 – Différentes configurations possibles pour l'heuristique de coloration.

Dans les premières étapes de saturation, cette coloration des parties n'est pas très avantageuse étant donné que presque l'intégralité des parties est juste initialisée. En effet, presque toutes les parties voient leur rang évoluer rapidement. Puis au fur et à mesure, le nombre de parties modifiées se restreint, accélérant ainsi le parcours de toutes les parties. Cette approche permet alors de considérer uniquement des noeuds susceptibles d'apporter de l'information pour terminer la résolution du problème. À la fin de la saturation, lorsque l'algorithme n'a plus de déductions à effectuer, autrement dit qu'aucune modification n'a eu lieu pendant une étape de saturation tous les noeuds sont devenus blancs excepté les hypothèses. La propriété de Pappus peut être vérifiée avant une éventuelle nouvelle étape de saturation. Cette heuristique apporte un gain de 25% à 50% sur le temps d'exécution en comparaison avec un parcours linéaire classique comme illustré dans la Table III.1.8.

	Exemple 1 (10 pts)	Exemple 2 (14 pts)	Exemple 3 (15 pts)	Exemple 4 (19 pts)
Parcours linéaire classique	0.120s	47s	2m42s	36h34m48s
Heuristique de coloration	0.092s	21s	2m1s	15h43m53s
Gain en %	-23%	-55%	-25%	-57%

TABLE III.1.8 – Temps d'exécution moyen en fonction de l'heuristique de parcours.

Finalement nous modifions l'Algorithme III.1.7 pour intégrer l'heuristique de coloration dans l'Algorithme III.1.9.

Algorithme III.1.9 : Parcours de l'ensemble des paires de l'ensemble des parties.

```
1 pour chaque partie X de E entre 1 et 2^n faire
2     pour chaque partie Y de E entre X et 2^n faire
3         si X est noire || Y est noire || X ∩ Y est noire || X ∪ Y est noire
4             ...
5         fin si
6     fin pour chaque
7 fin pour chaque
```

4.5 Saturation par strate

La saturation par strate ou par couche consiste à saturer un énoncé géométrique en plusieurs étapes en introduisant au fur et à mesure les différents points de la construction géométrique. L'idée consiste à découper l'énoncé en plusieurs paquets de points de plus en plus gros incluant à chaque fois le paquet de points précédent comme illustré dans la Figure III.1.12. Dans cette figure, nous incluons les informations de la saturation des 9 points du paquet rose dans la saturation du paquet rouge contenant 16 points avant de finalement saturer l'intégralité du problème à 19 points. De cette manière, le système peut utiliser l'information qui a été calculée à l'étape précédente sur un paquet de points restreint en tant que résultat intermédiaire afin d'assurer une saturation plus efficace et plus rapide de l'étape courante. Cette méthode de saturation permet de propager plus efficacement l'information sur les nouvelles parties qui sont introduites et élimine des étapes de preuves qui deviennent inutiles. Considérons l'exemple suivant où une partie X est complètement saturée avec un rang égal à 3 à l'étape N, la partie Y telle que $X \subset Y$ à l'étape *N+1* peut commencer directement avec un rang supérieur ou égal à 3 et ignorer par exemple l'étape de transition où le rang est supérieur ou égal à 2.

Bien que cette technique minimise la taille de la preuve en utilisant au maximum les informations de la strate précédente, le temps d'exécution pour chaque strate est cumulé. En découpant un énoncé géométrique à P points en plusieurs couches, le temps d'exécution de l'algorithme par couche se rapproche du temps d'exécution cumulé de la saturation de toutes les couches prises séparément. En effet, le système doit à chaque couche reparcourir l'intégralité des parties afin de compléter les informations qui peuvent être manquantes.

Dans l'exemple III.1.12, le temps d'exécution global de l'algorithme par couche correspond à l'addition du temps d'exécution de chacune des strates : rose, rouge et jaune. Cependant, il est important de noter que le temps d'exécution des strates précédentes est toujours négligeable en comparaison du calcul de la strate courante. Nous minimisons ainsi la taille de la preuve en augmentant légèrement le temps d'exécution de l'algorithme en cumulant celui des différentes couches.

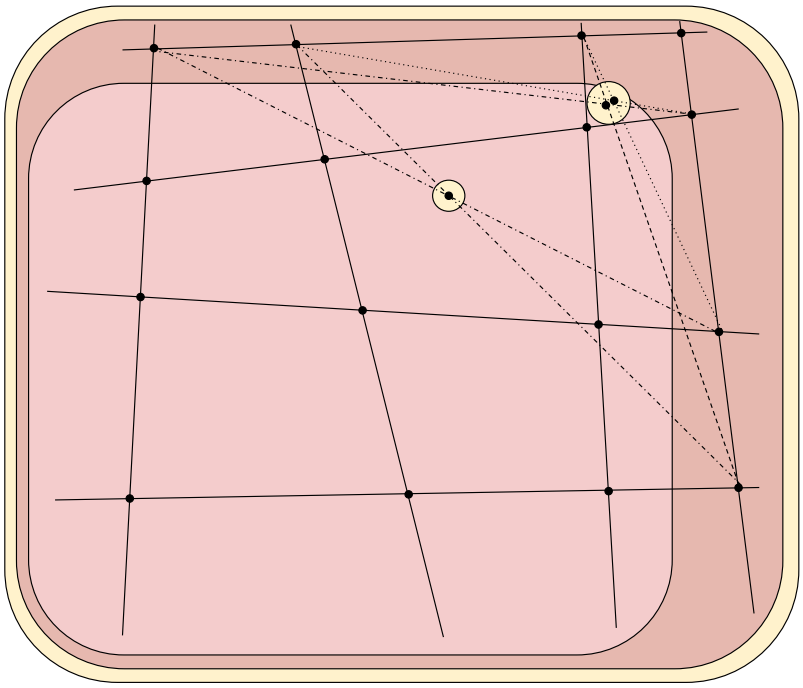


FIGURE III.1.12 – Stratification d'un énoncé géométrique contenant une application de Pappus.

Le principal intérêt de cette méthode réside dans le futur découpage de la preuve en lemmes intermédiaires. Lorsque la preuve devient trop longue, il est absolument nécessaire de la diviser pour qu'elle puisse être vérifiée. L'obtention d'un résultat valide est prioritaire sur l'augmentation du temps d'exécution. Un découpage manuel d'une preuve de plusieurs dizaines de milliers de lignes devenant hors de propos, il est nécessaire de mettre en place un mécanisme de scission automatique qui dans la mesure du possible préserve l'intelligibilité de la preuve.

En observant le graphe dans son intégralité, il est impossible d'indiquer les noeuds qui occupent un rôle central dans la preuve méritant la création d'un lemme intermédiaire. Pour nous aider, les deux indicateurs les plus significatifs sont la hauteur dans le graphe ainsi que le nombre de parents. Si un noeud divise assez la preuve en ayant une hauteur suffisante et qu'il peut être réutilisé dans le calcul de plusieurs parents, il est souhaitable d'en faire un lemme séparé. La difficulté réside dans le choix du critère : s'il est trop souple, le nombre de lemmes intermédiaires est trop élevé, la preuve devient inutilement plus longue. Au contraire, s'il est trop rigide, la preuve n'est pas assez découpée et la vérification échoue. De plus, ce critère évolue en fonction de la preuve effectuée, la topologie du graphe associé étant à chaque fois différente. Au final, le graphe doit être parcouru dans sa globalité pour élaborer un critère pertinent. L'utilisateur n'a dans ce cas aucun impact sur la sémantique du découpage, celui-ci est complètement automatisé.

4.6 Notre solution

La solution que nous privilégions est d'utiliser la saturation par strate introduite dans la section précédente en y associant une sémantique géométrique. L'utilisateur contrôle le découpage en lemmes intermédiaires en définissant les différentes strates géométriques à saturer dès l'énonciation du problème. Ce choix des strates se fait intuitivement lors de la construction de la configuration géométrique. L'utilisateur définit selon l'ordre de construction les différentes strates au sein du problème. Le nombre de strates et l'écart entre ces dernières définissent naturellement le critère de subdivision de la preuve.

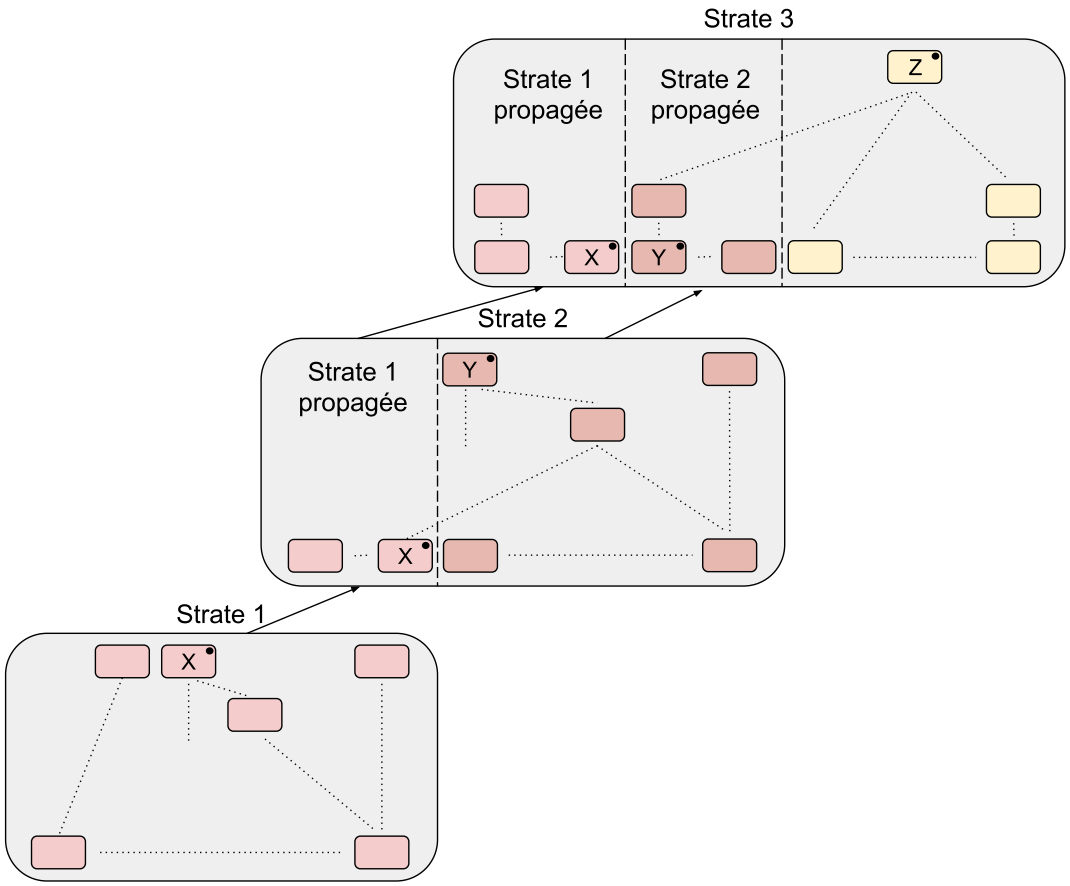


FIGURE III.1.13 – Preuves par couche d'un énoncé géométrique à 3 strates.

En reprenant la stratification en 3 couches d'un énoncé géométrique présenté dans la Figure III.1.12, nous illustrons le mécanisme de création de lemmes intermédiaires dans le schéma III.1.13. La saturation de l'énoncé géométrique complet se déroule en 3 étapes correspondant aux 3 strates de l'énoncé géométrique.

À chaque fois qu'une strate a été saturée, l'algorithme reporte les informations qui ont été calculées dans la strate courante (inférieure) vers la strate suivante (supérieure) : c'est la propagation de l'information. Dans l'exemple III.1.13, nous transférons l'information du noeud X de la strate 1 vers la strate 2. Dans un second temps, nous transférons le noeud X et Y de la strate 2 vers la strate 3. Ces informations sont ensuite utilisées pour effectuer la saturation de la nouvelle strate plus efficacement. Tous les noeuds finaux locaux d'une strate, qui sont réutilisés pour propager l'information dans la strate suivante et qui appartiennent à la preuve, définissent des lemmes intermédiaires. Ces lemmes sont des résultats intermédiaires servant de base à la saturation suivante. Ces noeuds ne peuvent plus évoluer sans l'éventuelle introduction d'une strate supplémentaire. Cette méthode a l'avantage de définir des lemmes intermédiaires pertinents qui sont des résultats finaux locaux pouvant être réemployés dans n'importe quelle preuve d'un résultat d'une des strates suivantes. Dans la Figure III.1.13, nous souhaitons reconstruire la preuve

du noeud Z. Le parcours de l'arborescence de ce noeud nécessite le calcul du noeud Y de la strate 2 ainsi que le noeud X de la strate 1. La preuve finale comporte au moins 3 lemmes dont deux lemmes intermédiaires pour X et Y.

Cette subdivision en couche peut être aisément généralisée à n strates où n est le nombre de points. Chaque strate correspondant à la saturation de l'énoncé géométrique en rajoutant un point supplémentaire à la construction. La manipulation d'un découpage aussi fin de l'énoncé géométrique n'est en pratique pas requis surtout si l'on ne souhaite pas multiplier le nombre de lemmes intermédiaires à prouver. Le but premier est d'obtenir une preuve suffisamment segmentée dont la vérification n'occupe pas toute la mémoire.

Les optimisations, présentées dans cette section 4, améliorent les performances des différentes parties du pipeline de notre prouveur. Elles permettent aussi de considérer et résoudre des preuves qui deviennent de plus en plus compliquées et conséquentes. Nous illustrons et analysons l'application de ces dernières sur des théorèmes variés qui ont une difficulté croissante dans le Chapitre suivant III.2.

CHAPITRE III.2

Un catalogue d'exemples

"Geometry is the art of correct reasoning from incorrectly drawn figures"

Henri Poincare (1854–1912)

Résumé

Maintenant que le pipeline du prouveur généralisé a été examiné, nous illustrons les résultats obtenus à travers un catalogue d'exemples incluant des théorèmes fondamentaux de la géométrie d'incidence projective. Pour cela , nous étudions des configurations géométriques qui sont de plus en plus compliquées à résoudre. Nous montrons sur les derniers exemples l'apport des optimisations telles que l'heuristique de coloration et le mécanisme de scission des preuves de grande taille.

Nous commençons ce catalogue par des exemples triviaux comportant uniquement quelques points. De cette manière, nous illustrons simplement le fonctionnement des différentes étapes du pipeline (section 1). Dans la suite, nous analysons la résolution de quelques lemmes intermédiaires de Desargues qui sont présentés dans [MNS09,MNS12] (section 2). Nous examinons naturellement la résolution automatique du théorème de Desargues comportant 15 points en dimension 3 en considérant le cas dégénéré où la figure est aplatie (section 3). Nous traitons ensuite l'unicité de la construction du point appelé conjugué harmonique par rapport à un triplet de point définissant la droite projective (section 4). Finalement, nous explorons les limites de notre prouveur en considérant la propriété de Dandelin-Gallucci comportant 19 points (section 5).

Contenu

Dans ce chapitre, nous illustrons le fonctionnement du prouveur en détaillant la résolution automatique d'exemples ou de théorèmes classiques de la géométrie d'incidence projective de plus en plus complexes. Ces exemples permettent de mettre en évidence les limites intrinsèques de la génération automatique des preuves et de mieux comprendre l'introduction des différentes optimisations présentées dans la section 4. Pour chaque exemple, nous donnons un énoncé mathématique, une figure associée, une intuition de la preuve ou la démonstration complète, la traduction de l'énoncé mathématique avec l'approche matroïdale sur les rangs, éventuellement le graphe de déductions associé (GD) ainsi que les résultats obtenus.

1 Lemmes triviaux

Les premiers exemples traités permettent de vérifier le bon fonctionnement du prouveur sur des énoncés simples contenant un nombre limité de point.

1.1 Restriction à une droite

Cet exemple illustre la capacité du prouveur à considérer des hypothèses avec des rangs qui ne sont pas forcément exacts ($\leq$ et $\geq$).

Lemme III.2.1 (Line unification)**.** *Si trois points A, B, C deux à deux distincts ne définissent pas un plan, alors ces trois points sont alignés.*

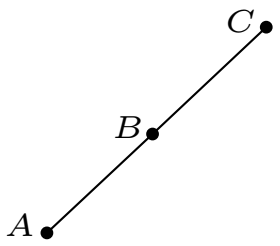


FIGURE III.2.1 – Illustration du lemme III.2.1.

Démonstration. Étant donné que les points sont deux à deux distincts, cela signifie que chaque paire de points engendre une droite. En rajoutant la contrainte que les trois points ne définissent pas un plan, nous déduisons que les trois droites sont confondues et que les points sont colinéaires. □

```
Lemma line_unification : forall A B C : Point,
rk(A, B) = 2 -> rk(A, C) = 2 ->
rk(B, C) = 2 -> rk(A, B, C) <= 2 ->
rk(A, B, C) = 2.
```

TABLE III.2.1 – Énoncé mathématique associé au lemme III.2.1.

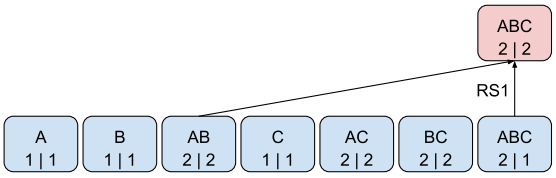


FIGURE III.2.2 – GD complètement saturé associé à la configuration géométrique du lemme III.2.1.

Une seule règle d'inclusion entre les parties AB et ABC s'applique pendant la saturation de cette configuration géométrique. Le rang maximum de la partie ABC est initialisé à 2 grâce aux hypothèses. Il reste à déduire que le rang minimum est 2. Nous donnons à titre d'exemple la preuve Coq générée à partir de la saturation dans la Table III.2.2.

```
(* \'{E}noncé du lemme *)
Lemma LP1P2P3 : forall P1 P2 P3 ,
rk(P1 :: P2 :: nil) = 2 -> rk(P1 :: P3 :: nil) = 2 ->
rk(P2 :: P3 :: nil) = 2 -> rk(P1 :: P2 :: P3 :: nil) <= 2 ->
rk(P1 :: P2 :: P3 :: nil) = 2.

(* Introduction des hypothèses *)
intros P1 P2 P3 HP1P2eq HP1P3eq HP2P3M .

(* Application de la règle d'inclusion entre AB et ABC *)
assert(HP1P2P3m2 : rk(P1 :: P2 :: P3 :: nil) >= 2).
{
 try assert(HP1P2eq : rk(P1 :: P2 :: nil) = 2) by (...).
 assert(HP1P2mtmp : rk(P1 :: P2 :: nil) >= 2) by (...).
 assert(Hcomp : 2 <= 2) by (...).
 assert(Hincl : incl (P1 :: P2 :: nil) (P1 :: P2 :: P3 :: nil)) by (...).
 assert(HT := rule_5 (P1 :: P2 :: nil) (P1 :: P2 :: P3 :: nil) 2 2 _ _ _);apply HT.
}
(* Nettoyage des hypothèses inutiles *)
try clear HP1P2M2. try clear HP1P2m2. try clear HP1P2P3m1.

(* Conclusion et validation de la preuve *)
assert(HP1P2P3M : rk(P1 :: P2 :: P3 ::  nil) <= 3) by (...).
assert(HP1P2P3m : rk(P1 :: P2 :: P3 ::  nil) >= 1) by (...).
intuition.
Qed.
```

TABLE III.2.2 – Preuve Coq associée au lemme III.2.1.

1.2 Colinéarité

Cet exemple illustre la capacité du prouveur à propager la colinéarité dans une configuration géométrique à partir des hypothèses.

Lemme III.2.2 (Collinearity). *Soient quatre points A, B, C, D avec B et C distincts. Si ces points définissent deux droites ABC et BCD alors les quatre points sont alignés.*

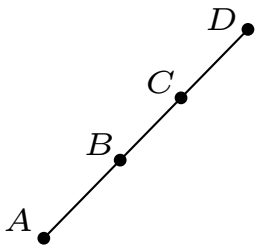


FIGURE III.2.3 – Illustration du lemme III.2.2.

Démonstration. Les droites ABC et BCD possèdent deux points distincts B et C en commun. Par pseudo-transitivité de la colinéarité, on déduit que les quatre points sont alignés et définissent une seule et même droite. □

```
Lemma collinearity : forall A B C D : Point,
rk(B, C) = 2 -> rk(A, B, C) = 2 -> rk(B, C, D) = 2 ->
rk(A, B, C, D) = 2.
```

TABLE III.2.3 – Énoncé mathématique associé au lemme III.2.2.

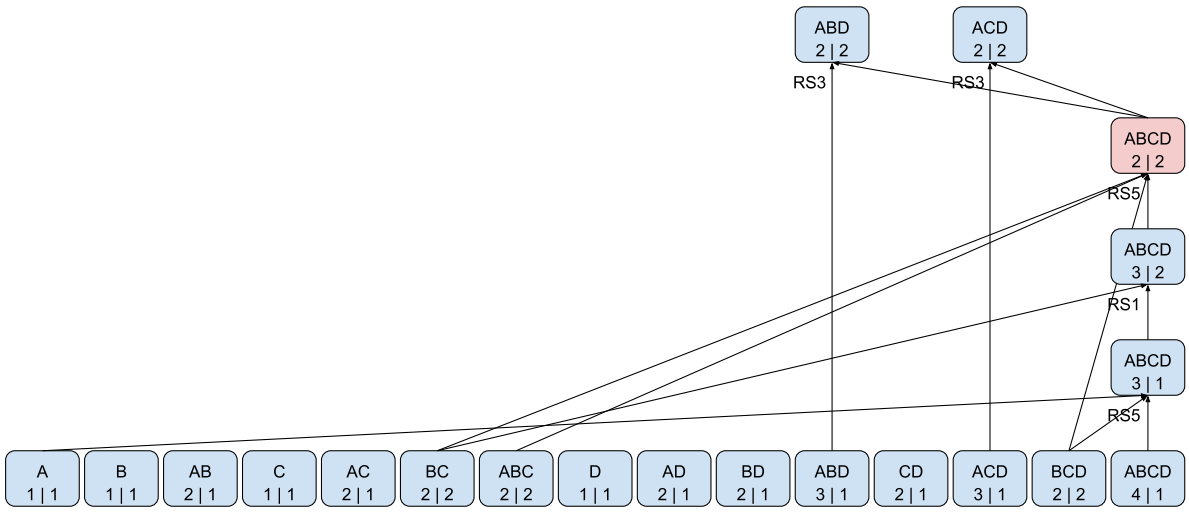


FIGURE III.2.4 – GD complètement saturé associé à la configuration géométrique du lemme III.2.2.

En 3 règles, le système déduit que la partie $ABCD$ définit une droite. À partir de ce résultat, le prouveur déduit la transitivé de la colinéarité sur les parties ABD et ACD. Notons que le rang des parties AB, AD, CD et BD n'est pas exact à la fin de la saturation. Ce sont les configurations dégénérées du lemme qui reste toujours vrai, que ces points soient égaux ou non.

1.3 Sur-contraint

Cet exemple illustre la capacité du prouveur à traiter des énoncés contenant plus d'informations que nécessaires.

Lemme III.2.3 (Overconstrained)**.** *Soient deux plans ABC et BCD non coplanaires, on peut montrer que les points B et C appartenant aux deux plans sont distincts.*

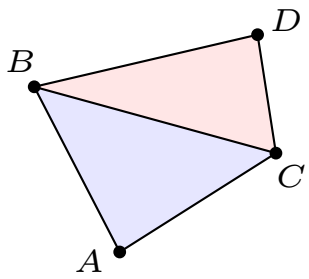


FIGURE III.2.5 – Illustration du lemme III.2.3.

Démonstration. Cet énoncé contient deux plans ABC et BCD non coplanaires avec deux points B et C en commun. Si trois points permettent de définir un plan, alors tout couple de points est distinct et représente une droite. B et C sont donc distincts et définissent une droite. □

```
Lemma over_constrained : forall A B C D : Point,
rk(A, B, C) = 3 -> rk(B, C, D) = 3 ->
rk(A, B, C, D) = 4 ->
rk(B, C) = 2.
```

TABLE III.2.4 – Énoncé mathématique associé au lemme III.2.3.

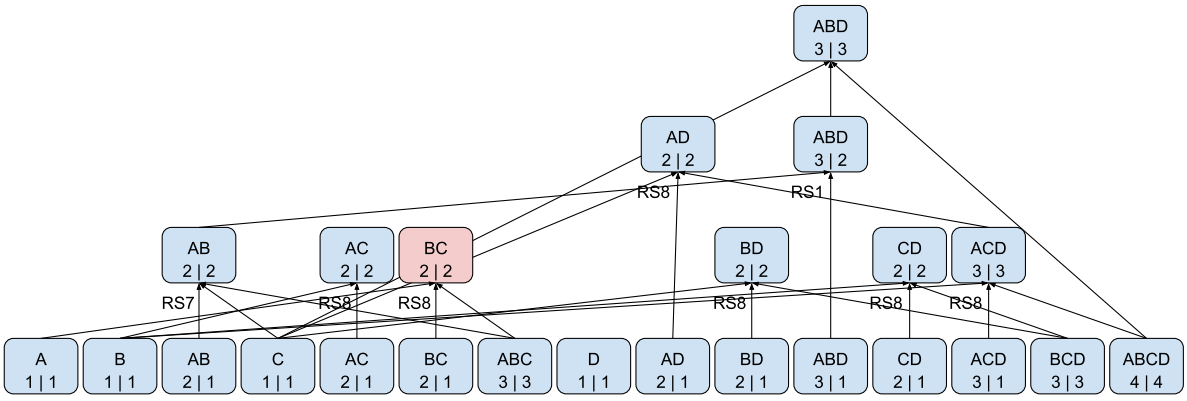


FIGURE III.2.6 – GD complètement saturé associé à la configuration géométrique du lemme III.2.3.

Cet exemple d'intersection entre deux plans permet d'observer qu'il est possible de déduire $rk(B, C) = 2$ de plusieurs manières. Pour obtenir ce résultat, la saturation utilise ici l'hypothèse

$rk(A, B, C) = 3$. Sachant que le rang de trois points est égal à 3, on peut conclure que le rang de deux points parmi trois est nécessairement 2. Il est possible d'effectuer une déduction analogue à partir des hypothèses $rk(B, C, D) = 3$ ou $rk(A, B, C, D) = 4$. L'énoncé géométrique est ici sur-contraint.

1.4 Égalité entre points

Cet exemple illustre la capacité du prouveur à traiter des égalités entre points comme deux points syntaxiquement distincts dont le rang est égal à 1.

Lemme III.2.4 (Points equality)**.** *Si ABC est un plan et si on construit un point D égal à C alors la figure ABD définit un plan.*

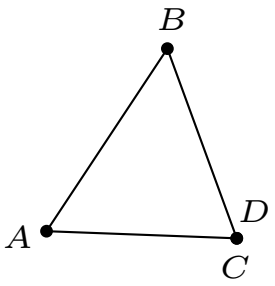


FIGURE III.2.7 – Illustration du lemme III.2.4.

Démonstration. En substituant le point C par le point D qui lui est égal dans le plan ABC, nous obtenons le plan ABD. □

```
Lemma points_equality : forall A B C D : Point,
rk(A, B, C) = 3 -> rk(C, D) = 1 ->
rk(A, B, D) = 3.
```

TABLE III.2.5 – Énoncé mathématique associé au lemme III.2.4.

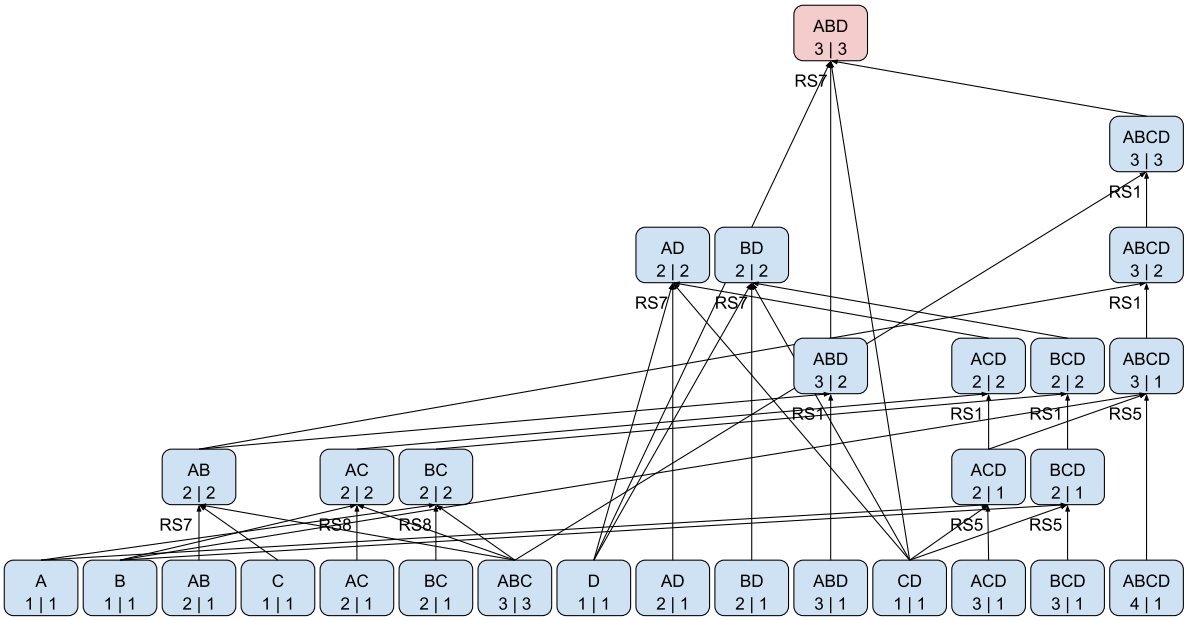


FIGURE III.2.8 – GD complètement saturé associé à la configuration géométrique du lemme III.2.4.

Ce dernier exemple permet d'observer qu'une égalité entre deux points n'interfère pas dans la construction d'une preuve. L'information que les points C et D sont égaux est au fur et à mesure propagée dans toutes les parties concernées avant de finalement conclure que le rang de ABD est égal à 3. Une amélioration notable pour les performances de notre algorithme lors de l'étape de saturation serait d'éliminer un des points lorsqu'une égalité entre points est détectée. La difficulté réside dans l'ajustement du graphe de déduction et dans la reconstruction de la preuve Coq en considérant deux points qui ne sont pas nécessairement égaux dès le départ. Si cette déduction intervient au milieu d'une preuve, toutes les hypothèses incluant l'un des deux points doivent être mises à jour au moment où la déduction est effectuée à la fois dans le GD mais aussi dans la preuve Coq associée. Ce changement fait partie des perspectives d'améliorations de notre prouveur.

Il est intéressant d'observer que l'on peut utiliser directement l'hypothèse $rk(A, B, C) = 3$ dans la partie $ABCD$ en utilisant une règle d'inclusion pour obtenir que $rk(A, B, C, D) \geq 3$ et ainsi simplifier deux étapes du graphe de déductions. Pour cela, il est nécessaire d'employer le parcours avec coloration plutôt qu'un parcours linéaire classique.

1.5 Résultats

Nous ne fournissons aucun résultat pour cette catégorie de lemmes puisque les preuves associées ne sont pas assez significatives pour analyser le comportement du prouveur dans sa globalité : la production de la preuve ainsi que sa vérification est immédiate.

2 Lemmes intermédiaires de Desargues

Ces lemmes proviennent de la démonstration du théorème de Desargues en utilisant uniquement les rangs [MNS09, MNS12]. Nous conservons le nommage associé à ces lemmes. Ce sont des configurations intermédiaires qui ont été identifiées pour faciliter la mécanisation de la preuve. Avant de nous attaquer à la démonstration complète du théorème de Desargues, nous prouvons

automatiquement ces lemmes afin de tester les capacités du prouveur. Les graphes de déductions associés à des configurations géométriques de 5 points ou plus ne sont plus exposés pour des questions de lisibilité.

2.1 Schéma L1

Lemme III.2.5 (L1 Scheme)**.** *Soit un plan ABO. Si on construit un point A' différent de O sur la droitre AO et un point B' différent de O sur la droite $B'O$ alors la figure $A'B'O$ définit un plan.*

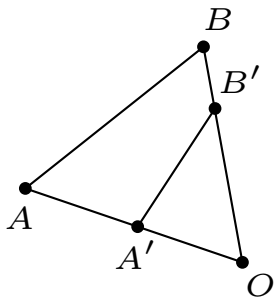


FIGURE III.2.9 – Illustration du lemme III.2.5.

Démonstration. Par construction les points A' et B' sont nécessairement distincts. En effet, ces deux points sont différents de O qui est l'unique point d'intersection entre les droites AO et BO. D'une part, nous avons que les trois points A', B' et O sont deux à deux distincts et non alignés. D'autre part, trois points ne peuvent pas déterminer un objet plus grand qu'un plan. Nous en déduisons que $A'B'O$ détermine exactement un plan. □

```
Lemma l1_scheme : forall A B O A' B' : Point,
rk(A, B, O) = 3 -> rk(A, A', O) = 2 ->
rk(B, B', O) = 2 -> rk(A', O) = 2 -> rk(B', O) = 2 ->
rk(A', B', O) = 3.
```

TABLE III.2.6 – Énoncé mathématique associé au lemme III.2.5.

2.2 Schéma rABOO'

Lemme III.2.6 (rABOO' scheme)**.** *Soit un espace défini par $ABOP$. Si on ajoute un point O' différent de O sur la droite OP alors la figure $ABOO'$ est un espace.*

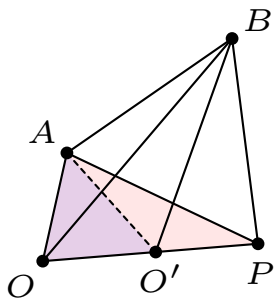


FIGURE III.2.10 – Illustration du lemme III.2.6.

Démonstration. Par construction les points O et O' sont distincts. Le point O' joue un rôle similaire au point P. La droite OO' est confondue avec la droite OP. Le plan AOO' est similaire au plan AOP. Et l'objet $ABOO'$ reste une figure de l'espace. □

```
Lemma rABOO'_scheme : forall A B P O O' : Point,
rk(A, B, P, O) = 4 -> rk(O, O', P) = 2 ->
rk(O, O') = 2 ->
rk(A, B, O, O') = 4.
```

TABLE III.2.7 – Énoncé mathématique associé au lemme III.2.6.

2.3 Schéma subl2rABMP

Lemme III.2.7 (subl2rABMP scheme)**.** *Soit un plan défini par ABC et contenant quatre points quelconques A', B', C' et O. Si on ajoute un point M égal à l'un de ces quatre points A', B', C' et O alors la figure $ABCM$ représente un plan.*

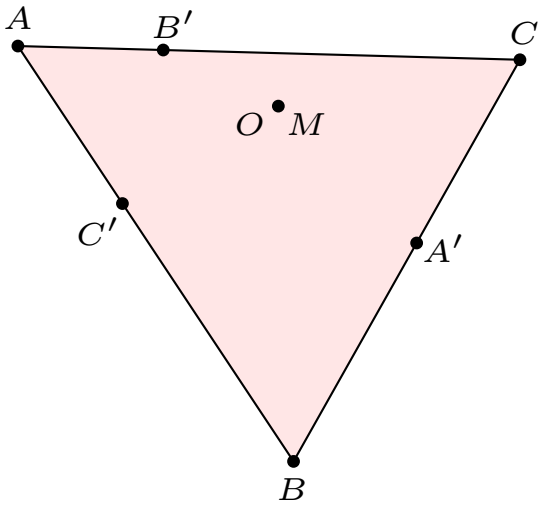


FIGURE III.2.11 – Illustration du lemme III.2.7.

Démonstration. Le plan ABC est étendu en construisant quatre points quelconques supplémentaires A', B', C' et O [1]. Notons ici que les points A', B', C' sont placés respectivement sur les droites BC, AC, AB, la preuve reste inchangée pour un cas plus général. En construisant un point M égal à l'un de ces quatre points, nous ajoutons un point M supplémentaire au plan déjà existant $ABCA'B'C'O$. Les plans représentés par ABC et $ABCA'B'C'O$ sont identiques puisqu'ils sont construits à partir de la même base ABC. Nous en déduisons que l'ajout du point M à ABC définit toujours un plan. □

```
Lemma subl2rABMP_scheme : forall A B C A' B' C' O M : Point,
rk(A, B, C) = 3 -> rk(A, B, C, A', B', C', O) = 3 ->
rk(M, O) = 1 \/ rk(M, A') = 1 \/ rk(M, B') = 1 \/ rk(M, C') = 1 ->
rk(A, B, C, M) = 3.
```

TABLE III.2.8 – Énoncé mathématique associé au lemme III.2.7.

2.4 Schéma rCC'O'PC"

Lemme III.2.8 (rCC'O'PC" scheme)**.** *Soit un plan ABC contenant quatre points quelconques A', B', C' et O avec C' différent de C. Nous construisons une droite $OO'P$ avec P un point en dehors du plan et O' un point sur cette droite différent de O. Si l'intersection C'' existe entre les droites CO' et $C'P$ alors les points de la figure $PO'CC'C''$ sont coplanaires.*

1. La figure associée à cette preuve et les suivantes représentent un cas particulier où les points A', B' et C' appartiennent aux côtés opposés respectifs. Cette configuration particulière n'influe en rien sur les démonstrations présentées.

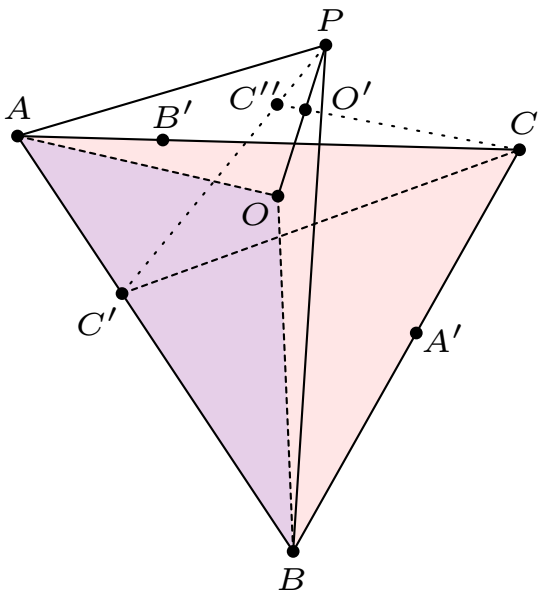


FIGURE III.2.12 – Illustration du lemme III.2.8.

Démonstration. Nous prenons le temps d'analyser la construction de ce lemme 3D qui est un peu compliqué. La définition du lemme III.2.8 donne une impression de liberté [2] sur l'emplacement du point O en précisant que celui-ci doit uniquement appartenir au plan ABC et permettre la construction de la droite OP. Nous obtenons ainsi un exemple de figure III.2.12 qui est dans les faits une configuration impossible.

La création de l'intersection C'' entre les CO' et $C'P$ entraîne une contrainte dissimulée sur le point O. En effet, cette intersection existe si et seulement si le point O est localisé sur la droite CC'. En observant la figure III.2.13, on s'aperçoit que les plans OPC en rouge et OPC' en bleu ne sont pas confondus. La droite OP représente l'intersection entre ces deux plans. Il est alors impossible pour la droite CO' qui coupe une première fois le plan bleu OPC' en O' de couper à nouveau ce plan en C'' excepté si ces plans sont confondus. Pour que les hypothèses sur l'intersection en un point C'' soient vérifiées, le point O est contraint d'appartenir à la droite CC'. Il est alors évident que $CC'C''PO'$ définit un plan. □

Pour aller plus loin, on se demande naturellement pourquoi le point O n'est pas inclus dans ce résultat puisqu'il appartient à la droite CC' qui est elle-même incluse dans le plan $CC'C''PO'$. En observant de plus près l'énoncé géométrique et les détails obtenus par le prouveur, il est possible de générer une unique configuration où le point O a la possibilité d'être à l'extérieur de la droite CC'. Pour cela, il faut considérer un cas particulier où le point O' est égal au point P. Dans ce cas, les droites CO' et $C'P$ se coupent automatiquement en C'' qui est aussi le point P et définissent nécessairement un plan comme illustré dans III.2.14. Le point O n'est plus contraint d'appartenir à la droite CC' et peut se placer n'importe où dans le plan ABC tout en conservant le résultat que $CC'C''PO'$ détermine un plan.

2. À noter que les points A' et B' sont des points complètement libres du plan qui n'interviennent pas dans la démonstration de ce lemme.

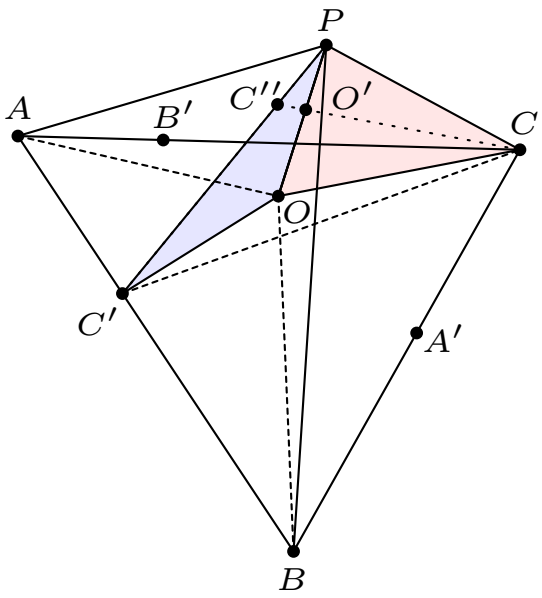


FIGURE III.2.13 – Illustration de la preuve du lemme III.2.8.

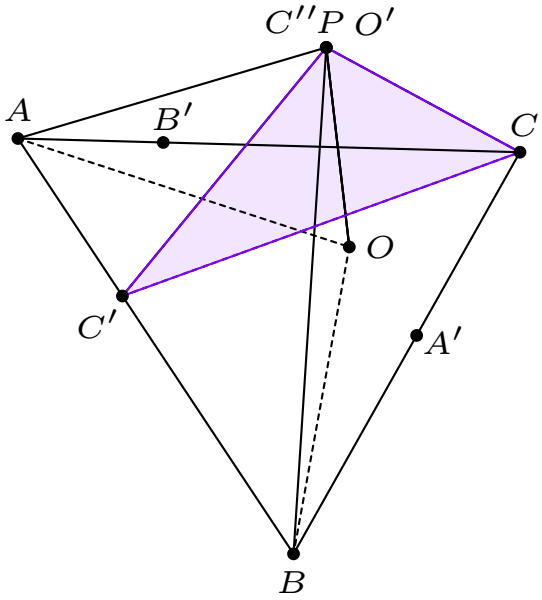


FIGURE III.2.14 – Illustration de la preuve du lemme III.2.8.

```
Lemma rCC'O'PC''_scheme : forall A B C A' B' C' O P O' C'' : Point,
rk(A, B, C, A', B', C', O) = 3 -> rk(C, C') = 2 ->
rk(A, B, O, P) = 4 -> rk(O, O', P) = 2 -> rk(O, O') = 2 ->
rk(C, O', C'') = 2 -> rk(C', P, C'') = 2 ->
rk(C, C', C'', P, O') = 3.
```

TABLE III.2.9 – Énoncé mathématique associé au lemme III.2.7.

2.5 Résultats

Nous détaillons les résultats obtenus pour ces lemmes intermédiaires de Desargues dans la Table III.2.10. Les notations abrégées *T.E*, *M.* et *Nb.* désigne respectivement le temps d'exécution, l'occupation mémoire sur la machine standard [3] et le nombre d'étapes dans la saturation.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
L1 (5 pts)	0.001s	<1%	3	147	1.1s	<1%
rABOO' (5 pts)	0.001s	<1%	2	140	0.9s	<1%
subl2rABMP (8 pts)	0.001s	<1%	3	95	0.9s	<1%
rCC'O'PC" (10 pts)	0.157s	<1%	4	177	1.15s	<1%

TABLE III.2.10 – Test de performance pour les lemmes intermédiaires de Desargues.

Ces tests de performances sont réalisés en supprimant les hypothèses le plus tôt possible, sans heuristique de coloration et sans subdivision. On peut remarquer que ces problèmes intermédiaires ne sont pas encore assez complexes pour atteindre les limites du prouveur. La génération de preuve et sa vérification sont effectuées en une seconde tout au plus avec une occupation mémoire négligeable. Ces preuves ne nécessitent donc pas l'utilisation de la coloration ou de la subdivision. En ce qui concerne la gestion des hypothèses, ces preuves restent suffisamment courtes pour ne pas être affectées par le choix de suppression des hypothèses ou non.

3 Théorème de Desargues

Nous nous intéressons maintenant à la preuve complète du théorème de Desargues III.2.1 dans sa version projective sans utiliser les lemmes intermédiaires démontrés précédemment. Rappelons que la propriété de Desargues est une propriété plane qui peut ne pas être vérifiée par un plan projectif. C'est le cas du plan de Moulton [Mou02] par exemple. Cependant cette propriété devient un théorème dans tout plan plongé dans un espace de dimension ≥ 3. En d'autres termes, cet énoncé a valeur d'axiome en géométrie projective plane alors que dans un espace projectif de dimension 3 ou plus, cette propriété devient un théorème.

3.1 Preuve du théorème de Desargues en 3D

Théorème III.2.1 (Théorème de Desargues)**.** *Soit E un espace projectif et P, Q, R, P', Q', R' des points de cet espace. Soient PQR et P'Q'R' deux triangles non aplatis. Si les droites (PP'), (QQ') et (RR') sont concourantes en un point O alors α, β et γ sont alignés avec $\alpha = (PR) \cap (P'R')$, $\beta = (QR) \cap (Q'R')$ et $\gamma = (PQ) \cap (P'Q')$.*

3. Spécificités de cette machine : Intel(R) Core(TM) i5-4460 CPU @3.20GHz avec 16Go de mémoire

L'idée générale de cette preuve dans l'espace projectif est assez classique [Kod14] : nous prouvons premièrement une version du théorème où les deux triangles ne sont pas coplanaires que nous appelons Desargues 3D illustré par la figure III.2.15. Puis, nous déduisons à partir de cette preuve, la démonstration du cas particulier en 2D où les deux triangles appartiennent au même plan comme dans la configuration III.2.16. Pour passer de la figure 2D à la version 3D de Desargues, nous réalisons une extrusion du triangle $P'Q'R'$ grâce au point S situé à l'extérieur du plan.

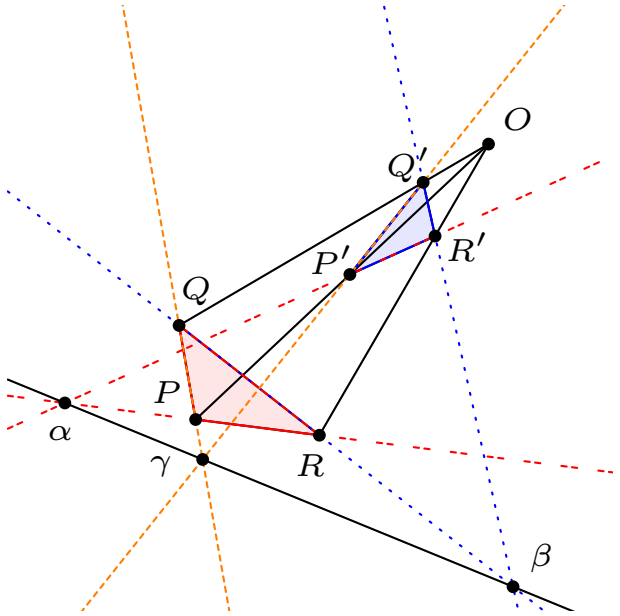


FIGURE III.2.15 – Illustration du théorème III.2.1.

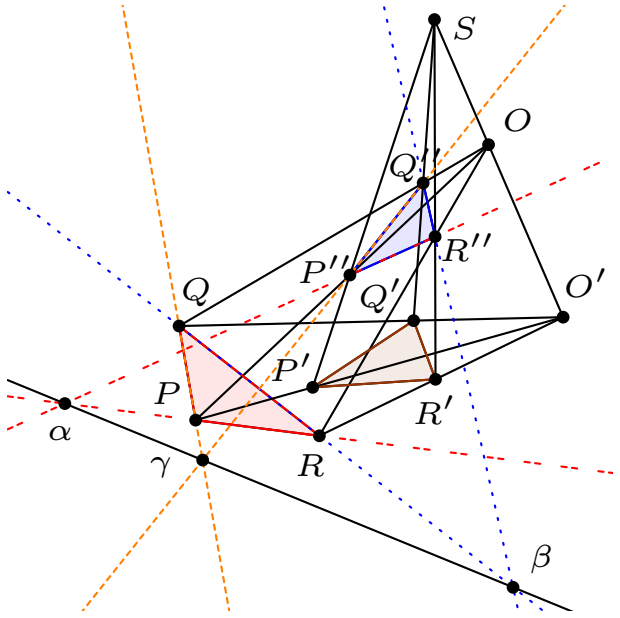


FIGURE III.2.16 – Illustration du théorème III.2.1 avec extrusion de la figure 2D.

Démonstration. **Preuve du théorème de Desargues en 3D**

Nous considérons deux triangles non dégénérés PQR et $P'Q'R'$ qui sont en perspectives par rapport au point O'. Pour éviter les cas dégénérés, nous spécifions que les points P et P', Q et Q', R et R' sont distincts deux à deux. Ces deux triangles non coplanaires permettent d'obtenir une figure dans l'espace où nous définissons les trois points α, β et γ de la manière suivante :

α est l'intersection entre les droites PR et PR',
β est l'intersection entre les droites PQ et PQ',
γ est l'intersection entre les droites QR et $Q'R'$.

Sous ces contraintes, **nous devons montrer que α, β et γ sont alignés**.

• Montrons premièrement que $PQR\alpha$ est un plan :

Sachant que le triangle PQR non dégénéré définit un plan et que le point α appartient à la droite PR, la figure $PQR\alpha$ détermine toujours un plan. Avec un raisonnement analogue, il est possible de déduire le même résultat pour $PQR\beta$ et $PQR\gamma$.

• Montrons deuxièmement que $PQR\alpha\beta$ est un plan :

Sachant que $PQR\alpha$ représente un plan et que le point β appartient à la droite PQ, la figure $PQR\alpha\beta$ définit toujours un plan.

• Montrons ensuite que $PQR\alpha\beta\gamma$ est un plan :

Sachant que $PQR\alpha\beta$ est un plan et que le point γ appartient à la droite QR, la figure $PQR\alpha\beta\gamma$ détermine toujours un plan. Avec un raisonnement purement analogue, nous montrons que $P'Q'R'\alpha\beta\gamma$ définit aussi un plan.

• D'autre part, montrons que $PQRP'Q'R'\alpha\beta\gamma$ est un espace 3D :

La figure $PQRP'Q'R'$, contenant les deux triangles qui sont non coplanaires, détermine un espace. En ajoutant les différents points d'intersections α, β et γ entre les droites contenues dans cet espace, il est assez simple de prouver que $PQRP'Q'R'\alpha\beta\gamma$ définit toujours un espace.

• Montrons finalement que $\alpha\beta\gamma$ est une droite :

Rappelons qu'en géométrie spatiale projective, l'intersection entre deux plans non confondus est une droite. Nous déduisons à partir de ce théorème que l'intersection entre les plans $PQR\alpha\beta\gamma$ et $P'Q'R'\alpha\beta\gamma$ est une droite. En effet, ces deux plans ne sont pas confondus puisque $PQRP'Q'R'\alpha\beta\gamma$ représente un espace. De plus, on peut remarquer que ces deux plans possèdent plusieurs points en commun : α, β et γ. Ces points appartiennent par conséquent à l'intersection entre ces deux plans. Nous pouvons conclure que $\alpha\beta\gamma$ sont situés sur cette droite d'intersection et que ces trois points sont alignés.

Preuve du théorème de Desargues avec extrusion de la figure 2D

La majorité des hypothèses ne sont pas modifiées dans la version 2D. Cependant, nous spécifions cette fois-ci que $PQRP'Q'R'$ définit un plan et que les deux triangles sont en perspectives par rapport au point O'.

Sous ces contraintes, **nous devons toujours montrer que α, β et γ sont alignés**.

Pour cela, nous devons élever le triangle $P'Q'R'$ en un nouveau triangle $P''Q''R''$ qui n'est pas coplanaire avec le triangle PQR afin d'obtenir une configuration de points où le théorème 3D de Desargues puisse être appliqué. Nous détaillons les principales étapes de cette construction. Nous commençons par construire un point S à l'extérieur du plan $PQRP'Q'R'P$. Nous construisons ensuite la droite SO' en considérant un troisième point O sur cette droite distinct de S et O' grâce à l'axiome *Three-Points* de la Table I.1.6. Puis, nous construisons P'' qui est l'intersection entre les droites SP' et OP. Cette intersection existe grâce à l'axiome de *Pasch* et le fait que PP' et SO se coupent en O'. Nous construisons de la même manière Q'' et R''.

Pour appliquer le théorème de Desargues 3D sur cette figure extrudée, nous devons prouver que cette dernière n'est pas une configuration dégénérée et que le triangle $P''Q''R''$ n'est pas aplati. Nous devons aussi vérifier que les intersections α, β et γ sont bien définies et correspondent aux points de la version 3D. □

```
Lemma desargues_3D : forall P Q R P' Q' R' O alpha beta gamma : Point,
rk(P, P') = 2 -> rk(Q, Q') = 2 -> rk(R, R') = 2 ->
rk(P, P', O) = 2 -> rk(Q, Q', O) = 2) -> rk(R, R', O) = 2 ->
rk(P, Q, R) = 3 -> rk(P', Q', R') = 3 -> rk(P, Q, R, P', Q', R') = 4 ->
rk(P, Q, beta) -> rk(P', Q', beta) ->
rk(P, R, alpha) -> rk(P', R', alpha) ->
rk(Q, R, gamma) -> rk(Q', R', gamma) ->
rk(alpha, beta, gamma) = 2.
```

Table III.2.11 – Énoncé mathématique associé au théorème III.2.1.

```
Lemma desargues_2D : forall P Q R P' Q' R' P'' Q'' R'' O' O S alpha beta gamma : Point,
rk(P, P') = 2 -> rk(Q, Q') = 2 -> rk(R, R') = 2 ->
rk(P, P', O) = 2 -> rk(Q, Q', O) = 2) -> rk(R, R', O) = 2 ->
rk(P, Q, R) = 3 -> rk(P', Q', R') = 3 -> rk(P, Q, R, P', Q', R', O') = 3 ->
rk(P, Q, R, S) = 4 -> rk(P, Q, R, O) = 4 ->
rk(S, O) = 2 -> rk(O', O, S) = 2 ->
rk(P, P'', O) = 2 -> rk(Q, Q'', O) = 2 -> rk(R, R'', O) = 2 ->
rk(P', P'', S) = 2 -> rk(Q', Q'', S) = 2 -> rk(R', R'', S) = 2 ->
rk(P, Q, beta) = 2 -> rk(P'', Q'', beta) ->
rk(P, R, alpha) = 2 -> rk(P'', R'', alpha) ->
rk(Q, R, gamma) = 2 -> rk(Q'', R'', gamma) ->
rk(alpha, beta, gamma) = 2.
```

Table III.2.12 – Énoncé mathématique associé au théorème III.2.1 avec extrusion de la figure 2D.

3.2 Résultats

Nous détaillons les résultats obtenus pour la preuve du théorème de Desargues en 3D dans le cas général et pour l'extrusion de la figure 2D. La Table III.2.13 représente les tests de performances obtenus avec le prouveur en supprimant les hypothèses le plus tôt possible, sans coloration et sans subdivision. La preuve de la version 2D nécessite 3 minutes pour générer une preuve d'environ 6 000 lignes qui occupera 10% de la mémoire lors de sa validation. Les temps d'exécution et l'occupation de la mémoire ne sont pas encore critiques.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	
desargues_3D (10 pts)	0.120s	<1%	3	679	3.4s	2%
desargues_2D (15 pts)	2m42s	<1%	4	6146	50s	10.8%

TABLE III.2.13 – Test de performance pour la preuve de Desargues en 3D.

Nous publions dans la Table III.2.14 l'évolution de ces résultats en utilisant l'heuristique de coloration. Le temps d'exécution de la preuve est ainsi amélioré dans les deux cas. Notons que la taille de la preuve *desargues_3D* augmente de manière non négligeable entraînant une vérification un peu plus longue et plus coûteuse en mémoire. Ce changement de taille dépend du cheminement que le prouveur utilise pour établir le résultat.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
desargues_3D (10 pts)	0.092s	<1%	4	1001	7.6s	2.8%
desargues_2D (15 pts)	2m1s	<1%	3	6096	1m8s	12.0%

TABLE III.2.14 – Test de performance avec coloration de la preuve de Desargues en 3D.

Pour illustrer la différence d'occupation mémoire lors de la validation de la preuve, nous présentons dans la Table III.2.15 les preuves de la Table III.2.13 sans la suppression des hypothèses au fur et à mesure de la preuve. Pour la démonstration *desargues_3D*, ce changement n'a que très peu d'impact, nous constatons seulement une augmentation de 0.7% de l'occupation mémoire. La taille de la preuve Coq n'est pas encore assez conséquente. Dans le cas de la preuve *desargues_2D*, nous observons une augmentation significative de la mémoire occupée, elle est multipliée par un facteur 4. En parallèle, le temps d'exécution de la vérification de la preuve par l'assistant de preuve Coq est lui aussi directement impacté. La gestion des hypothèses inutiles devient nécessaire pour vérifier des preuves atteignant plusieurs milliers de lignes.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
desargues_3D (10 pts)	0.120s	<1%	3	679	3.9s	2.7%
desargues_2D (15 pts)	2m42s	<1%	4	6146	4m14s	41.5%

TABLE III.2.15 – Test de performance sans suppression d'hypothèses pour la preuve de Desargues en 3D.

Par ailleurs, la preuve du théorème de Desargues dans le plan en considérant la propriété de Pappus est actuellement considérée dans le but de vérifier le théorème de Hessenberg I.1.1 introduit dans le Chapitre I.1. Cette preuve difficile nécessite une application triple de la propriété de Pappus dans le cas spécifique où les deux triangles de la configuration géométrique forme un triangle cévian [4] [MNS12].

4. Voir http ://mathworld.wolfram.com/CevianTriangle.html

4 Conjugué harmonique

Le théorème majeur suivant que nous prouvons automatiquement est la propriété du conjugué harmonique. Dans le plan euclidien, cette configuration géométrique s'intéresse au birapport [5] entre quatre points disposé sur une même droite. Lorsque ce birapport est égal à -1, on dit que les quatre points sont en division harmonique. Le quatrième point est alors appelé le conjugué du troisième point par rapport aux deux premiers. Comme la propriété de Desargues, la propriété du conjugué harmonique n'est pas forcément vérifiée par un plan projectif mais elle devient un théorème lorsque ce plan est plongé dans un espace de dimension ≥ 3.

4.1 Preuve du conjugué harmonique

En géométrie projective, le conjugué harmonique d'un triplet de points sur la droite projective est défini par la construction suivante :

Théorème III.2.2 (Théorème du Conjugué Harmonique)**.** *Soit E un espace projectif et A, B, C trois points colinéaires de l'espace. Soit R un point extérieur à la droite AB et soit une droite quelconque issue de C coupant RA et RB en Q et P respectivement. Si AP et BQ se coupent en U et RU coupe AB en D, alors D est indépendant du choix de R et de la droite CPQ. Le point D est appelé le conjugué harmonique de C par rapport à A et B.*

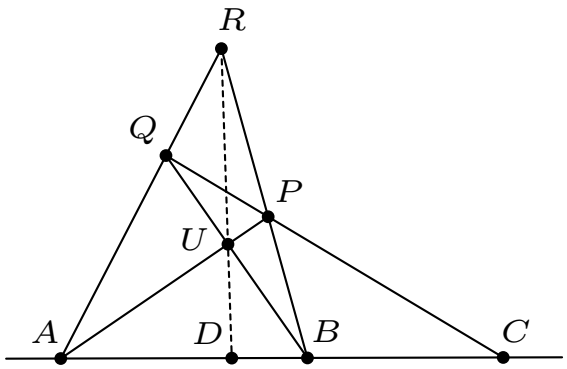


FIGURE III.2.17 – Illustration du théorème III.2.2.

Démonstration. Pour montrer que le point D reste identique peu importe la position du point R et quel que soit le choix des droites QR, RP et PQ, nous procédons comme suit en utilisant la démonstration du livre *Principles of Geometry* [Bak25].

5. Si A, B, C, D sont quatre points distinct d'une droite, on appelle birapport de (A, B) et (C, D) le rapport des mesures algébriques suivant : $r = \frac{\frac{\overline{CA}}{\overline{CB}}}{\frac{\overline{DA}}{\overline{DB}}}$.

Nous construisons de manière semblable à partir de la droite AB un autre plan non confondu avec le plan $ABCR$. Nous obtenons les droites $Q'R'$, $R'P'$ et $P'Q'$ passant respectivement par A, B et C comme illustré dans la Figure III.2.18. Nous souhaitons montrer que les points D et D' qui sont les intersections des droites RU et $R'U'$ avec la droite AB sont identiques.

Dans ce but, nous déduisons que l'intersection entre le plan RAR' et le plan $QPCP'Q'$ est la droite QQ'. Similairement, nous montrons que l'intersection entre le plan RBR' et le plan $QPCP'Q'$ est la droite PP'. Ces droites PP' et QQ' appartiennent au même plan et se coupent nécessairement grâce à l'axiome de *Pasch*. Ce point d'intersection doit être obligatoirement situé sur la droite RR' contenant tous les points communs aux plans RAR' et RBR'. Notons O ce point d'intersection entre les droites PP' et QQ' [6].

D'autre part avec un raisonnement analogue, nous déduisons que l'intersection entre le plan PAP' et le plan $QQ'OP'P$ est la droite PP' et que QQ' est l'intersection entre les plans QBQ' et $QQ'OP'P$. Ces droites PP' et QQ' appartenant au même plan se coupent en un point situé sur la droite d'intersection UU' entre les plans PAP' et QBQ'.

Grâce à l'unicité de l'intersection, nous déduisons que les droites PP' et QQ' se coupent en un point O à la fois situé sur la droite UU' et RR', nous montrons ainsi que UU' et RR' résident dans un même plan. Sachant que les droites UU' et RR' sont concourantes, les droites RU et $R'U'$ appartenant respectivement aux plans RAB et $R'AB$ sont elles aussi concourantes et leur point d'intersection se situe sur la droite AB. Les points D et D' coïncident (voir Figure III.2.19.

Maintenant supposons la construction d'autres droites dans le plan PQR passant par A, B et C. Plus précisément, AQ_1R_1 passant par A, BR_1P_1 passant par B et CP_1Q_1 passant par C. Soient AP_1 et BQ_1 deux droites concourantes en U_1 et la droite R_1U_1 qui rencontre la droite AB en D_1. Par la déduction précédente, on peut montrer que D_1 coïncide avec le point D' qui lui-même coïncide avec le point D. Nous démontrons ainsi que la position du point D est indépendante du choix des droites AQR, BRP et CPQ. □

6. Pour des soucis de lisibilité et de taille de la figure du conjugué harmonique, le point O n'est pas représenté.

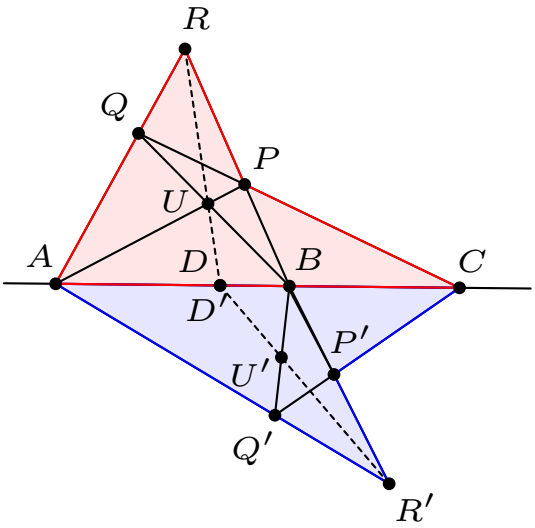


FIGURE III.2.18 – Illustration de la preuve du théorème III.2.2.

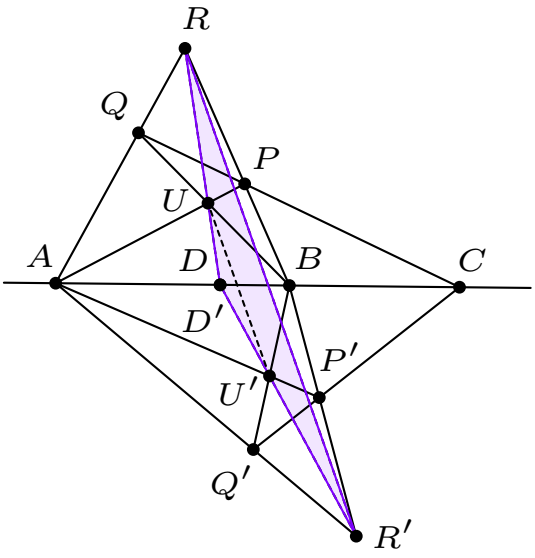


FIGURE III.2.19 – Illustration de la preuve du théorème III.2.2.

```
Lemma harmonic_conjugate : forall A B C P Q R P' Q' R' U U' D D' O
rk(A, B, C) = 2 -> rk(A, B) = 2 -> rk(A, C) = 2 -> rk(B, C) = 2 ->
rk(P, Q, R) = 3 -> rk(P', Q', R') = 3 -> rk(A, B, R, R') = 4 ->
rk(A, B, R) = 3 -> rk(A, B, Q) = 3 ->
rk(A, Q, R) = 2 -> rk(B, P, R) = 2 -> rk(C, P, Q) = 2 ->
rk(A, P, U) = 2 -> rk(B, Q, U) = 2 -> rk(D, R, U) = 2 ->
rk(A, B, R') = 3 -> rk(A, B, Q') = 3 ->
rk(A, Q', R') = 2 -> rk(B, P', R') = 2 -> rk(C, P', Q') = 2 ->
rk(A, P', U') = 2 -> rk(B, Q', U') = 2 -> rk(D', R', U') = 2 ->
rk(A, B, C, D, D') = 2 -> rk(P, P', O) = 2 -> rk(Q, Q', O) = 2 ->
rk(D,D') = 1.
```

TABLE III.2.16 – Énoncé mathématique associé au théorème III.2.2.

4.2 Résultats

La preuve de ce lemme à 14 points est comparable à celle de *desargues_2D*. Le temps d'exécution avoisine la minute pour une preuve de 7437 lignes occupant 12% de la mémoire Coq.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
harmonic (14 pts)	47s	<1%	5	7437	53s	12.0%

TABLE III.2.17 – Test de performance pour la preuve du conjugué harmonique.

Pour la version avec coloration, nous obtenons dans la Table III.2.19 un meilleur temps d'exécution au détriment d'une preuve générée qui est beaucoup plus longue que dans le cas d'un parcours linéaire classique. Rappelons que le cheminement retenu ne peut pas être prévu à l'avance, il est dans ce cas bien moins optimal. Cette augmentation de la taille de la preuve influe bien évidemment sur le temps d'exécution et la mémoire utilisée au sein de l'assistant de preuve Coq : 2m31s et 23% respectivement.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
harmonic (14 pts)	21s	<1%	5	12779	2m31s	23.3%

TABLE III.2.18 – Test de performance avec coloration pour la preuve du conjugué harmonique.

Finalement, pour améliorer les performances du logiciel Coq, nous effectuons la démonstration en utilisant le mécanisme de preuves par couche à trois niveaux. Ce mécanisme permet dans notre cas de peu impacter le temps d'exécution du prouveur afin de générer des preuves qui sont plus faciles à valider. Nous obtenons, avec ou sans coloration, des temps d'exécution Coq plus faibles pour une taille de preuve quasiment identique. La principale différence se situe au niveau de l'occupation mémoire qui ne dépasse plus les 5%.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
harmonic (no color)	49s	<1%	10	9156	31s	4.1%
harmonic (color)	26s	<1%	10	10331	36s	4.3%

TABLE III.2.19 – Test de performance avec subdivision pour la preuve du conjugué harmonique.

5 Propriété de Dandelin-Gallucci

Le dernier exemple que nous traitons est une configuration géométrique peu connue appelée propriété de Dandelin-Gallucci. Cette configuration implique plusieurs droites transversales [7] non coplanaires. La preuve de cette dernière est un résultat déjà connu mais la majorité des démonstrations de la littérature utilisent le concept de birapport ou le théorème de Desargues [BP15, Hor17]. Nous examinons dans ce chapitre la preuve de cette propriété à partir de la propriété de Pappus. Cette démonstration détaillée dans *Principles of Geometry* par H. F. Baker [Bak25] peut être généralisée en considérant le théorème de Dandelin-Gallucci qui indique que la propriété du même nom et la propriété de Pappus sont équivalentes en géométrie d'incidence projective. Dans ce manuscrit, seule l'implication de la propriété de Pappus vers la propriété de Dandelin-Gallucci est étudiée.

5.1 Preuve de la propriété de Dandelin-Gallucci

Si l'axiome de Pappus est admis, la proposition suivante est toujours vérifiée :

Théorème III.2.3 (Propriété de Dandelin-Gallucci)**.** *Si trois droites a, b, c quelconques ne se coupent pas deux à deux et sont coupées chacune par trois autres droites a', b', c' alors toute transversale au premier ensemble de droite a, b, c doit couper toute transversale du second ensemble de droites a', b', c'.*

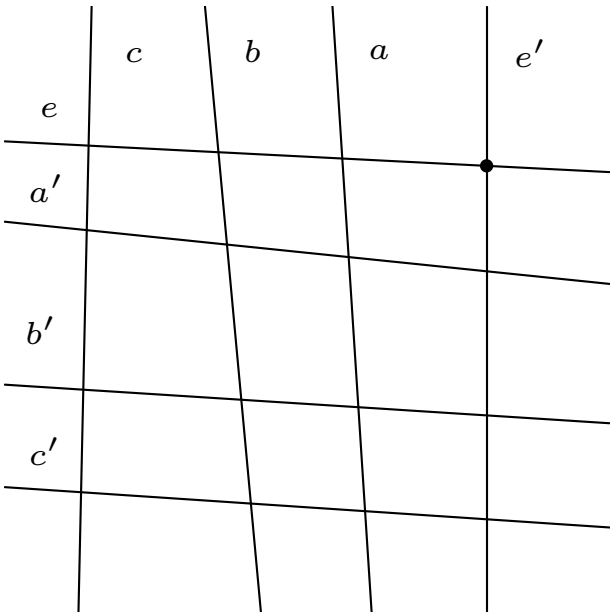


FIGURE III.2.20 – Illustration de la propriété III.2.3.

Démonstration. Soient a, b, c trois droites quelconques qui ne se coupent pas deux à deux et a', b', c' trois autres droites quelconques qui ne se coupent pas deux à deux. Chacune de ces trois droites a', b', c' coupe a, b, c. Les intersections respectives sont P, C, B' en a', puis C', Q, A en b' et B, A', R en c'.

7. Une droite qui coupe un autre ensemble de droites est une droite transversale pour cet ensemble. Il existe deux droites qui coupent quatre droites transversales deux à deux dans un espace projectif.

Construisons ensuite une transversale e par rapport aux droites a, b, c coupant ces dernières respectivement en L, M, N. Cette droite ne coupe jamais les droites a', b', c'. Pour justifier que e n'est pas coplanaire avec a', b' ou c' : si e et a' étaient coplanaires alors les points N, M, B', C' seraient coplanaires impliquant que les droites b et c soient aussi coplanaires. Ce qui est contradictoire.

Puis à partir d'un point N' sur la droite c', construire la transversale e' par rapport aux droites e et a' coupant ces dernières respectivement en O et L'. Nous obtenons la figure III.2.21. Cette transversale e' n'est pas coplanaire avec les droites a, b, c en utilisant un raisonnement analogue à la transversale e. Comme la droite c' ne coupe pas la droite e, la droite e' ne coupe pas les droites a, b, c. Il s'agit de montrer que cette transversale e' coupe la droite b'

La droite RP n'appartient pas au plan $[e, e']$ sinon R, P, L', N' seraient coplanaires impliquant que les droites a', c' soient aussi coplanaires, ce qui est contradictoire. La droite RP coupe à la fois la droite $N'L$ dans le plan $[a, c']$ et la droite NL' dans le plan $[a', c]$. Cette droite RP représente la droite d'intersection entre les plans $[a, c']$ et $[a', c]$. De plus, nous savons que les droites $N'L$ et NL' appartiennent au plan $[e, e']$ et donc qu'il existe une intersection non vide entre ces droites. Cette intersection appartient à la droite RP lorsque cette droite rencontre le plan $[e, e']$.

La transversale e' n'appartient pas au plan $[a, b']$ puisqu'elle ne coupe pas la droite a. Supposons que l'intersection entre la droite e' le plan $[a, b']$ est un point M' (voir Figure III.2.22). Montrons que ce point M' appartient forcément à la droite b'.

La droite PQ n'appartient pas au plan $[e, e']$ sinon P, Q, M, L seraient coplanaires impliquant que les droites b, a soient aussi coplanaires, ce qui est contradictoire. La droite PQ coupe à la fois la droite LM' dans le plan $[a, b']$ et la droite $L'M$ dans le plan $[a', b]$. Cette droite PQ représente la droite d'intersection entre les plans $[a, b']$ et $[a', b]$. De plus, nous savons que les droites LM' et $L'M$ appartiennent au plan $[e, e']$ et donc qu'il existe une intersection non vide entre ces droites. Cette intersection appartient à la droite PQ au moment où cette droite rencontre le plan $[e, e']$.

Les points P, Q, R ne sont pas alignés étant donné que les droites $N'L$, NL' et $L'M$ ne coupent pas en un point unique (sauf dans le cas dégénéré où le point N' est confondu avec le point B). Ces points définissent donc dans le cas général un plan. Il existe donc une droite d'intersection entre les plans PQR et $[e, e']$.

L'application de l'axiome de Pappus aux deux triplets de points sur la droite L, M, N et sur la droite L', M', N' aboutit à une droite dans le plan $[e, e']$ contenant les trois points suivants qui sont des intersections de droites : $(LM', L'M)$, $(MN', N'M)$, $(NL', N'L)$. Nous avons montré que le premier et le troisième points appartiennent respectivement à PQ et PR. Le second point $(MN', N'M)$ appartient donc aussi au plan PQR en étant sur la même droite que le premier et troisième points.

La droite MN' n'appartient pas au plan PQR sinon MN' couperait PQ en un point différent de Q. La droite PQ posséderait alors deux points dans le plan $[b, c']$ et appartiendrait à ce dernier. La droite PB ou a qui serait donc incluse dans le plan $[b, c']$ couperait nécessairement la droite b, ce qui est contradictoire. Cependant la droite MN' du plan $[b, c']$ coupe QR qui est une droite de ce même plan. Sachant que le point d'intersection $(MN', N'M)$ appartient au plan PQR, que MN' n'appartient pas au plan PQR et que MN' coupe QR qui est une droite de

PQR, nous déduisons que la droite $M'N$ coupe elle aussi QR.

Par conséquent, la droite $M'N$ appartient au plan NQR ou $[b', c]$. Par construction, le point M' appartient aussi au plan $[a, b']$. Nous déduisons que le point M' appartient à la droite b'. La transversale e' construite à partir de N' intersectant e et a' est la même transversale que celle qui est construite à partir de N' pour rencontrer a' et b' ; cette dernière coupe donc aussi e.

Nous montrons ainsi que toute transversale du premier ensemble de droites a, b, c coupe toute transversale du second ensemble de droites a', b', c'. □

Par cette démonstration, nous prouvons qu'à partir des axiomes de la géométrie projective et de l'axiome de Pappus, il est possible de prouver la propriété de Dandelin-Gallucci impliquant 19 points (16 points pour la configuration géométrique et 3 points pour l'application du théorème de Pappus). La réciproque est aussi vraie, la propriété de Dandelin-Gallucci et les axiomes de la géométrie d'incidence projective permettent de prouver la propriété de Pappus. Cependant, nous ne traitons pas cette implication [Bak25] dans cette thèse. En effet, pour le moment la propriété de Dandelin-Gallucci n'est pas intégrée dans notre prouveur.

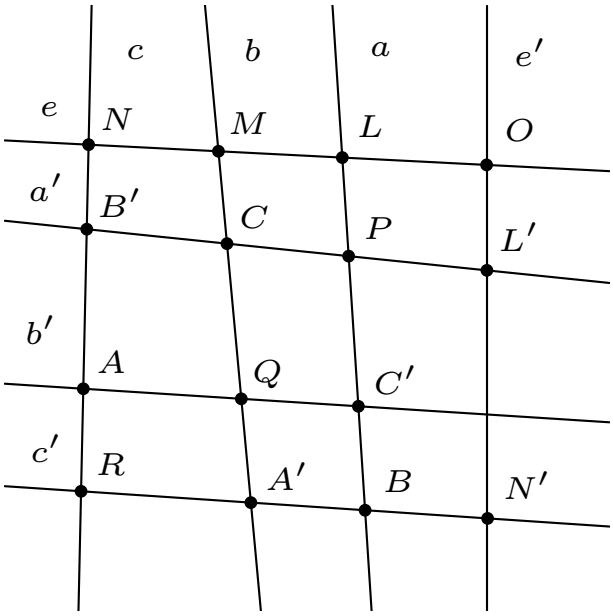


FIGURE III.2.21 – Illustration de la preuve du théorème III.2.3.

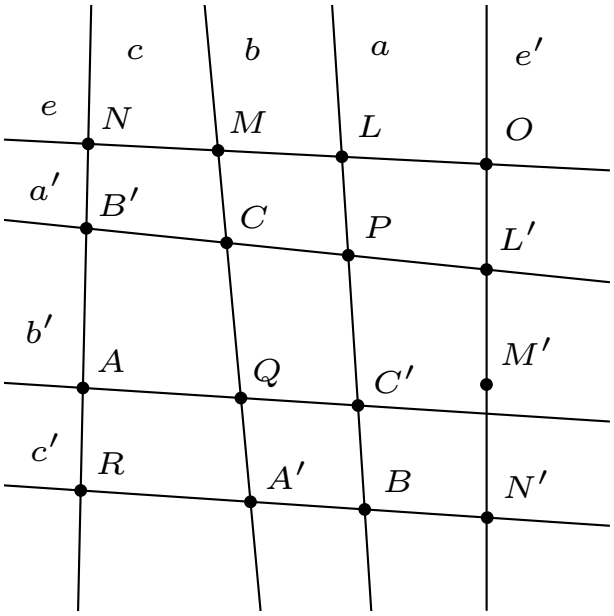


FIGURE III.2.22 – Illustration de la preuve du théorème III.2.3.

```
Lemma dg : forall A A' B B' C C' R P Q N M L N' M' L' O alpha beta gamma,
rk(B, C', P, L) = 2 -> */ line a */
rk(B', C, P, L') = 2 -> */ line a' */
rk(A', Q, C, M) = 2 -> /* line b */
rk(A, Q, C', M') = 2 -> /* line b' */
rk(R, A, B', N) = 2 -> /* line c */
rk(R, A', B, N') = 2 -> /* line c' */
rk(A', B, C, C', P, Q) = 4 -> /* a & b */
rk(A, B, B', C', P, R) = 4 -> /* a & c */
rk(A, A', B', C , Q, R) = 4 -> /* b & c */
rk(A, B', C, C', P, Q) = 4 -> /* a' & b' */
rk(A', B, B', C, P, R) = 4 -> /* a' & c' */
rk(A, A', B, C', Q, R) = 4 -> /* b' & c' */
rk(N, M, L, O) = 2 -> /* line e */
rk(N', M', L', O) = 2 -> /* line e */
rk(A, C', P, Q, B, M') = 3 -> /* plane [a & b'] & M' */
rk(B, C', P, N', M', L') = 4 -> /* a & e' */
rk(A', C, Q, N', M', L') = 4 -> /* b & e' */
rk(A, B', R, N', M', L') = 4 -> /* c & e' */
rk(B', C, P, N, M, L) = 4 -> /* a' & e */
rk(A, C', Q, N, M, L) = 4 -> /* b' & e */
rk(A', B, R, N, M, L) = 4 -> /* c' & e */
rk(N, L', alpha) = 2 -> rk(N', L, alpha) = 2 -> /* premier point Pappus */
rk(M', L, beta) = 2 -> rk(M, L', beta) = 2 -> /* deuxième point Pappus */
rk(N, M', gamma) = 2 -> rk(N', M, gamma) = 2 -> /* troisième point Pappus */
rk(A, C', Q, M') = 2. /* M' appartient à b' */
```

TABLE III.2.20 – Énoncé mathématique associé au théorème III.2.3.

5.2 Résultats

Cette preuve de théorème contenant 19 points pousse un peu plus loin les limites du prouveur. Le temps d'exécution pour trouver une solution est de 37h sans l'heuristique de coloration et de 16h avec cette dernière. Nous obtenons des preuves non divisées de plusieurs dizaines de milliers de lignes qui ne peuvent pas être validées sur notre machine de tests. Le nombre d'hypothèses traitées et la taille du terme de preuve en mémoire sont trop conséquents et empêche la validation. Le système Coq atteint les 80% d'occupation mémoire au bout de 40 000 lignes avant un débordement de la pile.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
dg (no color) (19 pts)	2238m	<1%	10	78116	X	≥80%
dg (color) (19 pts)	943m	<1%	10	105769	X	≥80%

TABLE III.2.21 – Test de performance pour la preuve du théorème de Dandelin-Gallucci.

L'introduction de la subdivision par couches va permettre de résoudre ce problème d'occupation mémoire et d'obtenir ainsi une validation par l'assistant de preuve Coq. Nous définissons ainsi grâce à la construction de l'énoncé géométrique trois couches géométriques. La première encapsule les 9 points situés à l'intersection des a, b, c et a', b', c'. La couche suivante rajoute les points d'intersections avec les droites transversales e et e'. La couche finale considère les trois points qui permettront l'application du théorème de Pappus. L'illustration de ce découpage a déjà servi d'exemple dans la Figure III.1.12 page 130.

Comme anticipé, le temps d'exécution de la recherche d'une solution est légèrement augmenté. Le temps de saturation des deux premières couches reste cependant négligeable par rapport à la dernière couche. Nous obtenons sans/avec heuristique de coloration une preuve découpée en 146/152 lemmes divisés sur 3 couches avec la répartition suivante : couche 1 (79/82 lemmes) - couche 2 (66/69 lemmes) - couche 3 (1/1 lemme).

Le système met cette fois-ci 5 à 7 minutes pour valider une preuve de 50 000 à 70 000 lignes en occupant pas plus de 7.5% de la mémoire.

	T.E prouveur	M. prouveur	Nb. étapes	Taille	T.E Coq	M. Coq
dg (no color) (19 pts)	2252m	<1%	15	48741	5m12s	7.6%
dg (color) (19 pts)	918m	<1%	10	71508	8m37s	7.4%

TABLE III.2.22 – Test de performance avec subdivision de la preuve du théorème de Dandelin-Gallucci.

Ce dernier exemple suggère qu'il est possible de pousser l'automatisation sur des exemples géométriques encore plus conséquents à condition d'améliorer principalement les performances en temps de notre prouveur pour la recherche d'une solution. La subdivision des preuves est généralisable au nombre de points de l'énoncé géométrique, ce qui permet de maîtriser la taille des démonstrations assez simplement. Nous détaillons les résultats obtenus ainsi que nos perspectives dans la conclusion de cette partie.

Conclusion : partie III

Bilan

Dans cette partie, nous avons construit un prouveur généralisé de théorèmes géométriques d'incidence. Ce prouveur divisé en deux parties permet non seulement de rechercher une solution mais aussi de pouvoir mémoriser et reconstruire la preuve associée sous la forme d'un certificat qui sera ensuite validé par l'assistant de preuve Coq. Cet outil apporte un résultat pratique robuste dans un cadre géométrique combinatoire s'appuyant sur la théorie des matroïdes. Il est en effet possible de prouver des configurations géométriques complexes de dimension 2 ou 3 en utilisant uniquement les propriétés matroïdales. Pour cela, ces propriétés sont converties en règles de réécritures avant d'être intégrées dans un algorithme de saturation. L'algorithme sature l'énoncé géométrique jusqu'à l'obtention du résultat ou que les rangs minimum et maximum de tous les ensembles de points aient convergé ou la détection d'une contradiction de l'énoncé. L'export de l'historique qui a été suivi en appliquant les règles de réécritures permet d'apporter une aide à la preuve dans ce contexte géométrique qui serait plus difficilement envisageable en raisonnant uniquement avec une approche en géométrique synthétique.

Le développement de ce prouveur à partir de la formalisation matroïdale de la géométrie d'incidence repose sur deux observations importantes déjà exprimées dans les chapitres précédents.

La première, introduite dès la Partie I, indique que l'unification de tous les concepts de la géométrie d'incidence peu importe la dimension sous la même fonction de rang permet de faciliter le développement des procédés d'automatisation en ne considérant qu'un type d'objet et de simplifier l'automatisation des démonstrations dans sa globalité. En effet, le coeur de la mécanisation de la démonstration en géométrie d'incidence projective se situe au niveau du calcul de l'inclusion, de l'union et de l'intersection entre les différents objets représentés par des points. Ces calculs ont l'avantage d'être naturellement inclus dans les propriétés matroïdales contrairement à la géométrie synthétique où cette mécanisation n'est pas triviale. Il est nécessaire d'énoncer chacune de ces opérations dans un lemme distinct complexifiant excessivement la découverte automatique de nouvelles déductions.

La deuxième observation, liée à la première, est une conséquence de notre hypothèse forte de non création d'objets et plus spécifiquement de nouveaux points. Cette contrainte nous empêche de considérer tout axiome de la Partie I contenant un quantificateur existentiel pour effectuer des déductions systématiques. Aucun axiome mis à part l'unicité ne remplit cette condition du point de vue de la géométrie synthétique, nous devons construire une couche de lemmes intermédiaires capturant ce qui est tout naturellement apporté par la théorie des matroïdes et ses propriétés sur la fonction de rang définies sans quantificateur existentiel.

En incorporant des résultats déjà obtenus ou des théorèmes fondamentaux en plus des règles réécritures, il est possible d'élargir les capacités de déductions de notre prouveur. Nous testons cette option en incluant une des propriétés centrales de la géométrie d'incidence à savoir la propriété de Pappus. Cette propriété va naturellement agrandir la classe des problèmes que l'on

peut traiter et ainsi accroître l'intérêt pour de tels outils afin de faciliter le raisonnement sur des parties simples de la théorie. Cependant l'ajout d'une nouvelle propriété doit être parfaitement mesuré pour ne pas impacter les performances du prouveur. Pour éviter d'alourdir inutilement la saturation, la recherche d'une configuration de Pappus, qui est une tâche coûteuse, doit être effectuée uniquement lorsqu'aucune déduction avec les règles matroïdales n'est possible. Un nouveau résultat ne doit pas être ajouté à la boucle de saturation si ce dernier n'est utilisé que très rarement. À noter que la règle de Pappus qui est incluse, vaut pour à la fois la version affine et la version projective de cette propriété puisqu'il n'est pas nécessaire d'identifier si les droites se coupent ou si celles-ci sont parallèles pour appliquer cette règle. Comme indiqué dans l'introduction de cette partie, cet outil permet de traiter à la fois des configurations affines et projectives en incluant uniquement des axiomes d'incidences qui sont communs aux deux visions de la géométrie. Le caractère projectif de la configuration géométrique est défini lors de la construction de chacun des points de l'énoncé. Cette construction complète de la configuration géométrique reste d'ailleurs la principale difficulté pour l'utilisateur puisqu'il va devoir donner l'intuition de la preuve en créant tous les points de l'énoncé avant de lancer le prouveur.

Le premier prototype du prouveur généralisé s'appuyant sur la saturation du problème avec les règles matroïdales permet de mécaniser sans aucune aide tout un ensemble de configurations que nous présentons dans le début de notre catalogue d'exemples. Nous déterminons néanmoins rapidement deux problèmes empêchant de considérer la résolution automatique de preuves comportant plus d'une dizaine de points. L'étude des performances confirme ces deux goulots d'étranglements dans le pipeline de notre prouveur. La première limite est le temps d'exécution exponentiel pour la production du certificat. La saturation du problème à partir de l'ensemble des parties est une technique coûteuse mais nécessaire lorsque le cheminement de la preuve n'est pas prédictible. Pour mieux maîtriser l'explosion de ce temps, les quelques améliorations que nous présentons dans cette thèse s'intéressent à l'optimisation de la boucle de saturation. La principale optimisation que nous présentons dans cette thèse est une heuristique de coloration qui permet de considérer uniquement les parties susceptibles d'aider à la saturation en commençant par les hypothèses de l'énoncé. Le deuxième facteur contraignant est l'explosion de l'occupation mémoire lors de la vérification de la trace. Si la preuve contenue dans cette trace n'est pas correctement segmentée, l'assistant de preuve Coq peut être rapidement débordé. La limite identifiée tourne autour d'une preuve d'une dizaine de milliers de lignes composée de quelques centaines d'hypothèses. Pour pallier ce problème, nous ajoutons un mécanisme de scission automatique de la preuve en un ensemble de lemmes intermédiaires. La scission s'appuie sur un découpage de l'énoncé géométrique en plusieurs couches de points qui sont saturées successivement en réinjectant les déductions précédentes. Ce mécanisme de division de la preuve, généralisable à n couches où n est le nombre de points, élimine définitivement la contrainte d'occupation mémoire et permet d'envisager l'automatisation de démonstrations bien plus conséquentes. Les preuves obtenues deviennent bien plus courtes, réduisant ainsi la taille du terme de preuve. L'analyse de la configuration de Dandelin-Galluci contenant 19 points permet de constater que notre prouveur arrive à produire et traiter automatiquement une démonstration avoisinant les 100 000 pas de preuves. La validation d'une démonstration aussi conséquente et complexe nécessite inévitablement l'aide de l'ordinateur pour être vérifiée.

Par ailleurs, l'analyse du comportement de notre prouveur révèle une grande proximité avec l'étude des problèmes de contraintes géométriques [JTNM06, MFLS06]. En fonction de l'avancée de la saturation, le système est en mesure de nous indiquer si l'énoncé n'est pas assez contraint, bien-contraint ou sur-contraint. Dans le cas où la configuration est sous-contrainte, le système est incapable d'obtenir le résultat recherché en incidence pure à partir des hypothèses. L'énoncé doit être complété avec de nouvelles hypothèses, ou l'ajout d'un théorème de haut-niveau non inclus dans la boucle de saturation. Pour le cas bien-contraint, le prouveur est en mesure de générer une

démonstration validant le résultat recherché. La dernière option est plus difficilement détectable : il faut lancer la résolution du problème en éliminant les hypothèses inutiles pour s'apercevoir que le problème reste soluble. Le fonctionnement de notre algorithme de saturation se rapproche aussi du domaine de la propagation des contraintes [VHDT92] où le but est de maintenir une cohérence locale entre les différents arcs. Cette propagation des contraintes fait partie des pistes possibles pour optimiser notre solveur.

Perspectives

Dans le but d'optimiser toujours plus le fonctionnement du prouveur, nous évoquons un ensemble de pistes à explorer pour améliorer les performances de ce dernier. La première idée à étudier est la fusion des points quand le système détecte que ces derniers sont égaux. De cette manière, nous éliminons facilement un point à traiter de l'ensemble des parties pour simplifier efficacement l'algorithme de saturation. Cette suppression nécessite des ajustements au niveau du prouveur : décalage des parties après suppression du point, copie des résultats existants et propagation de l'égalité entre les points. La principale difficulté apparaît lors de la vérification du certificat puisque l'assistant de preuve considère toujours les deux points comme deux objets distincts bien qu'ils soient égaux. Il est nécessaire de propager l'égalité dans chacun des lemmes intermédiaires produits après la mise en évidence de l'égalité.

Ensuite, dans le prototype actuel, le découpage de la preuve en plusieurs couches est réalisé manuellement en utilisant l'intuition géométrique. Pour automatiser ce découpage, il est par exemple possible d'ajouter un indicateur pour chaque noeud précisant le nombre de déductions qu'il est nécessaire de faire pour arriver jusqu'au résultat contenu dans ce noeud. Sachant qu'à partir d'un nombre de déductions, il est possible de calculer le nombre de blocs et donc d'approcher le nombre de lignes de preuves, il suffit de scinder en deux la couche courante lorsqu'un des indicateurs excède une limite imposée tout en transférant les résultats déjà calculés.

Une perspective plus lointaine que nous souhaitons mettre en place est la programmation en parallèle du mécanisme de saturation. En considérant une mémoire partagée entre tous les processeurs, l'application des règles de saturation avec différentes heuristiques et plusieurs ordres de parcours sur des processeurs en parallèle avec une gestion des désaccords permettrait d'obtenir un gain de temps non négligeable lors de la production du certificat. Une division fine et sans conflits des calculs est nécessaire afin d'éviter que la saturation ne devienne incomplète en oubliant l'application de certaines règles. Une parallélisation plus évidente consiste à diviser l'application des règles de réécritures sur plusieurs processeurs ou de scinder le problème géométrique en cluster de points variables afin de limiter les conflits entre processeurs.

Du point de vue des mathématiques, nous établissons que les axiomes de la géométrie d'incidence projective et la propriété de Pappus permettent de prouver la propriété de Dandelin-Gallucci. L'étude de la réciproque est un résultat tout aussi intéressant à réaliser. Pour cela, nous voulons étendre notre prouveur en rajoutant directement en tant que règle la configuration de Dandelin-Galluci.

Dans tous nos travaux, nous supposons que la création de nouveaux points dans une configuration géométrique est une tâche à la fois hors de propos mais aussi complexe à réaliser. L'idée d'identifier des configurations géométriques à partir d'une base de donnée ou de laisser une intelligence artificielle et un réseau de neurones construire automatiquement des points pertinents mais nécessaires à la démonstration est une piste à long terme.

Cette conclusion de Partie III clôture le coeur de notre manuscrit. Nous résumons le fil rouge de cette thèse ainsi que l'ensemble de nos contributions dans la Conclusion globale qui suit.

Conclusion globale

Synthèse

Dans cette thèse, nous nous sommes focalisés sur l'aide à la preuve en contexte géométrique au sein de l'assistant de preuve Coq. Nous avons non seulement proposé une méthodologie et des procédures visant à automatiser des démonstrations dans l'environnement spécifique de la géométrie d'incidence projective, mais nous avons aussi dégagé un ensemble de grands principes pour une mécanisation systématique des morceaux de preuves les plus récurrents dans un cadre plus général.

Nous avons formalisé dans ce but une librairie Coq sur la géométrie d'incidence projective en manipulant deux approches axiomatiques distinctes mais complémentaires de la géométrie. La première, communément appelée géométrie synthétique, est fondée sur un système d'axiomes classique bien décrit dans la littérature. La seconde, plus originale s'appuie sur les aspects combinatoires de la notion de rang provenant de la structure matroïdale sous-jacente à la géométrie d'incidence. Pour profiter des avantages de chaque approche et justifier une traduction bidirectionnelle, nous avons prouvé l'équivalence entre les deux formalisations de cette géométrie à la fois en 2D, $\geq$3D et 3D. L'analyse méthodologique de la mécanisation de cette preuve nous a offert un bon aperçu des possibilités d'automatisation que nous allions considérer dans la suite de nos travaux. Pour mieux définir ces dernières, nous avons identifié et avons dressé une typologie des éléments de preuves comme la clôture des hypothèses qui apparaissent de manière récurrente dans les démonstrations géométriques. Cette classification nous a permis de mieux cibler les étapes triviales, répétitives et fastidieuses que nous voulions mécaniser.

Nous avons ensuite évalué cette méthodologie de la preuve dans le contexte spécifique et simplifié des géométries finies définies par extension. Cette expérimentation nous a conduit à mettre en évidence un ensemble de critères d'aide à la preuve, généralisables, assurant un meilleur contrôle sur l'explosion combinatoire du nombre de configurations à vérifier lors des preuves de modèles et du théorème de Desargues.

Après avoir examiné attentivement les performances de l'assistant de preuve Coq pour formaliser des géométries finies complètement caractérisées, nous nous sommes intéressés à une automatisation plus générale des démonstrations en géométrie d'incidence. En observant la mécanisation des travaux précédents, nous avons établi que les propriétés matroïdales occupaient un rôle central dans l'automatisation des preuves utilisant cette approche combinatoire. Nous avons exploité ceci pour construire un prouveur automatique de configurations géométriques d'incidence s'appuyant sur une saturation des hypothèses. Ce prototype, basé sur un programme externe en langage C, permet de produire automatiquement un certificat contenant la preuve du théorème traité qui sera ensuite validé par l'assistant de preuve Coq. En optimisant les goulots d'étranglements en temps et en espace de ce prouveur, nous avons réussi à démontrer plusieurs théorèmes classiques et complexes, notamment le théorème de Desargues, la configuration du conjugué harmonique et le théorème de Dandelin-Galucci.

Ouvertures

Au final, cette thèse, résultat d'une combinaison de modélisation géométrique et de génie logiciel apporte plusieurs perspectives attrayantes pour une automatisation plus aisée et plus complète du raisonnement. Elle se situe dans le prolongement de l'axe de recherche initié par Dominique Michelucci et Pascal Schreck en établissant l'équivalence entre la géométrie synthétique et l'approche matroïdale et en proposant une mécanisation systématique des démonstrations dans ce contexte.

Cette thèse suggère que l'utilisation d'un assistant de preuve utilisant en parallèle des mécanismes d'automatisation externes est une solution pragmatique et efficace pour établir des résultats théoriques difficiles ou impossibles à obtenir en utilisant uniquement des prouveurs automatiques. Cette démarche semble être une tendance actuelle au regard des nombreuses publications sur le sujet.

Par ailleurs, toutes les formalisations de la géométrie se fondant sur la géométrie d'incidence, pourraient à terme profiter de cette approche matroïdale afin de mécaniser les fragments du raisonnement portant sur les problèmes d'incidence.

En plus des perspectives spécifiques à chaque contribution de notre thèse qui sont résumées à la fin de chaque fin de partie, nous exposons deux idées majeures à poursuivre pour améliorer et généraliser l'aide à la preuve que nous avons étudié.

D'une part, une intégration directe de notre prouveur externe dans l'assistant de preuve Coq, éventuellement combinée avec des solveurs automatiques déjà incorporés, permettrait d'éliminer de notre pipeline, la plupart des parties qui sont encore accomplies manuellement tout en fournissant un outil directement utilisable par la communauté.

D'autre part, l'outil que nous avons développé n'est pas en capacité de créer de nouveau points pour compléter une configuration initiale. Or dans bien des démonstrations il est indispensable d'introduire de nouveaux points, ce qui nécessite une bonne intuition, tâche reléguée à l'utilisateur humain à ce jour et qui pourrait être confiée à de l'intelligence artificielle dans des travaux futurs.

Quatrième partie

Annexes

Annexe A : systèmes d'axiomes en géométrie synthétique

1 Système d'axiomes 2D

(A1P2) Line-Existence : $\forall\ A\ B : \mathit{Point}, \exists\ l : \mathit{Line}, A \in l \wedge B \in l$

(A2P2) Point-Existence : $\forall\ l\ m : \mathit{Line}, \exists\ A : \mathit{Point}, A \in l \wedge A \in m$

(A3P2) Uniqueness : $\forall\ A\ B : \mathit{Point}, \forall\ l\ m : \mathit{Line},$
$A \in l \wedge B \in l \wedge A \in m \wedge B \in m \Rightarrow A = B \vee l = m$

(A4P2) Three-Points : $\forall\ l : \mathit{Line}, \exists\ A\ B\ C : \mathit{Point},$
$A \neq B \wedge B \neq C \wedge A \neq C \wedge A \in l \wedge B \in l \wedge C \in l$

(A5P2) Lower-Dimension : $\exists\ l\ m : \mathit{Line}, l \neq m$

Table IV.1 – Système d'axiomes standard pour la géométrie projective 2D.

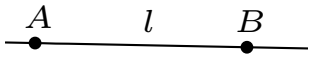
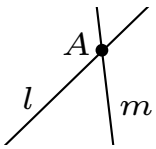
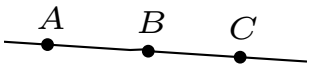
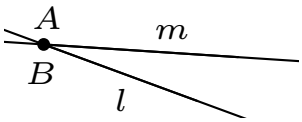
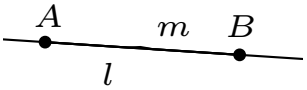
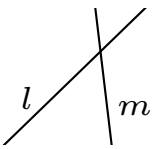
A1P2	A2P2	A3P2
		
A4P2		A5P2
		

Table IV.2 – Illustrations du système d'axiomes standard pour la géométrie projective 2D.

2 Système d'axiomes $\geq$ 3D et 3D

(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 IV.3 – Système d'axiomes pour la géométrie projective 3D.

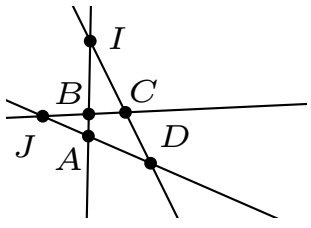
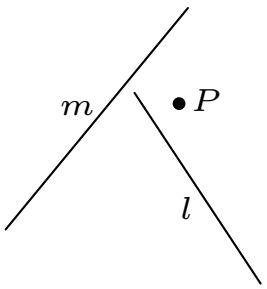
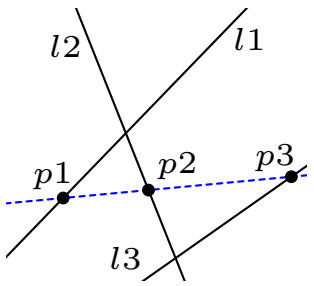
A2P3	A5P3	A6P3
		

TABLE IV.4 – Illustrations du système d'axiomes standard pour la géométrie projective 3D.

Annexe B : systèmes d'axiomes fondés sur la notion de rang

1 Système d'axiomes nD

(A1R2-R3) Rk-SubCardinal : $\forall\ X \subseteq E,\ 0 \leq rk(X) \leq |X|$

(A2R2-R3) Rk-NonDecreasing : $\forall\ X \subseteq Y \subseteq E,\ rk(X) \leq rk(Y)$

(A3R2-R3) Rk-SubModular : $\forall\ X,\ Y \subseteq E,\ rk(X \cup Y) + rk(X \cap Y) \leq rk(X) + rk(Y)$

TABLE IV.5 – Système d'axiomes basé sur les rangs pour la géométrie projective nD.

2 Système d'axiomes 2D

(A4R2) Rk-Singleton : $\forall\ P : Point,\ rk\{P\} \geq 1$

(A5R2) Rk-Couple : $\forall\ P\ Q : Point,\ P \neq Q \Rightarrow rk\{P, Q\} \geq 2$

(A6R2) Rk-Inter : $\forall\ A\ B\ C\ D : Point, \exists\ J : Point,\ rk\{A, B, J\} = rk\{C, D, J\} = 2$

(A7R2) Rk-Three-Points : $\forall\ A\ B : Point, \exists\ C,\ rk\{A, B, C\} = rk\{B, C\} = rk\{A, C\} = 2$

(A8R2) Rk-Lower-Dimension : $\exists\ A\ B\ C : Point,\ rk\{A, B, C\} \geq 3$

TABLE IV.6 – Système d'axiomes basé sur les rangs pour la géométrie projective 2D.

3 Système d'axiomes ≥ 3D et 3D

(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-Dim : $\exists\ A\ B\ C\ D : Point,\ rk\{A, B, C, D\} \geq 4$

TABLE IV.7 – Système d'axiomes basé sur les rangs pour la géométrie projective ≥3D.

(A9R3) Rk-Upper-Dim : $\forall\ A\ B\ C\ D\ E\ F, \exists\ J1\ J2\ J3$
$rk\{A, B\} = 2 \wedge rk\{C, D\} = 2 \wedge rk\{E, F\} = 2 \wedge$
$rk\{A, B, C, D\} \geq 3 \wedge$
$rk\{A, B, E, F\} \geq 3 \wedge$
$rk\{C, D, E, F\} \geq 3 \Rightarrow$
$rk\{A, B, J1\} = 2 \wedge rk\{C, D, J2\} = 2 \wedge$
$rk\{E, F, J3\} = 2 \wedge rk\{J1, J2, J3\} \leq 2$

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

TABLE IV.8 – Système d'axiomes basé sur les rangs pour la géométrie projective 3D.

Annexe C : propriété de Desargues et Pappus

1 Propriété de Desargues

Proprieté (Propriété de Desargues)**.** *Soit E un espace projectif* ≥ 2 *et P, Q, R, P', Q', R' des points de cet espace. Soient PQR et P'Q'R' deux triangles non aplatis et non confondus. Si les droites (PP'), (QQ') et (RR') sont concourantes en un point O alors* α, β *et* γ *sont alignés avec* $\alpha \in (PR) \cap (P'R')$, $\beta \in (QR) \cap (Q'R')$ *et* $\gamma \in (PQ) \cap (P'Q')$.

Description informelle. *Si deux triangles sont en perspectives par rapport à un point, alors ils sont en perspectives par rapport à une droite.*

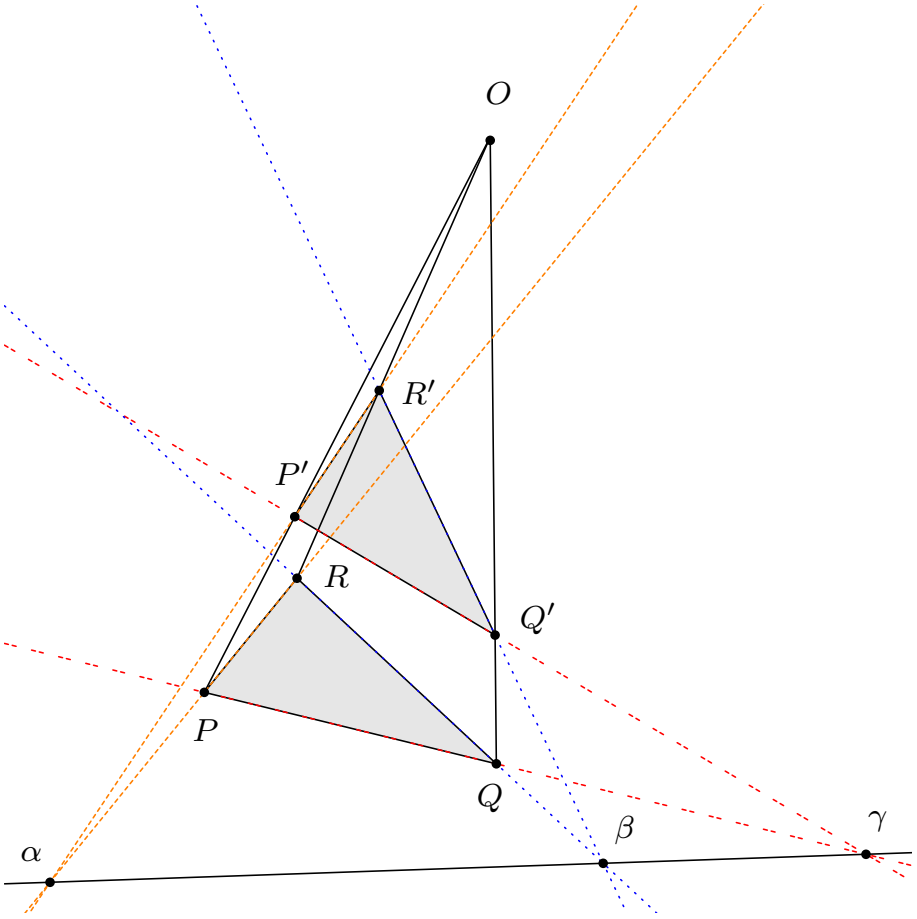


FIGURE IV.1 – Une configuration du théorème de Desargues dans l'espace projectif.

2 Propriété de Pappus

Proprieté (Propriété de Pappus)**.** *Soit E un espace projectif de dimension ≥ 2 et F un sous-espace de E formant un plan. Dans ce plan F, soient P, Q, R trois points distincts alignés sur une droite d, et soient P' Q' R' trois autres points distincts alignés sur une droite d', alors les point α β γ sont alignés avec $\alpha = (RQ') \cap (QR')$, $\beta = (PR') \cap (RP')$ et $\gamma = (PQ') \cap (QP')$.*

Description informelle. *La propriété de Pappus est une configuration à 9 points et 9 droites où chaque droite passe par 3 points et chaque point est l'intersection de trois droites.*

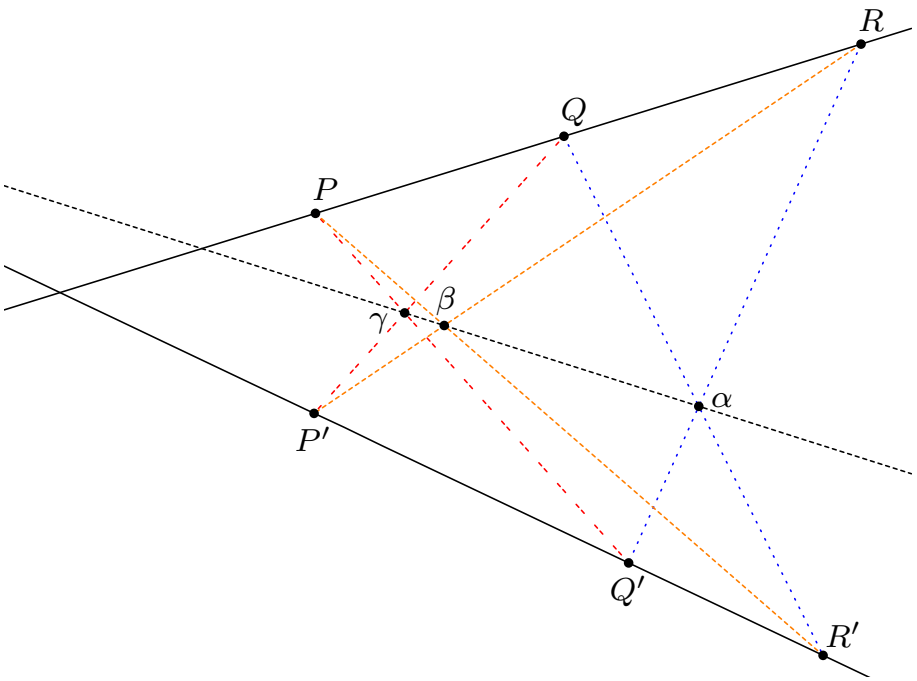


FIGURE IV.2 – Une configuration du théorème de Pappus dans l'espace projectif.

Annexe D : implantation Coq des systèmes d'axiomes sur les rangs

```
(* Types *)
Class MatroidRk '(S : FSetSpecs Point) := {

Point : Set;
rk : set Point -> nat

}.
```

TABLE IV.9 – Classe de types pour la structure projective basé sur les rangs.

```
(* Structure matroïdale *)
Class Matroid '(MR : MatroidRk) := {

(* A1R2-R3 Rk-SubCardinal *)
matroid1  : forall X, rk X >= 0 /\ rk X <= cardinal X;

(* A2R2-R3 Rk-NonDecreasing *)
matroid2 : forall X Y, Subset X Y -> rk X <= rk Y;

(* A3R2-R3 Rk-SubModular *)
matroid3 : forall X Y, rk(union X Y) + rk(inter X Y) <= rk X + rk Y

}.
```

TABLE IV.10 – Classe de types pour la structure matroïdale.

```
(* nD *)
Class RankProjective `(M : Matroid) := {

rk_singleton_ge : forall P, rk (singleton P) >= 1;

rk_couple_ge : forall P Q, ~ P [==] Q -> rk(couple P Q) >= 2;

rk_three_points : forall A B, exists C : Point,
  rk(triple A B C) = 2 /\ rk(couple B C) = 2 /\ rk(couple A C) = 2

}.
```

TABLE IV.11 – Classe de types pour les axiomes indépendants de la dimension avec la notion de rang.

```
(* 2D *)
Class RankProjectivePlane `(RP : RankProjective) := {

rk_inter : forall A B C D, exists J,
  Point, rk (triple A B J) = 2 /\ rk (triple C D J) = 2;

rk_lower_dim : exists P0 P1 P2 : Point, rk (triple P0 P1 P2) >= 3

}.
```

TABLE IV.12 – Classe de types pour le plan projectif avec la notion de rang.

```
(* >=3D *)
Class RankProjectiveSpaceOrHigher `(RP : RankProjective) := {

rk_pasch : forall A B C D, rk (quadruple A B C D) <= 3 ->
  exists J, rk (triple A B J) = 2 /\ rk (triple C D J) = 2;

rk_lower_dim : exists P0 P1 P2 P3, rk (quadruple P0 P1 P2 P3) >= 4

}.
```

TABLE IV.13 – Classe de types pour l'espace projectif au moins en dimension 3 avec la notion de rang.

```
(* 3D *)
Class RankProjectiveSpaceOrHigher `(RP : RankProjective) := {

rk_upper_dim : forall A B C D E F,
rk (couple A B) = 2 /\ rk (couple C D) = 2 /\ rk (couple E F) = 2 /\
rk (quadruple A B C D) >= 3 /\
rk (quadruple A B E F) >= 3 /\
rk (quadruple C D E F) >= 3 ->
exists J1 : Point, exists J2 : Point, exists J3 : Point,
rk(triple A B J1) = 2 /\ rk(triple C D J2) = 2 /\
rk(triple E F J3) = 2 /\ rk(triple J1 J2 J3) <= 2

}.
```

TABLE IV.14 – Classe de types pour l'espace projectif exactement en dimension 3 avec la notion de rang.

Annexe E : implantation Coq de plusieurs définitions récursives

Toutes ces fonctions récursives sont définies à partir d'une liste l, d'un point a et d'un type A.

1 Non égalité

```
Fixpoint contains_two_distinct_points l :=
match l with
| nil => false
| a :: nil => false
| a:: ((b::q) as reste) =>  if (eq_dec_u a b)
                            then (contains_two_distinct_points (reste)) else true
end.
```

Table IV.15 – Définition récursive du prédicat `collinear`.

2 Non colinéarité

```
Fixpoint contains_three_non_collinear_points l :=
match l with
| nil => false
| a::r => if collinear_with_all a (all_pairs r)
          then contains_three_non_collinear_points r else true
end.
```

Table IV.16 – Définition récursive du prédicat `contains_three_non_collinear_points`.

```
Fixpoint collinear_with_all a l :=
match l with
| nil => true
| (b,c)::reste => if collinear a b c then collinear_with_all a reste else false
end.
```

Table IV.17 – Définition récursive du prédicat `collinear_with_all`.

```
Fixpoint all_pairs (l : list Point) : list (Point * Point) :=
match l with
| nil => nil
| a :: reste => (map_monotonic (fun x:Point => (a,x)) reste) ++ all_pairs reste
end.
```

TABLE IV.18 – Définition récursive du prédicat `all_pairs`.

3 Non coplanarité

```
Fixpoint contains_four_non_coplanar_points l := match l with
| nil => false
| a :: r => if coplanar_with_all a (all_triples r)
            then contains_four_non_coplanar_points r else true
end.
```

TABLE IV.19 – Définition récursive du prédicat `contains_four`.

```
Fixpoint coplanar_with_all a l :=
match l with
| nil => true
| (b,c,d) :: r => if coplanar a b c d then coplanar_with_all a r else false
end.
```

TABLE IV.20 – Définition récursive du prédicat `coplanar_with_all`.

```
Fixpoint all_triples (l : list Point) : list (Point * Point * Point) :=
match l with
| nil => nil
| a :: reste =>  flatten _ (List.map (fun y:Point =>
       (map_monotonic (fun x:Point => (a,x,y)) reste)) reste) ++ all_triples reste
end.
```

TABLE IV.21 – Définition récursive du prédicat `all_triples`.

```
Fixpoint flatten A (l : list (list A)) : list A :=
match l with
|nil => nil
|a :: r => app a (flatten A r)
end.
```

TABLE IV.22 – Définition récursive du prédicat `flatten`.

Annexe F : Définition Coq de la colinéarité et de la coplanarité

La fonction `proj1_sig` de la librairie standard permet de récupérer le témoin d'existence de la droite entre les points a et b. La notation `eq_dec_u` désigne la décidabilité entre les points, la notation `eq_dec_l` désigne la décidabilité entre les droites et enfin la notation `eq_dec_p` désigne la décidabilité de l'intersection entre les deux droites passées en paramètre.

```
(* Definition collinear *)
Definition collinear (a b c : Point) : bool :=
if (eq_dec_u a b)
   then true
   else if incid_dec c (proj1_sig (a1_exist a b)) then true else false.
```

Table IV.23 – Définition du prédicat `collinear`.

```
(* Definition coplanar *)
Definition coplanar (a b c d : Point) : bool :=
if (eq_dec_u a b)
   then true
   else if (eq_dec_u c d)
        then true
        else let l1 := (proj1_sig (a1_exist a b)) in
             let l2 := (proj1_sig (a1_exist c d)) in
             if (eq_dec_l l1 l2) then true
             else if (eq_dec_p l1 l2) then true
             else false.
```

Table IV.24 – Définition du prédicat `coplanar`.

Annexe G : Architecture Coq de la bibliothèque *ProjectiveGeometry*

Nous résumons l'architecture globale de notre bibliothèque *ProjectiveGeometry* rassemblant la formalisation de la géométrie projective d'incidence, les géométries finies et les preuves automatiques générées par notre prouveur de configurations géométriques d'incidence dans les figures IV.4 et IV.3. Les blocs de couleur permettent de regrouper les parties du développement visant le même objectif et peuvent être interprétés de la façon suivante :

- **Bloc violet** : Systèmes d'axiomes et tactiques en géométrie synthétique ;
- **Bloc jaune** : Systèmes d'axiomes et tactiques fondés sur la notion de rangs ;
- **Bloc gris** : Preuve de l'implication géométrie synthétique vers les rangs [BMS16, BMS19] ;
- **Bloc rouge** : Preuve de l'implication rangs vers la géométrie synthétique [BMS16, BMS19] ;
- **Bloc vert** : Preuve du théorème de Desargues en utilisant les rangs [MNS09, MNS12] ;
- **Bloc bleu clair** : Développement sur les décidabilités, les hexamys et différents modèles dont Moulton [MNS11] ;
- **Bloc orange** : Développement des géométries finies en géométrie synthétique [BMS18] ;
- **Bloc rose** : Développement des géométries finies avec l'approche matroïdale et preuves automatiques générées par le prouveur [BMS18] ;
- **Bloc bleu foncé** : Développement des espaces finies en géométrie synthétique en utilisant l'extension Coq *ssreflect* [GM10, MT17].

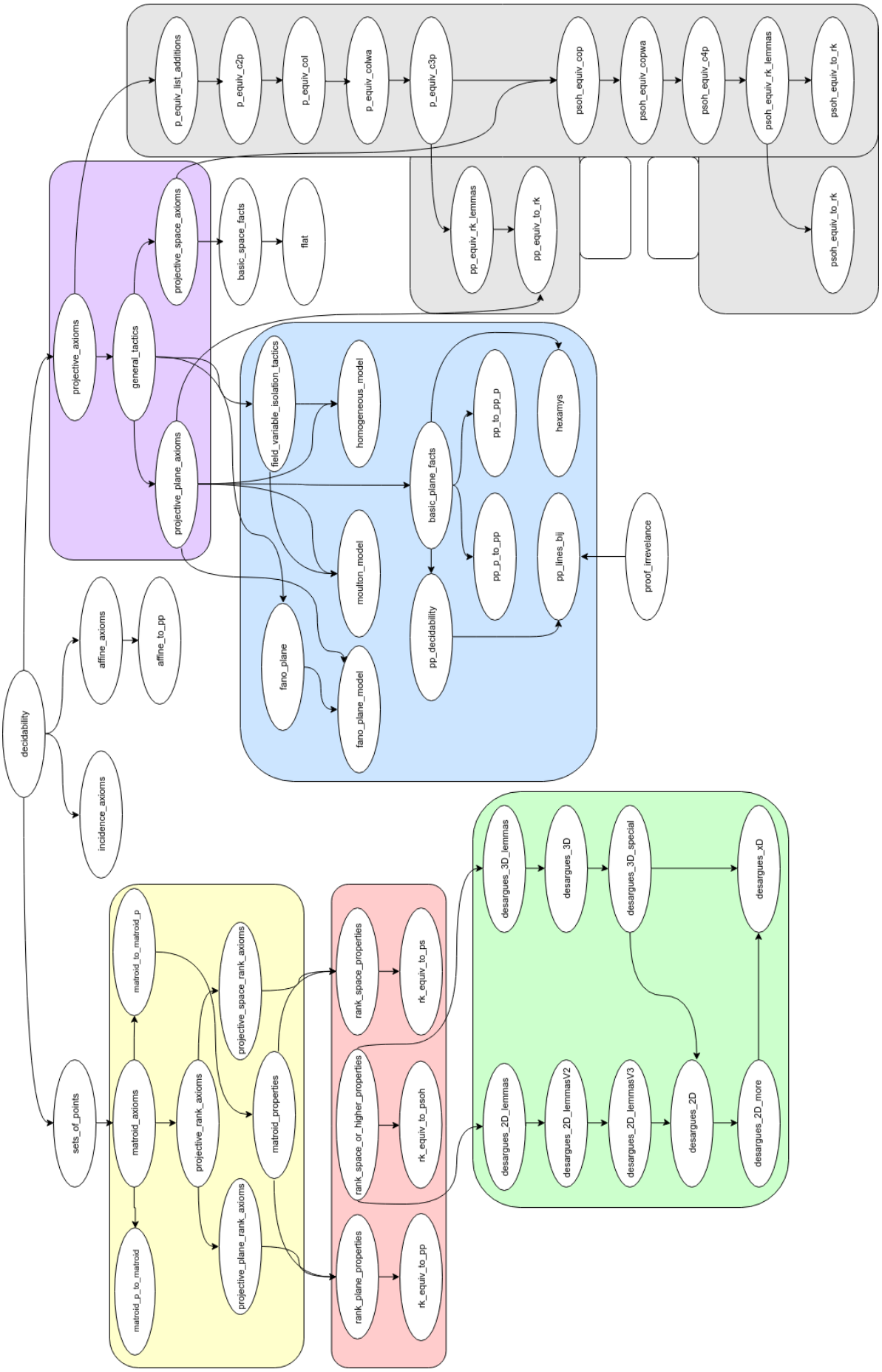


FIGURE IV.3 – Architecture partie I de la bibliothèque *ProjectiveGeometry*.

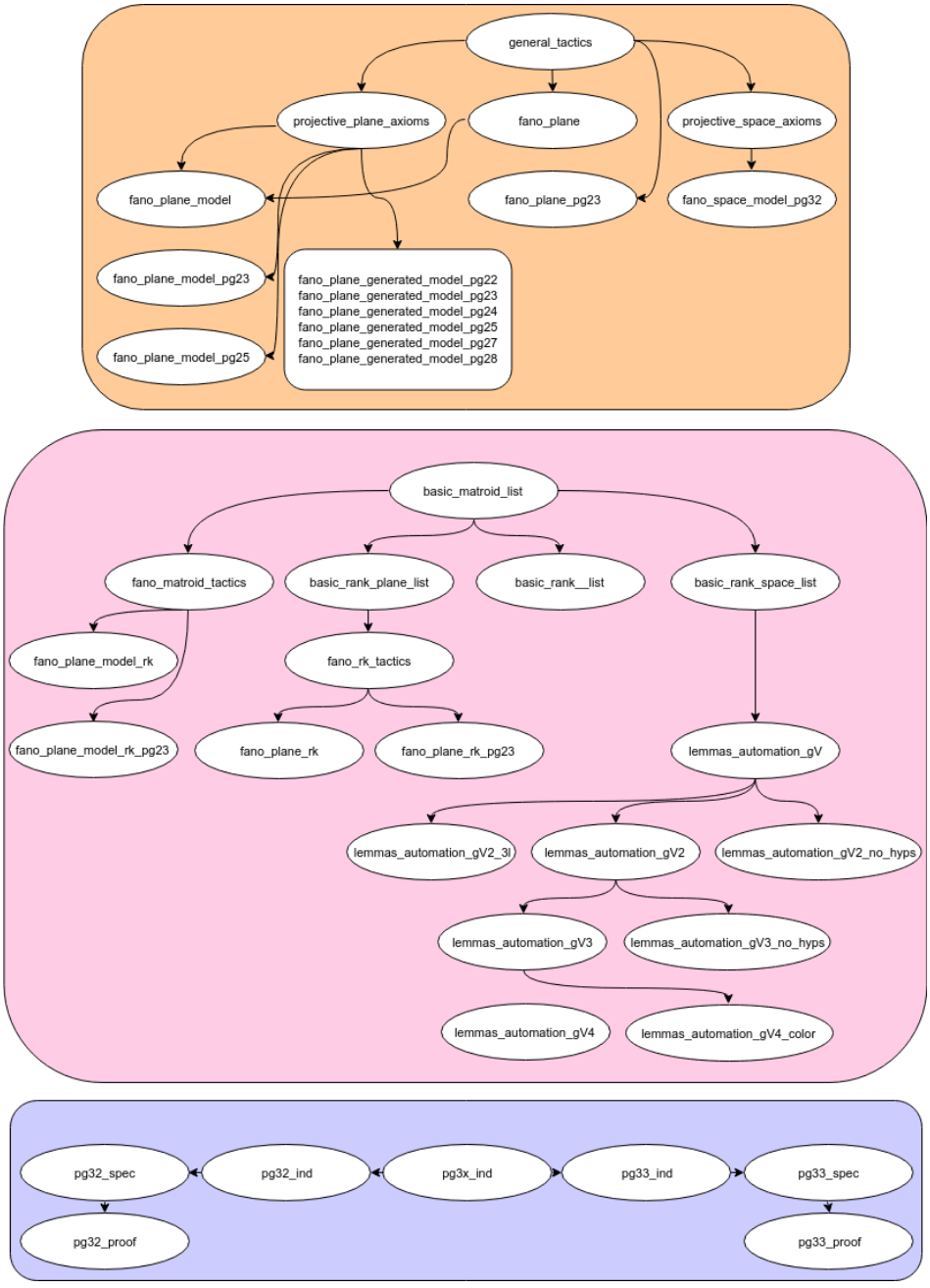


FIGURE IV.4 – Architecture partie II de la bibliothèque *ProjectiveGeometry*.

Liste des tableaux

Table des figures

Liste des Algorithmes

Bibliographie

[ADGR05] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. In *arXiv preprint cs/0509025*, 2005.

[AFG+11a] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In *International Conference on Certified Programs and Proofs*, pages 135–150. Springer, 2011.

[AFG+11b] Mickaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Wener. 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.

[AH89] Kenneth I. Appel and Wolfgang Haken. Every Planar Map is Four Colorable. volume 98. American Mathematical Soc., 1989.

[Bak25] Henry Frederick Baker. Principles of Geometry. volume 1-6. Cambridge University Press, 1925.

[Bat97] Lynn Margaret Batten. Combinatorics of Finite Geometries. Cambridge University Press, 1997.

[BBN16] Gabriel Braun, Pierre Boutry, and Julien Narboux. From Hilbert to Tarski. In *Eleventh International Workshop on Automated Deduction in Geometry*, page 19, 2016.

[BC04] Yves Bertot and Pierre Castéran. Interactive Theorem Proving and Program Development, Coq'Art : The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer Science, 2004.

[BDL16] Arthur Blot, Pierre-Évariste Dagand, and Julia Lawall. From Sets to Bits in Coq. In *FLOPS 2016*, Kochi, Japan, 2016.

[BDN09] Ana Bove, Peter Dybjer, and Ulf Norell. A Brief Overview of Agda–a Functional Language with Dependent Types. In *International Conference on Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.

[BdODF09] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT : an open, trustable and efficient SMT-solver. In *International Conference on Automated Deduction*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.

[Bee13] Michael Beeson. Proof and Computation in Geometry. In Tetsuo Ida and Jacques Fleuriot, editors, *International Workshop on Automated Deduction in Geometry*, volume 7993, pages 1–30, Berlin, Heidelberg, 2013. Springer.

[Ben12] Christoforus Juan Benvenuto. Galois Field in Cryptography. In *University of Washington*, 2012.

[BM15] David Braun and Nicolas Magaud. Des preuves formelles en Coq du théorème de Thalès pour les cercles. In *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015)*, Val d'Ajol, France, 2015.

[BMS16] David Braun, Nicolas Magaud, and Pascal Schreck. An Equivalence Proof Between Rank Theory and Incidence Projective Geometry. In *Automated Deduction in Geometry 2016*, Proceedings of Automated Deduction in Geometry 2016, pages 62–77, 2016.

[BMS18] David Braun, Nicolas Magaud, and Pascal Schreck. Formalizing Some "Small" Finite Models of Projective Geometry in Coq. In *Proceedings of the 13th International Conference on Artificial Intelligence and Symbolic Computation (AISC'2018)*, pages 54–69, Suzhou, China, 2018.

[BMS19] David Braun, Nicolas Magaud, and Pascal Schreck. Two Cryptomorphic Formalizations of Projective Incidence Geometry. In *Annals of Mathematics and Artificial Intelligence*, volume 85, pages 193–212. Springer Verlag, 2019.

[BN12] Gabriel Braun and Julien Narboux. From Tarski to Hilbert. In *International Workshop on Automated Deduction in Geometry*, pages 89–109. Springer, 2012.

[BNS15] Pierre Boutry, Julien Narboux, and Pascal Schreck. A Reflexive Tactic for Automated Generation of Proofs of Incidence to an Affine Variety. 2015.

[BNSB14a] Pierre Boutry, Julien Narboux, Pascal Schreck, and Gabriel Braun. A Short Note About Case Distinctions in Tarski's Geometry. In Francisco Botana and Pedro Quaresma, editors, *Automated Deduction in Geometry 2014*, Proceedings of Automated Deduction in Geometry 2014, pages 1–15. Springer, 2014.

[BNSB14b] Pierre Boutry, Julien Narboux, Pascal Schreck, and Gabriel Braun. Using Small Scale Automation to Improve both Accessibility and Readability of Formal Proofs in Geometry. In Francisco Botana and Pedro Quaresma, editors, *Automated Deduction in Geometry 2014*, Proceedings of Automated Deduction in Geometry 2014, pages 1–19. Springer, 2014.

[Bos38] Raj Chandra Bose. On the Application of the Properties of Galois Fields to the Problem of Construction of Hyper-Graeco-Latin Squares. In *Sankhyā : The Indian Journal of Statistics*, pages 323–338. JSTOR, 1938.

[Bou97] Samuel Boutin. Using Reflection to Build Efficient and Certified Decision Procedures. In *International Symposium on Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 515–529. Springer Berlin Heidelberg, 1997.

[BP11] Jasmin Christian Blanchette and Lawrence C. Paulson. Hammering away : A user's guide to Sledgehammer for Isabelle/HOL. 2011.

[BP15] John Bamberg and Tim Penttila. Completing Segre's Proof of Wedderburn's Little Theorem. In *Bulletin of the London Mathematical Society*, volume 47, pages 483–492. Oxford University Press, 2015.

[BR98] Albrecht Beutelspacher and Ute Rosenbaum. Projective Geometry : from Foundations to Applications. Cambridge University Press, 1998.

[Bru11] Richard H. Bruck. Construction Problems in Finite Projective Spaces. In *Finite geometric structures and their applications*, pages 105–191. Springer, 2011.

[BTT15] Bruno Barras, Carst Tankink, and Enrico Tassi. Asynchronous Processing of Coq Documents : from the Kernel up to the User Interface. In *International Conference on Interactive Theorem Proving*, pages 51–66, Nanjing, China, 2015. Springer.

[Bue95] Francis Buekenhout. Handbook of Incidence Geometry : Buildings and Foundations. Elsevier, 1995.

[BW98] Bruno Buchberger and Franz Winkler. Gröbner bases and applications. volume 17. Cambridge University Press Cambridge, 1998.

[BW11] Simeon Ball and Zsuzsa Weiner. An Introduction to Finite Geometry. In *Preprint*, volume 162, 2011.

[Cal18] Guillermo Calderón. Formalizing Constructive Projective Geometry in Agda. In *Electronic Notes in Theoretical Computer Science*, volume 338, pages 61–77. Elsevier, 2018.

[CGZ94] Shang-Ching Chou, Xiao-Shan Gao, and Jingzhong Zhang. Machine proofs in geometry : Automated production of readable proofs for geometry theorems. volume 6. World Scientific, 1994.

[Chl13] Adam Chlipala. Certified Programming with Dependent Types : a Pragmatic Introduction to the Coq Proof Assistant. MIT Press, 2013.

[CK18] Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq : Automation for Dependent Type Theory. In *Journal of automated reasoning*, volume 61, pages 423–453. Springer, 2018.

[Coq02] Coq development team. The Coq Proof Assistant : Reference Manual : Version 7.2. Number RT-0255, page 290. 2002.

[Cox03] Harold Scott Macdonald Coxeter. Projective Geometry. Springer Science & Business Media, 2003.

[CP88] Thierry Coquand and Christine Paulin. Inductively Defined Types. In *International Conference on Computer Logic*, pages 50–66. Springer, 1988.

[Cré17] Pierre Crégut. Omega : a Solver of Quantifier-free Problems in Presburger Arithmetic. In *The Coq Proof Assistant Reference Manual, Version*, volume 8, 2017.

[CS03] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop : Model Computation-Principles, Algorithms, Applications*, pages 11–27. Citeseer, 2003.

[CS12] Pierre Castéran and Matthieu Sozeau. A Gentle Introduction to Type Classes and Relations in Coq. In *Technical Report hal-00702455, version 1*. 2012.

[DDS00] Christophe Dehlinger, Jean-François Dufourd, and Pascal Schreck. Higher-Order Intuitionistic Formalization and Proofs in Hilbert's Elementary Geometry. In *Revised Papers from the Third International Workshop on Automated Deduction in Geometry*, ADG 2000, pages 306–324. Springer, 2000.

[Del00] David Delahaye. A Tactic Language for the System Coq. In *Proceedings of Logic for Programming and Automated Reasoning (LPAR)*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000.

[Dem12] Peter Dembowski. Finite Geometries : Reprint of the 1968 Edition. Springer Science & Business Media, 2012.

[DGG16] Catherine Dubois, Alain Giorgetti, and Richard Genestier. Tests and Proofs for Enumerative Combinatorics. In *TAP 2016 : International Conference on Tests and Proofs*, pages 57–75, Vienna, Austria, 2016. Springer.

[DM01] David Delahaye and Micaela Mayero. Field : une Procédure de Décision pour les Nombres Réels en Coq. In *JFLA : Journées Francophones des Langages Applicatifs*, pages 1–16, Pontarlier, France, 2001.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3 : An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In *Proceedings of CADE 2015*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[Dup08] Jean Duprat. Une Axiomatique de la Géométrie Plane en Coq. In *Dix-neuvièmes Journées Francophones des Langages Applicatifs (JFLA 2008)*, pages 123–136, Etretat, France, 2008.

[EDH02] Euclid, Dana Densmore, and Thomas Little Heath. Euclid's elements : All thirteen books complete in one volume. Green Lion Press, 2002.

[EMT+17] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq : A plug-in for Integrating SMT Solvers into Coq. In *Computer Aided Verification - 29th International Conference*, Heidelberg, Germany, 2017.

[EP11] Christian Eder and John Edward Perry. Signature-based Algorithms to Compute Gröbner Bases. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation*, pages 99–106. ACM, 2011.

[FMM+06] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Tiu. Expressiveness+ automation+ soundness : Towards combining SMT solvers and Interactive Proof Assistants. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–181. Springer, 2006.

[Fon18] Pascal Fontaine. *Satisfiability Modulo Theories : State-of-the-art, Contributions, Project.* PhD thesis, Université de lorraine, 2018.

[FT10] Laurent Fuchs and Laurent Théry. A Formalization of Grassmann-Cayley Algebra in COQ and Its Application to Theorem Proving in Projective Geometry. In *Automated Deduction in Geometry, ADG 2010*, volume 6877 of *Lecture Notes in Computer Science*, pages 51–67, Munich, Germany, 2010.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy and Christine Paulin and David Pichardie, editor, *ITP 2013, 4th Conference on Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179, Rennes, France, 2013. Springer.

[Gal08] Évariste Galois. Manuscrits de Évariste Galois. Gauthier-Villars, 1908.

[GHL60] Herbert Gelernter, James R. Hansen, and Donald W. Loveland. Empirical Explorations of the Geometry Theorem Machine. In *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*, pages 143–149. ACM, 1960.

[GL02] Benjamin Grégoire and Xavier Leroy. A Compiled Implementation of Strong Reduction. In *ACM SIGPLAN Notices*, volume 37, pages 235–246. ACM, 2002.

[GM10] Georges Gonthier and Assia Mahboubi. An Introduction to Small Scale Reflection in Coq. In *Journal of formalized reasoning*, volume 3, pages 95–152, 2010.

[GM12] Gary Gordon and Jennifer McNulty. Matroids : a Geometric Introduction. Cambridge University Press, 2012.

[GNS11] Jean-David Génevaux, Julien Narboux, and Pascal Schreck. Formalization of Wu's Simple Method in Coq. In *CPP 2011 First International Conference on Certified Programs and Proofs*, volume 7086 of *Lecture Notes in Computer Science*, pages 71–86, Kenting, Taiwan, 2011. Springer.

[Gon05] Georges Gonthier. A Computer-checked Proof of the Four Colour Theorem. 2005.

[Gon07] Georges Gonthier. The Four Colour Theorem : Engineering of a Formal Proof. In *Asian Symposium on Computer Mathematics*, pages 333–333. Springer, 2007.

[Gui04] Frédérique Guilhot. Formalisation en Coq et Visualisation d'un Cours de Géometrie pour le Lycée. In *Quinzièmes Journées Francophones des Langages Applicatifs (JFLA 2004)*, volume 7, page 15, Sainte-Marie-de-Ré, France, 2004. Revue des Sciences et Technologies de l'Information, Technique et Science Informatiques, Langages applicatifs.

[GZND11] Georges Gonthier, Beta Ziliani, Aleksandar Nanevski, and Derek Dreyer. How to Make Ad Hoc Proof Automation Less Ad Hoc. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 163–175, New York, NY, USA, 2011. ACM.

[HAB+17] Thomas C. Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A Formal Proof of the Kepler Conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

[Hal43] Marshall Hall. Projective planes. In *Transactions of the American Mathematical Society*, volume 54, pages 229–277. JSTOR, 1943.

[Hal98] Thomas C. Hales. The Kepler Conjecture. In *arXiv preprint math.MG/9811078*. Springer, 1998.

[Hal07] Thomas C. Hales. The Jordan Curve Theorem, Formally and Informally. In *The American Mathematical Monthly*, volume 114, pages 882–894. Taylor & Francis, 2007.

[Har96] John Harrison. HOL Light : a Tutorial Introduction. In *International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer Berlin Heidelberg, 1996.

[Har09] John Harrison. Formalizing an Analytic Proof of the Prime Number Theorem. In *Journal of Automated Reasoning*, volume 43, pages 243–261. Springer, 2009.

[Hil60] David Hilbert. Foundations of Geometry (Grundlagen der Geometrie). Open Court, La Salle, Illinois, 1960. Second English edition, translated from the tenth German edition by Leo Unger. Original publication date, 1899.

[Hor17] Ákos G Horváth. Gallucci's axiom revisited. In *arXiv preprint arXiv :1712.04800*, 2017.

[Jan11] Predrag Janicic. Automated Reasoning : some Successes and New Challenges. In *Central European Conference on Information and Intelligent Systems*, page 13. Faculty of Organization and Informatics Varazdin, 2011.

[JNQ12] Predrag Janičić, Julien Narboux, and Pedro Quaresma. The Area Method : a Recapitulation. In *Journal of Automated Reasoning*, volume 48, pages 489–532. Springer, 2012.

[Jon03] Simon Peyton Jones. Haskell 98 Language and Libraries : the Revised Report. Cambridge University Press, 2003.

[JTNM06] Christophe Jermann, Gilles Trombettoni, Bertrand Neveu, and Pascal Mathis. Decomposition of Geometric Constraint Systems : a Survey. In *International Journal of Computanional Geometry & Application*, volume 16, pages 379–414. World Scientific, 2006.

[Kah95] Gilles Kahn. Constructive Geometry According to Jan von Plato. In *Coq contribution. Coq*, volume 5, page 10. 1995.

[KEH+09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. seL4 : Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[Kod14] Dimitrios Kodokostas. Proving and Generalizing Desargues' Two-Triangle Theorem in 3-Dimensional Projective Space. In *Geometry*, volume 2014. Hindawi, 2014.

[KRB84] Brian W Kernighan, Dennis M Ritchie, and Thierry Buffenoir. Le Langage C. Masson, 1984.

[Kus90] Eugeniusz Kusak. Desargues Theorem in Projective 3-Space. In *J. of Formalized Mathematics*, volume 2. Citeseer, 1990.

[KV13] Laura Kovács and Andrei Voronkov. First-order Theorem Proving and Vampire. In *International Conference on Computer Aided Verification*, pages 1–35. Springer, 2013.

[Lam96] Clement W. H. Lam. The Search for a Finite Projective Plane of Order 10. In *The Organic Mathematics Project Proceedings*, volume 20, 1996.

[Ler06] Xavier Leroy. Formal Certification of a Compiler Back-end or : Programming a Compiler with a Proof Assistant. In *ACM SIGPLAN Notices*, volume 41, pages 42–54. ACM, 2006.

[Les11] Stéphane Lescuyer. First-Class Containers in Coq. In *Stud. Inform. Univ.*, volume 9, pages 87–127, 2011.

[LKT91] Clement W. H. Lam, Galina Kolesova, and Larry Thiel. A Computer Search for Finite Projective Planes of Order 9. In *Discrete Mathematics*, volume 92, pages 187–195. Elsevier, 1991.

[LTS89] Clement W. H. Lam, Larry Thiel, and Stanley Swiercz. The Non-Existence of Finite Projective Planes of Order 10. In *Canadian Journal of Mathematics*, volume 41, pages 1117–1123. Cambridge University Press, 1989.

[LW93] Xavier Leroy and Pierre Weis. *Manuel de référence du langage Caml*. InterEditions, 1993.

[LW03] Hongbo Li and Yihong Wu. Automated Short Proof Generation for Projective Geometric Theorems with Cayley and Bracket Algebras : I. Incidence Geometry. In *Journal of Symbolic Computation*, volume 36, pages 717–762. Elsevier, 2003.

[Mao11] Linfan Mao. Combinatorial Geometry with Applications to Field Theory, graduate textbook in mathematics. Infinite Study, 2011.

[MBG06] Sean McLaughlin, Clark Barrett, and Yeting Ge. Cooperating theorem provers : A case study combining HOL-Light and CVC Lite. In *Electronic Notes in Theoretical Computer Science*, volume 144, pages 43–51. Elsevier, 2006.

[MF03] Laura Meikle and Jacques Fleuriot. Formalizing Hilbert's Grundlagen in Isabelle/Isar. In *Theorem proving in higher logics*, volume 2758, pages 319–334. Springer Berlin Heidelberg, 2003.

[MF06] Laura Meikle and Jacques Fleuriot. Mechanical Theorem Proving in Computational Geometry. In Hoon Hong and Dongming Wang, editors, *International Workshop on Automated Deduction in Geometry*, pages 1–18. Springer Berlin Heidelberg, 2006.

[MFLS06] Dominique Michelucci, Sebti Foufou, Loïc Lamarque, and Pascal Schreck. Geometric Constraints Solving :some Tracks. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, pages 185–196, 2006.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[MNKJ16] Vesna Marinkovic, Mladen Nikolic, Zoltán Kovács, and Predrag Janicic. Portfolio Methods in Theorem Proving for Elementary Geometry. In *Automated Deduction in Geometry 2016*, Proceedings of Automated Deduction in Geometry 2016, pages 152–161, 2016.

[MNS09] Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing Desargues Theorem in Coq using Ranks. In *24th Annual ACM Symposium on Applied Computing*, Proceedings of SAC 2009, pages 1110–1115. ACM, 2009.

[MNS11] Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing Projective Plane Geometry in Coq. In *Automated Deduction in Geometry*, volume 6301 of *Lecture Notes in Computer Science*, pages 141–162. Springer, 2011.

[MNS12] Nicolas Magaud, Julien Narboux, and Pascal Schreck. A Case Study in Formalizing Projective Geometry in Coq : Desargues Theorem. In *Computational Geometry : Theory and Applications*, volume 45 of *Special Issue on geometric reasoning*, pages 406–424. Elsevier, 2012.

[Mou02] Forest Ray Moulton. A Simple Non-Desarguesian Plane Geometry. In *Transactions of the American Mathematical Society*, volume 3, pages 192–195. JSTOR, 1902.

[MS04] Dominique Michelucci and Pascal Schreck. Detecting Induced Incidences in the Projective Plane. In *isiCAD Workshop. Citeseer*, 2004.

[MS06] Dominique Michelucci and Pascal Schreck. Incidence Constraints : a Combinatorial Approach. In *International J. of Computational Geometry & Application*, volume 16, pages 443–460. World Scientific, 2006.

[MT17] Assia Mahboubi and Enrico Tassi. Mathematical Components. 2017.

[MW05] J. H. Maclagan-Wedderburn. A theorem on finite algebras. In *Transactions of the American Mathematical Society*, volume 6, pages 349–352. JSTOR, 1905.

[Nar04] Julien Narboux. A Decision Procedure for Geometry in Coq. In *Theorem Proving in Higher Order Logics 2004*, volume 3223, pages 225–240. Springer, 2004.

[Nar06a] Julien Narboux. *Formalisation et Automatisation du Raisonnement Géométrique en Coq.* PhD thesis, Université Paris Sud-Paris XI, 2006.

[Nar06b] Julien Narboux. Mechanical Theorem Proving in Tarski's Geometry. In *Automated Deduction in Geometry*, volume 4869, pages 139–156. Springer, 2006.

[NGdV94] Rob P. Nederpelt, Jan Herman Geuvers, and Roel C. de Vrijer. Selected Papers on Automath. volume 133. Elsevier, 1994.

[NJF18] Julien Narboux, Predrag Janičić, and Jacques Fleuriot. Computer-assisted Theorem Proving in Synthetic Geometry. In Meera Sitharam, Audrey St. John, and Jessica Sidman, editors, *Handbook of Geometric Constraint Systems Principles*, Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 2018.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories : from an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL (T). In *Journal of the ACM (JACM)*, volume 53, pages 937–977. ACM, 2006.

[NPW02] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. Isabelle/HOL : a Proof Assistant for Higher-order Logic. volume 2283. Springer Science & Business Media, 2002.

[Oxl06] James G. Oxley. Matroid Theory. volume 3. Oxford University Press, USA, 2006.

[Pot08] Loïc Pottier. Connecting Gröbner Bases Programs with Coq to do Proofs in Algebra, Geometry and Arithmetics. In Sutcliffe, G., Rudnicki, P., Schmidt, R., Konev, B., Schulz, and S., editors, *Knowledge Exchange : Automated Provers and Proof Assistants*, CEUR Workshop Proceedings, page 418, Doha, Qatar, 2008.

[Pri99] Alan R. Prince. Projective Planes of Order 12 and PG (3, 3). In *Discrete mathematics*, volume 208, pages 477–483. Elsevier, 1999.

[PW98] Tim Penttila and B William. Regular packings of PG (3, q). In *European Journal of Combinatorics*, volume 19, pages 713–720. Elsevier, 1998.

[RG95] Jürgen Richter-Gebert. Mechanical Theorem Proving in Projective Geometry. In *Annals of Mathematics and Artificial Intelligence*, volume 13, pages 139–172. Springer, 1995.

[RK70] Thomas Gerald Room and P. B. Kirkpatrick. Mini Quaternion Geometry : an Introduction to the Study of Projective Plans. Cambridge University Press, 1970.

[Ros17] Kenneth H. Rosen. Handbook of Discrete and Combinatorial Mathematics. Chapman and Hall/CRC, 2017.

[Sch19] Pascal Schreck. On the Mechanization of Straightedge and Compass Constructions. In *Journal of Systems Science and Complexity*, volume 32, pages 124–149, 2019.

[Sco08] Phil Scott. Mechanising Hilbert's Foundations of Geometry in Isabelle. In *Master's thesis, University of Edinburgh*. Citeseer, 2008.

[SF12] Phil Scott and Jacques Fleuriot. A Combinator Language for Theorem Discovery. In Johan Jeuring, JohnA. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *International Conference on Intelligent Computer Mathematics*, volume 7362 of *Lecture Notes in Computer Science*, pages 371–385. Springer Berlin Heidelberg, 2012.

[SO08] Matthieu Sozeau and Nicolas Oury. First-class type classes. In *International Conference on Theorem Proving in Higher Order Logics*, volume 5170, pages 278–293. Springer, 2008.

[SST13] Wolfram Schwabhäuser, Wanda Szmielew, and Alfred Tarski. Metamathematische methoden in der geometrie. Springer-Verlag, 2013.

[Sut10] Geoff Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12, Dakar, Senegal, 2010. Springer.

[Tar00] Gaston Tarry. Le Problème des 36 Officiers. Secrétariat de l'Association française pour l'avancement des sciences, 1900.

[Tar59] Alfred Tarski. What is Elementary Geometry ? In *Studies in Logic and the Foundations of Mathematics*, volume 27, pages 16–29. Elsevier, 1959.

[Tar98] Alfred Tarski. A Decision Method for Elementary Algebra and Geometry. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 24–84. Springer, 1998.

[TG15] Tobias Tebbi and Jason Gross. A Profiler for Ltac. In *Coq PL Workshop 2015*. 2015.

[VHDT92] Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A Generic Arc-Consistency Algorithm and its Specializations. In *Artificial intelligence*, volume 57, pages 291–321. Elsevier, 1992.

[Vor03] Andrei Voronkov. Automated Reasoning : Past Story and New Trends. In *IJCAI*, pages 1607–1612, 2003.

[VP95] Jan Von Plato. The Axioms of Constructive Geometry. In *Annals of pure and Applied logic*, volume 76, pages 169–200. North-Holland, 1995.

[VY18] Oswald Veblen and John Wesley Young. Projective geometry. volume 2. Ginn, 1918.

[WCA+14] Dongming Wang, Xiaoyu Chen, Wenya An, Lei Jiang, and Dan Song. OpenGeo : an open geometric knowledge base. In *International Congress on Mathematical Software*, pages 240–245. Springer, 2014.

[Wel10] Dominic J.A. Welsh. Matroid Theory. Courier Corporation, 2010.

[Wie06] Freek Wiedijk. The Seventeen Provers of the World : Foreword by Dana S. Scott. volume 3600 of *Lecture Notes in Computer Science*. Springer, 2006.

[Wu78] Wen-tsün Wu. On the decision problem and the mechanization of theorem-proving in elementary geometry. In *Scientia Sinica*, volume 21, pages 159–172. Science China Press, 1978.

[Wu86] Wen-tsün Wu. Basic Principles of Mechanical Theorem Proving in Elementary Geometries. In *Journal of automated Reasoning*, volume 2, pages 221–252. Springer, 1986.

[Zil14] Ziliani, Beta and Sozeau, Matthieu. Towards a Better-behaved Unification Algorithm for Coq. In *UNIF 2014 Workshop*, pages 74–87, Vienna, Austria, 2014.