Contributions au développement des méthodes formelles de preuves et applications à la géométrie

Nicolas Magaud

ICube - UMR 7357 CNRS Université de Strasbourg

Mémoire d'Habilitation à Diriger des Recherches
Discipline : Informatique

présenté le 06 novembre 2020 par

Nicolas Magaud

Contributions au développement des méthodes formelles de preuves et applications à la géométrie

---

Jury d'habilitation

Rapporteurs

M. Gilles Dowek	*Directeur de recherche, INRIA Paris-Saclay*
M. Jacques Fleuriot	*Professeur, Université d'Edimbourg*
M. Dominique Michelucci	*Professeur, Université de Bourgogne*

Examinateurs

M. Cédric Bastoul	*Professeur, Université de Strasbourg,* *Directeur scientifique chez Huawei France*
Mme Catherine Dubois	*Professeur, ENSIIE Evry*

Garant d'habilitation

M. Pascal Schreck	*Professeur, Université de Strasbourg*

Résumé

L'utilisation des assistants de preuve comme Coq prend de plus en plus d'ampleur. De tels outils permettent de démontrer formellement aussi bien des résultats mathématiques que des propriétés de correction d'algorithmes et de programmes informatiques. La géométrie est un domaine d'application riche et bien adapté pour mettre à l'épreuve ces systèmes d'aide à la preuve. Dans nos travaux, nous nous intéressons à trois domaines spécifiques de la géométrie : la formalisation d'algorithmes géométriques, l'automatisation au moins partielle des démonstrations dans le cadre de la géométrie projective et enfin la modélisation des nombres réels dans un formalisme bien adapté à la géométrie discrète.

Nos premiers travaux en géométrie ont permis de démontrer dans Coq la correction de deux variantes de l'algorithme incrémental de calcul de l'enveloppe convexe d'un ensemble de points du plan en utilisant une approche topologique avec les cartes combinatoires. Ces travaux ont mis en évidence le besoin de disposer d'outils pour automatiser au moins partiellement les démonstrations en géométrie. Nous avons alors étudié cette question dans le cadre simple de la géométrie projective en utilisant une approche combinatoire et la notion de rang d'un ensemble de points. L'outil proposé permet de démontrer automatiquement de nombreux théorèmes emblématiques de la géométrie projective et de produire une trace sous la forme d'un script de preuve vérifiable par Coq. Enfin, faire des preuves en géométrie, qu'elles soient automatisées ou non, ne peut se faire sans disposer d'un modèle informatique des nombres réels. Nous avons formalisé dans Coq la droite d'Harthong-Reeb qui est un modèle du continu ainsi que son implantation informatique avec les entiers de Laugwitz-Schmieden. Ce modèle est bien adapté au contexte de la géométrie discrète et permet notamment de représenter facilement des objets continus de manière discrète en utilisant un schéma d'arithmétisation.

Nos travaux ont un double objectif. Il s'agit d'une part de confronter notre expertise en preuves formelles à des problèmes issus de nouveaux domaines et de démontrer que les outils dont nous disposons sont bien adaptés à la réalisation de preuves formelles, notamment en géométrie. D'autre part, de tels travaux permettent de mettre en évidence les points d'amélioration des outils existants et éventuellement d'en accroître la fiabilité, l'efficacité et la facilité d'utilisation.

Remerciements

Je tiens tout d'abord à remercier Gilles Dowek, Jacques Fleuriot et Dominique Michelucci d'avoir accepté d'être les rapporteurs de cette habilitation à diriger les recherches. Je remercie également Cédric Bastoul et Catherine Dubois d'avoir accepté d'être examinateurs de ce travail. Enfin, je remercie Pascal Schreck d'avoir accepté le rôle de garant de cette habilitation.

Yves Bertot a encadré mon travail de thèse de doctorat il y a maintenant une quinzaine d'années. Alors que le projet de rédiger ce document était dans l'air depuis un certain temps, il a su, en quelques mots, m'encourager et me permettre de démarrer efficacement la rédaction. Je le remercie infiniment pour cela. Merci également à Laurent Fuchs, Alain Ketterlin et Pascal Mathis de m'avoir toujours encouragé dans cette entreprise.

Depuis mon recrutement comme maître de conférences à l'Université Louis Pasteur, devenue aujourd'hui l'Université de Strasbourg, j'ai eu la chance de co-encadrer deux thèses de doctorat : celles de Christophe Brun et de David Braun. Je les remercie sincèrement pour le travail réalisé ensemble ainsi que toutes les discussions scientifiques ou non que nous avons eu pendant leurs thèses. Je remercie également leurs directeurs de thèse officiels, Jean-François Dufourd pour le premier et Pascal Schreck pour le deuxième, qui m'ont permis de m'initier et de m'aguerrir à la délicate et passionnante mission d'encadrant de thèse.

Ces quinze années de recherche ont également données lieu à des collaborations stimulantes et fructueuses avec de nombreux chercheurs, parmi eux : Eric Andrès, Agathe Chollet , Marie-Andrée Da Col, Jean-François Dufourd, Laurent Fuchs, Alain Giorgetti, Gaëlle Largeteau-Skapin, Loïc Mazo, Julien Narboux, Pascal Schreck, Guy Wallet. C'est un plaisir de travailler avec vous ! Je remercie également tous ceux qui suivent, de manière plus ou moins lontaine, mon travail depuis des années.

Je remercie le laboratoire ICube, l'équipe de recherche IGG, l'EPI INRIA Camus pour les excellentes conditions de travail qu'ils proposent. Je remercie également le CNRS pour m'avoir permis, via un accueil en délégation dans mon propre laboratoire, de consacrer une année pleine et entière à la recherche.

Enfin, tout cela n'aurait pas été possible sans la patience et la compréhension de ma femme et de nos deux enfants. Merci !

Table des matières

Chapitre 1

Introduction

Les méthodes formelles de spécification et de preuve ont pour objectif de renforcer la fiabilité des logiciels développés afin de garantir leur exécution correcte vis-à-vis de leur spécification (qui dénote leur comportement attendu). Améliorer un programme ne se résume pas à le rendre plus performant, une exigence majeure est qu'il respecte sa spécification dans tous les cas possibles. Pour cela, plusieurs approches co-existent. Le test systématique et rigoureux est une solution qui permet de débugger efficacement, mais cela présente l'inconvénient d'être coûteux, souvent difficile à mettre en œuvre et surtout incomplet (à de très rares exceptions près). L'approche formelle à l'aide d'un assistant de preuve comme Coq est plus générale bien que plus difficile à mettre en oeuvre du fait du niveau d'expertise requise pour l'utilisateur. Ce dernier fait le travail d'inventivité et la machine sert uniquement à vérifier que le raisonnement mené est bien valide. Dans les cas les plus critiques (lorsque la vie humaine est en jeu - informatique embarquée dans une voiture ou un avion par exemple - ou bien lorsque les coûts financiers sont excessifs et que le premier essai, au sens littéral du terme, doit être le bon - Ariane 5 ou une mission spatiale automatique -), il est indispensable de démontrer avec toute la rigueur possible qu'un programme vérifie bien certaines propriétés.

La généralisation de l'utilisation de ces techniques passe par une prise en main facilitée pour les ingénieurs. Cela nécessite le développement de nouvelles preuves formelles afin de pousser les systèmes existants vers leurs limites, de proposer des améliorations (ajout de bibliothèques, nouvelles procédures de décision, heuristiques) et également d'augmenter l'expérience acquise et ainsi proposer un cadre méthodologique pour démocratiser cette approche.

Depuis le début des années 2000, de nombreux développements formels de grande envergure ont été réalisés avec l'assistant de preuves Coq ou d'autres systèmes similaires, aussi bien en mathématiques qu'en informatique. En voici quelques exemples, parmi les plus significatifs dans chacun de ces deux domaines.

Du côté des mathématiques, Georges Gonthier et Benjamin Werner ont démontré formellement le théorème des quatre couleurs [Gon08]. Un effort collectif important a également été mené pour décrire les outils mathématiques nécessaires pour spécifier et démontrer formellement dans Coq le théorème de Feit-Thompson (ou théorème de l'ordre impair) [GAA$^+$13], dont la preuve initiale était longue de plus de deux cents pages. Plus proche de nous, nos collègues strasbourgeois ont formalisé les fondements de la géométrie au sein d'une bibliothèque formelle Coq nommée GeoCoq [BBB$^+$18], en s'appuyant sur différentes axiomatiques notamment celle de Tarksi, celle d'Hilbert et celle d'Euclide. Dans le cadre des systèmes Isabelle/HOL et HOL light, la conjecture de Képler, dont une preuve mathématique avait été proposée par Thomas Hales quelques années auparavant, a été formellement démontrée [HAB$^+$15]. La production de cette preuve formelle a permis de valider définitivement la démonstration mathématique proposée par Thomas Hales et de lever les dernières réserves de la communauté scientifique.

Du côté de l'informatique, Xavier Leroy a implanté dans Coq un compilateur, *CompCert*, pour le langage C et il en a prouvé la correction [Ler09a, Ler09b]. Plus récemment, un analyseur statique utilisant l'interprétation abstraite et couvrant la majeure partie du C99 a été intégré à Compcert [JLB+15] afin de garantir l'absence d'erreurs à l'exécution pour les programmes vérifiés. Dans le domaine des systèmes d'exploitation, une preuve formelle du micro-noyau seL4 a été réalisée en Isabelle/HOL [KAE+10]. Ce développement formel a permis de détecter et d'éliminer plusieurs bugs présents dans les premières versions de ce micro-noyau.

Dans ce contexte global, mon principal axe de recherche depuis mon recrutement comme maître de conférences à Strasbourg est de travailler sur la formalisation, dans l'assistant de preuve Coq, de notions et algorithmes issus de la géométrie, de prouver formellement certaines de leurs propriétés et de proposer des outils pour automatiser au moins partiellement la construction de certaines démonstrations.

1.1 Contexte professionnel et encadrement doctoral

Après des études d'informatique dans le cadre du magistère Informatique et Modélisation, dispensé à l'École Normale Supérieure de Lyon, j'ai effectué ma thèse de doctorat sous la direction d'Yves Bertot à l'INRIA Sophia-Antipolis (2000-2003). J'ai soutenu ma thèse intitulée *Changements de représentation des données dans le calcul des constructions* fin 2003 et je suis parti en séjour post-doctoral à l'*University of New South Wales* à Sydney en Australie pendant 2 ans. J'ai ensuite été recruté comme maître de conférences à l'Université Louis-Pasteur de Strasbourg à la rentrée 2005.

Depuis mon recrutement comme maître de conférences, j'ai eu l'opportunité d'encadrer deux thèses de doctorat sur des thèmes alliant preuves formelles et géométrie.

Mon premier doctorant, Christophe Brun, a réalisé un travail combinant géométrie algorithmique et preuves formelles. Son travail de thèse a été financé par le projet ANR Galapagos : Géométrie, Preuves et Algorithmes, qui s'est déroulé de 2007 à 2011 et regroupait les sites de Poitiers, Strasbourg et INRIA Sophia-Antipolis. Sa thèse, soutenue en 2010 et co-encadrée par Jean-François Dufourd, s'intitule *Preuves formelles pour le calcul d'enveloppes convexes dans le plan avec des hypercartes*.

Mon deuxième doctorant, David Braun, a travaillé sur le développement d'outils de preuves automatiques pour la géométrie projective. Il a tout d'abord montré que deux systèmes d'axiomes distincts pour décrire la géométrie projective (une description synthétique et une description combinatoire) sont équivalents. Ensuite, lors de mon année d'accueil en délégation au CNRS dans mon propre laboratoire (2016-2017), nous avons travaillé sur l'étude de modèles finis de la géométrie projective et sur les premisses d'un prouveur automatique pour la géométrie projective. Sa thèse, soutenue en 2019 et co-encadrée par Pascal Schreck, s'intitule *Approche combinatoire pour l'automatisation en Coq des preuves formelles en géométrie d'incidence projective*.

J'ai également travaillé, au début des années 2010, avec une chercheuse post-doctorante, Agathe Chollet, sur la formalisation des résultats mathématiques obtenus dans sa thèse. Ce travail de modélisation en Coq d'un modèle discret du continu, la droite d'Harthong-Reeb, et de son implantation informatique a été réalisé en collaboration avec Laurent Fuchs de l'Université de Poitiers.

1.2 Démarche scientifique

Tous les travaux présentés dans ce document modélisent dans l'assistant de preuve Coq des objets mathématiques ou informatiques et leurs propriétés. Nous nous intéressons plus spécifique-

ment à des notions et algorithmes issus de la géométrie et nous cherchons à prouver formellement certaines de leurs propriétés.

Nous avons ici un double objectif. D'une part, nous cherchons par nos méthodes formelles de preuve, à vérifier que les résultats déjà établis sont bien corrects, qu'aucune hypothèse ou cas particulier n'a été occulté, renforçant ainsi la confiance dans les systèmes déjà existants. D'autre part, nos travaux peuvent être vus comme des tests de résistance (*stress-tests* en anglais) pour notre système de preuve favori, à savoir Coq. Formaliser des résultats connus en mathématiques et en informatique dans un assistant de preuve comme Coq nécessite tout d'abord de bien poser le problème considéré en le spécifiant correctement. Il faut également maîtriser les outils de preuve mis à disposition pour les utiliser à bon escient. Mais cela n'est pas suffisant. Les outils formels de spécification et de preuve comme Coq étant encore des outils de recherche, il faut souvent proposer de nouvelles approches de structuration des développements formels, développer de nouvelles tactiques pour automatiser certains raisonnements spécifiques au domaine étudié. Ce travail délicat contribue à rendre l'assistant de preuve plus générique et plus facilement utilisable pour d'autres formalisations.

L'obtention d'une nouvelle formalisation a donc deux effets notables : cela confirme qu'un résultat mathématique ou informatique déjà connu est bien correct et cela renforce également les aptitudes d'un outil comme Coq et le rend mieux adapté aux développements formels à venir. Souvent les problèmes doivent être structurés en plusieurs niveaux d'abstraction pour pouvoir être bien traités. Dans la plupart des cas, nous commençons par une description mathématique, facile à modéliser, des objets considérés et nous nous intéressons ensuite à des implantations informatiques de ces objets.

Ce document résume cette démarche telle qu'elle a été menée dans trois domaines de la géométrie.

1.3 Axes de recherche

Les trois domaines de la géométrie auxquels nous nous sommes intéressés sont les suivants : la géométrie algorithmique, la géométrie projective, et le calcul réel exact vu d'une perspective de géométrie discrète.

Modélisation géométrique à base topologique

La modélisation géométrique à base topologique et sa formalisation dans Coq est un thème de recherche actif depuis de nombreuses années à Strasbourg [PD00, DD04b, DD04a]. Pour faire nos premières armes en géométrie algorithmique, nous avons choisi de modéliser en Coq la notion d'enveloppe convexe afin de valider la pertinence de l'approche topologique basée sur les cartes combinatoires. Nous avons utilisé les cartes combinatoires et leur modélisation en Coq pour décrire formellement en Coq deux variantes de l'algorithme incrémental de calcul de l'enveloppe convexe d'un ensemble de points du plan. Nous avons également démontré la correction de ces deux variantes de l'algorithme vis-à-vis de la spécification formelle de l'enveloppe convexe. Ces travaux ont montré la nécessité de disposer d'outils d'automatisation des preuves et de savoir gérer de manière exacte les calculs sur les réels. Comme nous le montrons dans la suite de ce document, nous avons alors exploré ces deux domaines de recherche mis en évidence par ces premiers travaux.

Géométrie projective

Nous avons étudié comment formaliser en Coq la géometrie projective. Nous avons tout d'abord étudié la géometrie projective plane, construit certains de ses modèles finis et démontré

l'indépendance de la propriété de Desargues. Nous avons ensuite modélisé l'espace projectif, en utilisant notamment une approche combinatoire basée sur la notion de rangs. Les rangs permettent de décrire de manière homogène les énoncés en ne les exprimant qu'avec des points. Cette modélisation avec les rangs nous a permis de démontrer de manière originale le théorème de Desargues [MNS12].

Pendant la thèse de David Braun, nous avons prouvé formellement l'équivalence entre l'axiomatique usuelle de l'espace projectif et la description combinatoire basée sur les rangs de ce même espace [BMS19]. Ce résultat connecte les résultats déjà obtenus sur la preuve du théorème de Desargues avec l'axiomatisation usuelle de la géométrie projective. David Braun a ensuite proposé dans sa thèse un algorithme de construction d'une démonstration tirant profit de l'approche combinatoire étudiée jusqu'ici. Il s'agit d'un outil de recherche d'une solution par saturation du contexte à partir de l'expression des hypothèses et de la conclusion sous forme de propriétés sur des rangs d'ensembles de points bien choisis, permettant ainsi de maîtriser l'éventuelle explosion combinatoire. Cet outil, externe au système Coq, peut néanmoins produire une trace permettant de faire valider la démonstration par Coq. Il a été utilisé avec succès pour démontrer plusieurs théorèmes emblématiques de la géométrie projective en 3D.

Calcul réel exact

La droite d'Harthong-Reeb est un modèle original du continu, bien adapté à la géométrie discrète. Ce modèle, qui permet via une approche non-standard de représenter les réels par des suites d'entiers, a été revisité par Agathe Chollet dans sa thèse [Cho10] et appliqué avec succès pour obtenir une représentation discrète d'objets comme des droites ou des cercles.

En collaboration avec Laurent Fuchs (Université de Poitiers) et Agathe Chollet (Université de la Rochelle), nous avons formalisé en Coq la droite d'Harthong-Reeb. Cette représentation discrète du continu est basée sur l'analyse non standard et notamment la notion d'infiniment petits. Nous avons repris les résultats proposés dans la thèse d'Agathe Chollet et avons vérifié leur correction [MCF14]. Cela a permis par ailleurs de simplifier la preuve mathématique, et de la rendre plus accessible, notamment en mettant en évidence les abus de notation et en s'assurant qu'ils étaient corrects.

Une fois que nous avons établi le résultat mathématique que la droite d'Harthong-Reeb est bien un modèle du continu, nous avons cherché à disposer d'une implantation concrète en informatique. Pour cela, nous représentons les éléments de la droite d'Harthong-Reeb par des suites d'entiers (appelés entiers de Laugwitz-Schmieden). Nous poursuivons les travaux autour de la formalisation de la droite d'Harthong-Reeb et notamment de son modèle constructif, basé sur les entiers de Laugwitz-Schmieden. Nous cherchons à relier cette modélisation du continu avec d'autres représentations plus traditionnelles des réels disponibles dans Coq. Ce travail est un préalable à la description en Coq des Ω-AQAs, qui sont une approche discrète pour représenter des applications affines [ACF$^+$14].

Ces trois axes de recherche sont complétés par d'autres travaux plus modestes, notamment un développement de démonstrations dans différents formalismes du théorème de Thalès pour les cercles [BM15] ainsi qu'une étude sur les transformations polyhédriques [PNVM12] destinée à vérifier que ces transformations ne font pas apparaître ou disparaître de dépassements de capacité.

1.4 Manuscrit

Dans ce document, j'utilise pour décrire la démarche suivie le *nous* narratif. Ce *nous* fait référence aux auteurs des travaux considérés, y compris moi. Dans de nombreuses situations, ce *nous* a aussi pour objectif d'associer le lecteur aux raisonnements présentés et ainsi de lui faire prendre part à notre démarche scientifique.

Le manuscrit est divisé en trois chapitres principaux, traitant chacun d'un des domaines d'application étudiés. Le chapitre 2 décrit l'etude initiale que nous avons menée dans le domaine de la modélisation géométrique à base topologique : il s'agit de modéliser avec des cartes combinatoires et d'implanter en Coq deux programmes fonctionnels de calcul de l'enveloppe convexe d'un ensemble de points du plan, puis d'en établir formellement la correction. Le chapitre 3 présente les travaux sur l'automatisation des preuves effectués dans le contexte de la géométrie projective. Nous décrivons notamment deux approches complémentaires pour modéliser formellement la géométrie projective ainsi qu'un prouveur automatique basé sur une approche combinatoire. Le chapitre 4 est consacré à la formalisation en Coq d'un modèle du calcul réel exact adapté pour la géométrie, la droite d'Harthong-Reeb. Ce modèle se base sur l'arithmétique non-standard et a l'originalité de n'utiliser que des entiers. Enfin, le chapitre 5 synthétise les travaux présentés dans ce document et ouvre des perspectives prometteuses pour les prochaines années.

Chapitre 2

Modélisation géométrique à base topologique

L'algorithmique géométrique [BY98, dBCvKO08] reste un domaine difficile de l'informatique avec de multiples représentations des données, des algorithmes complexes avec de nombreux cas particuliers et des risques d'échecs dus aux approximations numériques. Un projet ambitieux comme CGAL [dt04] unifie et sécurise, au moins partiellement, les structures de données et propose une bibliothèque de référence grâce à l'établissement de normes de programmation strictes, mais qui restent informelles. Nous proposons de compléter cet effort par l'utilisation conjointe de méthodes formelles de spécification pour décrire les programmes de manière précise, exhaustive et non-ambiguë, et de méthodes formelles de preuve pour s'assurer de manière convaincante que ces programmes ne comportent pas d'erreurs. Nous utilisons pour cela l'assistant de preuve Coq [BC04, Coq19], qui implante le calcul des constructions inductives et propose des techniques de spécification et de preuve formelle éprouvées. Le système Coq permet de construire aisément des programmes (fonctionnels) et de prouver des propriétés de ces programmes que ce soit par induction structurelle ou par induction bien fondée. De plus, il propose un langage de tactiques complet facilitant la construction des démonstrations.

Nous nous intéressons plus spécifiquement au problème de calcul de l'enveloppe convexe d'un ensemble de points du plan. L'enveloppe convexe d'un ensemble E de points du plan est un sous-ensemble C des points de E qui forme un polygone convexe tel que tous les points de E se trouve à l'intérieur du polygone formé par C comme illustré dans la figure 2.1.

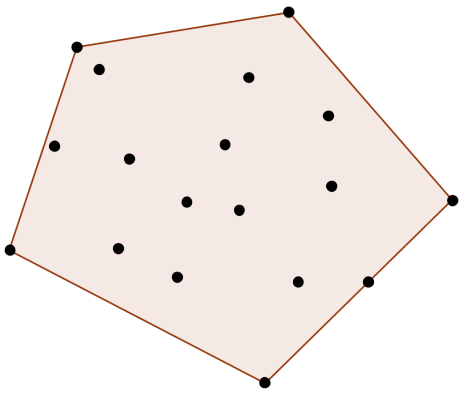


FIGURE 2.1 – Un exemple d'ensemble de points et son enveloppe convexe

Dans nos travaux, nous adoptons une approche basée sur la modélisation géométrique à base

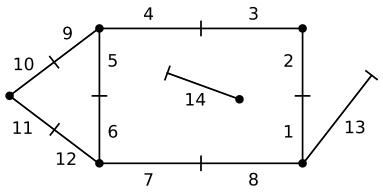
topologique, dont de nombreux aspects ont été décrits formellement avec succès en Coq [PD00, DD04b, DD04a]. Les cartes combinatoires [Jac70] se décrivent facilement de manière formelle (avec un type inductif) et sont bien adaptées à la description de programmes de géométrie algorithmique récursifs (récursion structurelle dont la terminaison est garantie par construction). Du point de vue numérique, on suppose l'existence d'un prédicat d'orientation toujours correct et la preuve de correction fait alors abstraction des problèmes numériques. On suppose que l'on ne fait que des calculs réels exacts. Nous prouvons dans ce cadre simplifié que l'objet construit par l'algorithme est bien une carte planaire, que cette carte représente un polygone, et que son plongement dans le plan est bien l'enveloppe convexe de l'ensemble des points considérés. Nous étudions deux variantes de l'algorithme d'insertion d'un point dans une enveloppe déjà construite. La première version est purement récursive sur la structure inductive des cartes combinatoires et la récursion ne suit donc pas nécessairement la structure géométrique de l'enveloppe convexe. La seconde approche prend en compte l'aspect géométrique, en identifiant les arêtes en conflit lors de l'ajout d'un point et parcourt l'enveloppe convexe en suivant les arêtes adjacentes afin de déterminer les sommets extrémaux gauche et droit qui formeront avec le point inséré la nouvelle partie de l'enveloppe convexe. Cette description est plus proche des implantations usuelles, mais en prouver la terminaison dans Coq est plus complexe. Elle présente l'avantage que l'algorithme décrit en Coq est récursif terminal et que l'on peut donc facilement en déduire une implantation impérative en C++.

Ces travaux ont été réalisés pendant la thèse de Christophe Brun, financée par l'ANR Galapagos et que j'ai co-encadrée avec Jean-François Dufourd entre 2007 et 2010. Les résultats obtenus sont décrits dans les articles suivants : [BDM12a, BDM12b].

2.1 Représentation informatique des cartes combinatoires

Nous définissons les cartes combinatoires et plus précisement les hypercartes et nous les spécifions en Coq.

Définition 1 (Hypercarte)**.** *(1) Une* hypercarte *(en dimension 2) est une structure algébrique* $M = (D, \alpha_0, \alpha_1)$*, où* D *est un ensemble fini, dont les éléments s'appellent des* brins*, et où* α_0 *et* α_1 *sont des permutations sur* D*.*
(2) Quand α_0 *est une involution sur* D *(i.e.* $\forall x \in D$*,* $\alpha_0(\alpha_0(x)) = x$*), l'objet* M *est appelé une* carte combinatoire orientée*, ou tout simplement une* carte*.*



D	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a0	2	1	4	3	6	5	8	7	10	9	12	11	13	14
a1	8	3	2	9	4	12	6	13	5	11	10	7	1	14

FIGURE 2.2 – Un exemple d'hypercarte (en fait un carte combinatoire orientée). Intuitivement, elle représente une subdivision du plan en trois faces : un rectangle (face interne), un triangle (face interne) qui sont collés ensemble et une face externe avec les deux demi-segments restants (13 et 14).

Dans ce contexte, 0 et 1 décrivent les deux *dimensions* possibles. Les cellules topologiques d'une hypercarte peuvent être définies de manière combinatoire par la notion usuelle d'orbite. Soit f une permutation des brins d'une hypercarte. L'*orbite* de x pour f, noté $\langle f \rangle(x)$, est l'ensemble des brins accessibles depuis x en itérant f.

Les orbites pour α_0, notées 0-*orbites*, sont les *arêtes* de l'hypercarte et celles de α_1, notées 1-*orbites*, sont ses *sommets*. Il est important de noter que dans une carte, chaque arête est composée d'au plus deux brins. Les *faces* sont les orbites de $\phi = \alpha_1^{-1} \circ \alpha_0^{-1}$. Le nombre de brins dans une orbite s'appelle le *degré*. Les *composantes connexes* sont définies comme d'habitude : une hypercarte est considérée comme un 2-graphe équipé avec α_0 et α_1, vues comme deux relations binaires.

Un *plongement* d'une hypercarte est un dessin sur une surface où les brins sont représentés par des demi-segments orientés avec les conventions suivantes : (1) les demi-segments de tous les brins d'un même sommet (resp. arête) ont la même origine (resp. extrémité) ; (2) les demi-segments orientés d'une arête, d'un sommet ou d'une face sont parcourus dans le sens direct quand on suit les permutations α_0, α_1 ou ϕ, respectivement ; (3) les demi-segments (ouverts) n'ont pas d'intersection. Quand une hypercarte peut être plongée dans un plan, elle est dite *planaire*. Dans ce cas, chaque face qui entoure une région bornée (resp. non-bornée) à sa gauche est appelée région *interne* (resp. *externe*). Pour plus de détails sur les plongements et la planarité, le lecteur peut consulter [Duf09]. Par exemple, dans la figure 2.2, une hypercarte (en fait une carte) $M = (D, \alpha_0, \alpha_1)$ est plongée dans le plan avec des demi-arêtes droites. Elle contient 14 brins, 8 arêtes (décrites par des *petits traits*), 6 sommets (décrits par des *petits ronds*), 4 faces et 2 composantes connexes. Par exemple, $\langle\alpha_0\rangle(1) = \{1, 2\}$ est une arête du brin 1 et $\langle\alpha_1\rangle(1) = \{1, 8, 13\}$ un sommet. On a $\phi(1) = 3$, $\phi(3) = 5$, $\phi(5) = 7$ et $\phi(7) = 1$. La face (interne) de 1 est $\langle\phi\rangle(1) = \{1, 3, 5, 7\}$ et la face (externe) de 2 est $\langle\phi\rangle(2) = \{2, 13, 8, 12, 10, 4\}$.

Spécification des hypercartes libres

Dans notre spécification en Coq, les brins, de type `dart`, sont des entiers naturels et les dimensions, de type `dim`, sont nommées `zero` et `one`. Les hypercartes sont d'abord approximées par la notion générale d'*hypercartes libres*, décrite par le type `fmap` muni de 3 constructeurs, `V`, `I` and `L`, correspondants respectivement à l'hypercarte *vide*, à l'*insertion* d'un brin, et à la *couture* de deux brins :

```
Inductive fmap : Set :=
 V : fmap
| I : fmap -> dart -> point -> fmap
| L : fmap -> dim -> dart -> dart -> fmap.
```

Par exemple, un morceau de l'hypercarte M de la figure 2.2, avec seulement les brins $1, 2, 3$ et 8 est décrite en Coq par le terme suivant : `(L (L (L (I (I (I (I V 1 p1) 2 p2) 3 p3) 8 p8) zero 1 2) one 2 3) one 1 8)`.
Quand les brins sont insérés dans une hypercarte libre, on leur associe un plongement `point`, qui est un couple de nombres réels. Cela suffit à plonger les brins sur des demi-segments droits et les arêtes sur des segments, dans le plan. Comme on peut le remarquer sur la figure 2.2, il faut bien sûr s'assurer de la cohérence géométrique du plongement. En effet, les points `p2` et `p3` associés respectivement aux brins 2 et 3 doivent être égaux.

Lors de la définition du type `fmap` dans Coq, un principe d'induction sur les hypercartes est automatiquement créé. Ce principe est utilisé pour construire des fonctions et prouver des propriétés par induction sur les hypercartes.

Spécification des hypercartes

Nous introduisons des préconditions pour les opérations `I` et `L` pour éviter les hypercartes incohérentes. La précondition pour `I` exprime qu'un brin que l'on va insérer dans une hypercarte

libre `m` doit être différent de `nil` (= 0) et de tous les brins déjà insérés dans `m`. La précondition pour `L` exprime que les brins `x` et `y` que l'on s'apprête à coudre à la dimension `k` dans l'hypercarte `m` sont déjà présents, que `x` n'a pas de `k`-successeur et que `y` n'a pas de `k`-prédecesseur pour la relation α_k. L'opération α_k est notée en Coq `cA m k`.

Ces deux préconditions permettent de spécifier un invariant `inv_hmap`, que nous utilisons pour caractériser les hypercartes. Cette approche est plus pragmatique que de décrire les hypercartes par un type (dépendant) `hmap` où les types des constructeurs `I` et `L` auraient comme préconditions les propriétés `prec_I` et `prec_L`. Dans notre représentation, une hypercarte est vue comme une hypercarte libre (de type `fmap`) vérifiant l'invariant `inv_hmap`. Cela permet de simplifier l'écriture des calculs en travaillant directement avec des cartes libres et en séparant donc l'application des opérations de base sur les cartes des preuves de leur légitimité.

```
Definition prec_I (m:fmap)(x:dart) : Prop := x <> nil /\ ~ exd m x.


Definition prec_L (m:fmap)(k:dim)(x y:dart) :  Prop :=
exd m x /\ exd m y /\   ~ succ m k x /\ ~ pred m k y /\ cA m k x <> y.


Fixpoint inv_hmap (m:fmap) : Prop :=
  match m with
    V => True
  | I m0 x p => inv_hmap m0 /\ prec_I m0 x
  | L m0 k x y => inv_hmap m0 /\ prec_L m0 k x y
    end.
```

Le prédicat `exd m x` exprime que le brin `x` existe dans l'hypercarte `m`. Les opérations `cA`, `cA_1`, `cF` et `cF_1` simulent le comportement des fonctions α_k, α_k^{-1}, ϕ et ϕ^{-1}. Par exemple, si l'hypercarte de la figure 2.2 est `m`, alors on a `exd m 4`, `cA m one 4 = 9`, `cA_1 m one 7 = 12`, `cA m zero 13 = 13`, `cA_1 m zero 13 = 13`. De plus, quand le brin à insérer n'appartient pas à l'hypercarte, on a `cA m one 15 = nil` et `cA_1 m one 15 = nil`. Enfin, nous avons `cF m 1 = 3`, `cF_1 m 1 = 7`.

2.2 Traitement des propriétés géométriques

Le calcul de l'enveloppe convexe s'appuie non seulement sur des propriétés topologiques, mais aussi sur des propriétés géométriques des points considérés. Nous choisissons de travailler en géométrie euclidienne en deux dimensions et nous définissons chaque point p par ses coordonnées $(x_p, y_p) \in \mathbb{R}$ dans le plan. Afin de pouvoir calculer l'enveloppe convexe incrémentalement, nous avons besoin de déterminer l'orientation de trois points dans le plan.

Nous suivons l'approche proposée par Knuth dans [Knu92] pour gérer les questions d'orientation dans le plan. Cette approche a été utilisée avec succès pour décrire et prouver formellement, en utilisant des structures de données simples, l'algorithme incrémental et la marche de Jarvis [PB01] ainsi que l'algorithme de Graham [MF06]. Nous commençons par spécifier le prédicat d'orientation et ses propriétés, puis nous l'implantons dans le cas où le plan est représenté par $\mathbb{R}^2$.

2.2.1 Spécification du prédicat d'orientation

Le prédicat `ccw`(p, q, r) exprime si les points p, q, r sont énumérés dans le sens direct ou non [1]. La figure 2.3 décrit les différents cas possibles pour le prédicat *ccw*. L'exemple (a) présente un cas où le triplet (p, q, r) est orienté dans le sens direct. L'exemple (b) présente un cas où les trois points sont alignés. L'exemple (c) présente un cas où le triplet (p, q, r) est orienté dans le sens indirect.

En réalité, dans son axiomatisation, Knuth choisit de ne pas prendre en compte les cas dégénérés. Il suppose que les trois points sont toujours en *position générale*, c'est-à-dire qu'ils sont tous distincts deux à deux et qu'ils ne sont pas tous les trois sur la même droite. Nous faisons la même hypothèse dans notre travail. De ce fait, le cas (b) de la figure 2.3 ne se présente jamais [2].

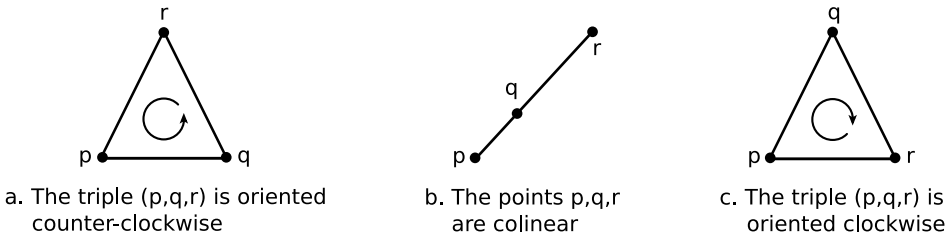


FIGURE 2.3 – Le prédicat d'orientation `ccw`

Le prédicat d'orientation est spécifié de la manière suivante :

Propriété 1 (Prédicat d'orientation géométrique)**.**
P.1 (cyclicité) : $\forall p, q, r, ccw(p, q, r) \Rightarrow ccw(q, r, p)$.
P.2 (symétrie) : $\forall p, q, r, ccw(p, q, r) \Rightarrow \neg ccw(p, r, q)$.
P.3 (non-dégénérescence) : $\forall p, q, r, \neg collinear(p, q, r) \Rightarrow ccw(p, q, r) \vee ccw(p, r, q)$.
P.4 (intériorité) : $\forall p, q, r, t, ccw(t, q, r) \wedge ccw(p, t, r) \wedge ccw(p, q, t) \Rightarrow ccw(p, q, r)$.
P.5 (transitivité) : $\forall p, q, r, s, t, ccw(t, s, p) \wedge ccw(t, s, q) \wedge ccw(t, s, r) \wedge ccw(t, p, q) \wedge ccw(t, q, r) \Rightarrow ccw(t, p, r)$.
P.5 bis (transitivité bis) : $\forall p, q, r, s, t, ccw(s, t, p) \wedge ccw(s, t, q) \wedge ccw(s, t, r) \wedge ccw(t, p, q) \wedge ccw(t, q, r) \Rightarrow ccw(t, p, r)$.

Même si le cas de la colinéarité est explicitement exclu de notre formalisation, une axiomatisation complète exige de disposer d'un prédicat `collinear` qui exprime que trois points sont sur la même droite. Les propriétés 1, 2, et 3 sont faciles à comprendre. Les properties 4, 5, et 5 bis, légèrement plus techniques, sont illustrées dans la figure 2.4. Les lignes en pointillé correspondent aux hypothèses de ces propriétés et les lignes pleines à leurs conclusions.

Toutes ces propriétés sont nécessaires, non seulement pour pouvoir décrire un algorithme de calcul de l'enveloppe convexe correct pour toute configuration de points en position générale, mais aussi pour pouvoir faire une preuve formelle de sa correction. Cette spécification du prédicat d'orientation sera utilisée dans notre développement formel en Coq comme une interface abstraite. Cela laisse la possibilité d'utiliser différentes modélisations de la géométrie (fondations de la géométrie dans le style de GeoCoq [BBB+18], géométrie analytique sur $\mathbb{R}^2$, etc.). Afin de nous assurer de la cohérence de cette spécification, nous prouvons toutes les propriétés énoncées ci-dessus dans le cas du plan euclidien $\mathbb{R}^2$.

1. `ccw` est l'abbréviation de counterclockwise, qui correspond au sens inverse des aiguilles d'une montre aussi appelé sens direct.

2. En réalité, dans les cas industriels, on utilise souvent de petites perturbations du nuage de points pour être dans une situation générale avec une probabilité proche de 1.

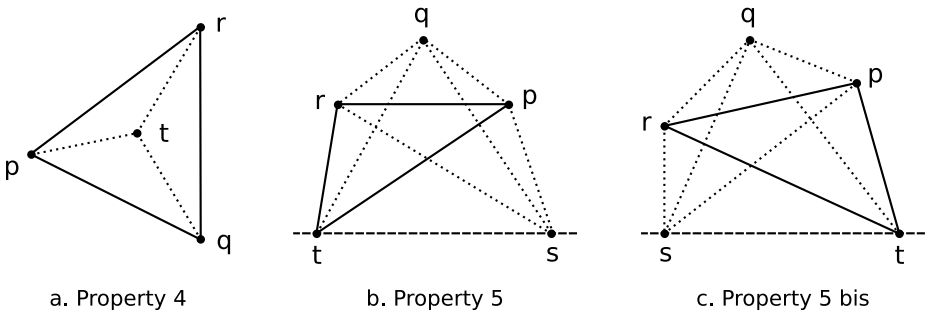


FIGURE 2.4 – Les propriétés 4, 5 et 5 bis du prédicat d'orientation de Knuth `ccw`

2.2.2 Implantation du prédicat d'orientation de Knuth dans $\mathbb{R}^2$

Notre implantation s'appuie sur la définition suivante du prédicat `ccw`.

Définition 2 (Orientation d'un triplet de points)**.**
Soit (p, q, r) un triplet de points dans le plan dont les coordonnées dans $\mathbb{R}$ sont (x_p, y_p), (x_q, y_q) et (x_r, y_r). Le prédicat d'orientation est défini suivant le signe du déterminant de $det(p, q, r)$.

$$det(p, q, r) = \begin{vmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{vmatrix}$$

Si $det(p, q, r) > 0$, alors on a $ccw(p, q, r)$ et cela signifie que p, q, r sont énumérés dans le sens direct. De même, si $det(p, q, r) \leq 0$, alors $ccw(p, q, r)$ n'est pas vérifiée et cela signifie que p, q, r sont énumérés dans le sens indirect ou bien sont collinéaires.

Quand il s'agit d'implanter ce calcul de déterminant en Coq, nous sommes confrontés au fait que les nombres réels n'existent pas en informatique et qu'ils sont seulement approchés par les nombres flottants. Décrire d'une manière informatique les nombres réels afin de pouvoir calculer avec est une question difficile et importante que nous aborderons lors de la formalisation en Coq de la droite d'Harthong-Reeb (voir chapitre 4). Dans le cadre de notre travail sur le calcul de l'enveloppe convexe, nous utilisons simplement la description axiomatique des nombres réels de la bibiliothèque standard de Coq [Coq19]. Les opérations de base (`+`, `-`, $\times$, `/`) sont spécifiées et des propriétés de plus haut niveau sont démontrées à partir de cette spécification abstraite. En Coq, la fonction `det` est simplement définie ainsi :

```
Definition det (p q r : point) : R :=
  (fst p * snd q) - (fst q * snd p) - (fst p * snd r) +
  (fst r * snd p) + (fst q * snd r) - (fst r * snd q).
```

Nous utilisons cette définition pour décrire le prédicat d'orientation `ccw` :

```
Definition ccw (p q r : point) : Prop := (det p q r > 0).
```

Nous montrons alors que toutes les propriétés décrites précédemment sont vérifiées par la fonction `ccw`. Nous montrons également que le prédicat d'orientation est décidable :

```
Lemma ccw_dec : forall (p q r : point), {ccw p q r}+{~ccw p q r}.
```

Cela signifie que nous pouvons l'utiliser dans les expressions conditionnelles de nos algorithmes.

Maintenant que nous disposons d'un cadre formel pour gérer l'orientation des triplets de points, nous ferons abstraction des aspects numériques des calculs et nos algorithmes s'appuieront uniquement sur le prédicat d'orientation `ccw` et ses propriétés, notamment sa décidabilité.

2.3 Définition de l'enveloppe convexe d'un ensemble de points

2.3.1 Définition mathématique

Le calcul de l'enveloppe convexe d'un ensemble de points du plan est un des premiers problèmes qui a été étudié en géométrie algorithmique. De nombreuses définitions, conduisant à différents algorithmes ont été proposées dans la littérature [PS93, BY98, dBCvKO08, Ede87].

Nous choisissons la définition de l'enveloppe convexe qui nous semble la mieux adaptée au modèle topologique des hypercartes, à l'utilisation du prédicat d'orientation de Knuth et à l'algorithme de calcul incrémental que nous étudions. Comme indiqué précédemment, nous supposons que les points sont en *position générale*.

La définition de l'enveloppe convexe s'appuie sur les notions de polygone et de convexité.

Définition 3 (Polygone, polygone simple)**.**
Un polygone *est une figure géométrique plane formée d'une ligne brisée (appelée aussi ligne polygonale) fermée, c'est-à-dire d'une suite cyclique de segments consécutifs.*
Un polygone est dit simple *si deux côtés non consécutifs ne se rencontrent pas et deux côtés consécutifs n'ont en commun que l'un de leurs sommets.*

Définition 4 (convexité, polygone convexe)**.**
Un objet géométrique E est dit convexe *lorsque pour tous points A et B de E, le segment $[A,\ B]$ se trouve intégralement dans E.*
Un polygone est dit convexe *s'il est simple et si son intérieur est convexe. L'intérieur d'un polygone T dont les sommets sont notés t_i, numérotés dans le sens d'un parcours direct $i = 1, \ldots, n$ avec $n+1 = 1$, est convexe si et seulement si pour chaque arête $[t_i t_{i+1}]$ de T et tout point p intérieur à T, distinct de t_i et de t_{i+1}, on a ccw(t_i,t_{i+1},p).*

Soit P un ensemble de points dans le plan.

Définition 5 (Enveloppe convexe)**.**
*L'*enveloppe convexe *de P est le polygone convexe T dont les sommets t_i, numérotés dans le sens d'un parcours direct $i = 1, \ldots, n$ avec $n+1 = 1$, sont des points de P tels que, pour chaque arête $[t_i t_{i+1}]$ de P et pour chaque point p of P distinct de t_i et de t_{i+1}, on a ccw(t_i,t_{i+1},p). En d'autres termes, chaque point p de P distinct de t_i et de t_{i+1} est à gauche de la droite engendrée $\overrightarrow{t_i t_{i+1}}$.*

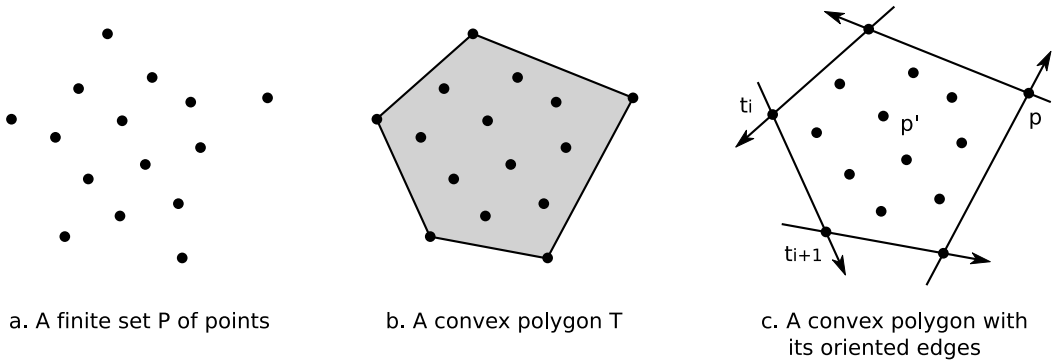


FIGURE 2.5 – Caractérisation de l'enveloppe convexe

La figure 2.5 montre comment caractériser l'enveloppe convexe avec le prédicat d'orientation *ccw*. A gauche (a), nous avons un ensemble fini de points P. Au milieu (b), nous avons un

polygone convexe T à l'intérieur grisé. A droite (c), les flèches représentent les droites orientées $\overrightarrow{t_i t_{i+1}}$ obtenues à partir des arêtes $[t_i t_{i+1}]$ of T. Tous les points $p, p', \ldots$ se trouvent à gauche de ces droites orientées.

2.3.2 Algorithme incrémental

L'enveloppe convexe de l'ensemble de points P du plan est construite pas-à-pas. Chaque étape prend en entrée l'enveloppe convexe courante T (celle construite avec les points déjà traités), un nouveau point p de P et retourne une nouvelle enveloppe convexe T'. Deux cas de figure sont possibles. Soit p se trouve à l'*intérieur* de T et l'algorithme passe au point suivant, soit le point p se trouve à l'*extérieur* de T et l'algorithme doit supprimer les arêtes $[t_i t_{i+1}]$ de T qui sont *visibles* depuis p − celles qui vérifient $\neg\ ccw(t_i, t_{i+1}, p)$ −. Pour construire T', il faut aussi créer deux nouvelles arêtes $[t_l p]$ et $[p t_r]$ qui relie p au *sommet extrémal gauche* t_l et au *sommet extrémal droit* t_r (voir fig. 2.6). Cette description correspond à celles habituellement présentées dans les livres comme [dBCvKO08]. De plus, nous rappelons que, comme nous avons supposé que les points sont en position générale, p ne peut jamais se trouver exactement sur le polygone déjà construit.

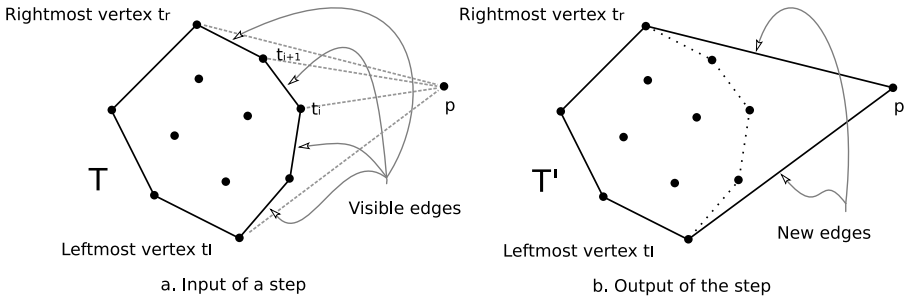


FIGURE 2.6 – Calcul d'une nouvelle enveloppe convexe T' à partir du polygone convexe T et d'un point p

En Coq, l'algorithme incrémental de calcul de l'enveloppe convexe est décomposé en trois fonctions `CH`, `CHI` and `CHID`.

— La première fonction, `CH` initialise le calcul. Pour un point unique, l'algorithme commence avec un ensemble initial de deux points et calcule la première enveloppe convexe, une simple *arête* entre les deux points. La fonction `CH` appelle ensuite la fonction `CHI` avec cette première enveloppe et les points restants à traiter.

— La deuxième fonction, `CHI`, prend chaque élément s de l'ensemble initial P et appelle la fonction d'insertion `CHID` pour éventuellement agrandir l'enveloppe convexe déjà construite.

— La troisième fonction, qui est le cœur dans notre algorithme, `CHID`, effectue l'opération d'insertion d'un nouveau point p dans l'enveloppe convexe déjà construite T. Elle procède par tests sur le prédicat d'orientation *ccw*. En effet, d'après la définition 5, l'intérieur du polygone correspond à l'ensemble des points x du plan tel que $ccw(t_i, t_{i+1}, x)$ pour toute arête $[t_i t_{i+1}]$ de T.

La droite engendrée $\overrightarrow{t_i t_{i+1}}$ divise le plan en deux demi-plans ouverts en fonction de la valeur de $ccw(t_i, t_{i+1}, x)$ pour chaque point x. Il est facile de localiser un point p par rapport à chaque arête $[t_i t_{i+1}]$ du polygone T, en testant $ccw(t_i, t_{i+1}, p)$. En répétant ce test pour $i = 1, \ldots, n$, on en déduit si p est à l'intérieur ou bien à l'extérieur de T.

Si p est à l'intérieur de T, l'enveloppe convexe de $(T \cup p)$ est la même que celle de T. Sinon (p est à l'extérieur de T) l'algorithme supprime les arêtes de T qui sont visibles de p et crée deux nouvelles arêtes $[t_l p]$ et $[p t_r]$ pour relier s avec les sommets extrémaux gauche t_l et droit t_r. Le processus est décrit dans la figure 2.6.

Définition 6 (Arêtes visibles, sommet extrémal gauche, sommet extrémal droit)**.**
Soit T un polygone convexe dans le plan avec au moins deux sommets et soit s un point du plan.
(1) L'arête $[t_i t_{i+1}]$ de T est visible *depuis p si et seulement si $\neg\ ccw(t_i, t_{i+1}, p)$.*
(2) Le sommet t_l de T est le sommet extrémal gauche *par rapport à p si et seulement si $ccw(t_{l-1}, t_l, p)$ et $\neg\ ccw(t_l, t_{l+1}, p)$.*
(3) De manière similaire, le sommet t_r de T est le sommet extrémal droit *par rapport à p si et seulement si $\neg\ ccw(t_{r-1}, t_r, p)$ et $ccw(t_r, t_{r+1}, p)$.*

Nous aurons à montrer l'équivalence de l'existence des sommets t_l et t_r. En effet, quand s est à l'intérieur du polygone, t_l et t_r n'existent pas. Sinon, quand s est à l'extérieur, les deux existent. Il n'y a pas d'autres cas possibles puisque, par hypothèse, le point p ne peut être sur aucune des droites formées par deux points distincts de T. Enfin, nous prouverons également que ces deux sommets t_l et t_r, s'ils existent, sont uniques.

Dans la suite de ce chapitre, nous allons voir comment implanter un algorithme incrémental de calcul de l'enveloppe convexe en utilisant les hypercartes.

2.3.3 Aide à la mise au point : extraction du programme Coq vers OCaml

L'assistant de preuve Coq propose un mécanisme d'extraction qui permet de générer automatiquement des programmes en OCaml ou en Haskell à partir des descriptions fonctionnelles développées en Coq. Les programmes que nous allons implanter en Coq peuvent donc traduits vers Ocaml en utilisant ce mécanisme, puis connectés à une interface de visualisation de l'enveloppe convexe.

Cela donne la possibilité de tester si notre programme se comporte correctement avant même de commencer tout travail de preuve. C'est la première étape avant de pouvoir envisager sereinement de démontrer formellement qu'il respecte bien sa spécification. Cette phase de tests *en conditions réelles* permet également d'évaluer comment nos structures de données et nos algorithmes se comportent en termes d'utilisabilité et d'efficacité.

Cette procédure d'extraction naïve n'est pas optimale et nous verrons plus tard dans ce chapitre comment elle peut être optimisée en transposant de manière explicite notre structure de données (le type inductif des hypercartes et les opérations par filtrage implantées au dessus) vers une représentation plus efficace dans la plate-forme de modélisation géométrique CGoGN [IGG] développée à Strasbourg.

2.4 Programmes de calcul incrémental de l'enveloppe convexe dans le plan

Nous décrivons formellement en Coq deux programmes fonctionnels de calcul de l'enveloppe convexe dans le plan. Les deux fonctionnent de manière incrémentale et sont basés sur les hypercartes. Le premier est un programme récursif structurel, alors que le second utilise une récursion bien-fondée plus proche des implantations usuelles (et du raisonnement géométrique).

2.4.1 Calcul par récursion structurelle sur la structure inductive des cartes

Représentation des données

L'*ensemble initial des points* du plan dont on veut calculer l'enveloppe convexe est représenté par un terme du type `fmap` que l'on contraint à être une carte combinatoire orientée où chaque point est représenté par un brin isolé sans couture et dont le plongement correspond aux coordonnées des points (voir figure 2.7 (a)). L'*enveloppe convexe* finale est un polygone représenté par un terme de type `fmap` que l'on contraint à être une carte combinatoire orientée. Chaque sommet du polygone est représenté par un sommet topologique (deux brins distincts avec le même plongement reliés à la dimension `one`) et chaque arête est représentée par une arête topologique (deux brins distincts avec des plongements différents reliés à la dimension `zero`). Tous les états intermédiaires de calculs sont également représentés par des cartes combinatoires orientées. Dans la suite, nous ne considérerons que les objets de type `fmap` qui sont effectivement des cartes combinatoires orientées.

Comme le calcul incrémental se fonde sur des tests d'orientation, il est nécessaire d'orienter le polygone construit dans le sens direct. Cela est fait en reliant toujours les brins dans la même direction, comme cela est indiqué par de petites flèches sur la figure 2.7 (b). Il est également possible d'associer un brin de référence à la carte afin de pouvoir l'orienter. Cette solution sera utilisée, plus loin dans ce document, dans la deuxième version du programme. Tous les brins représentant des points sont conservés dans le résultat et ceux intérieurs à l'enveloppe convexe restent des brins isolés sans coûture. Si nécessaire, ces brins pourraient être effacés.

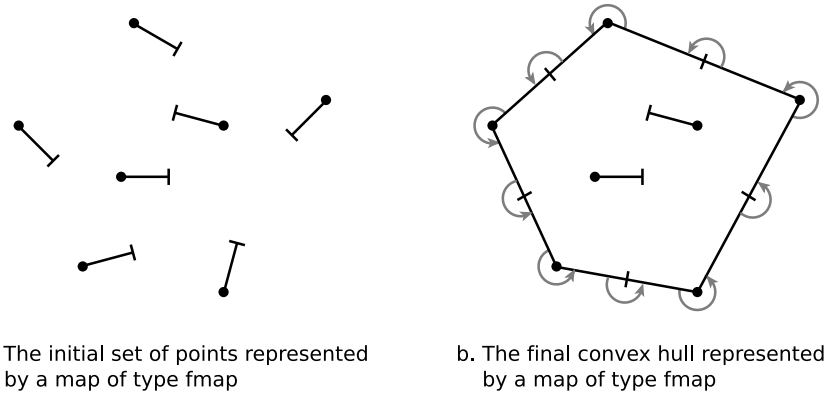


FIGURE 2.7 – L'entrée et la sortie de l'algorithme sous forme d'objets de type `fmap`

Préconditions

La précondition de la fonction principale `CH` contient quatre prédicats :

```
Definition prec_CH (m:fmap) : Prop :=
  inv_hmap m /\ linkless m /\ well_emb m /\ noncollinear m.
```

L'hypercarte `m` doit vérifier l'invariant `inv_hmap` des cartes présenté dans la section 2.1. Il faut aussi qu'aucuns des brins ne soient cousus entre eux (pas de constructeur `L`), ce qu'exprime le prédicat `linkless`. Le prédicat `well_emb` exprime que le plongement géométrique est cohérent, c'est-à-dire que tous les brins de l'entrée ont des plongements différents. Enfin le prédicat `noncollinear` exprime notre hypothèse que trois brins ayant des plongements différents ne peuvent pas être plongés sur trois points alignés :

```
Definition noncollinear (m:fmap) : Prop :=
  forall (d1 d2 d3 : dart),
  let p1 := (fpoint m d1) in let p2 := (fpoint m d2) in
  let p3 := (fpoint m d3) in exd m d1 -> exd m d2 -> exd m d3 ->
  p1 <> p2 -> p1 <> p3 -> p2 <> p3 -> ~ collinear p1 p2 p3.
```

Dans la définition ci-dessus, `fpoint m d` est le point sur lequel le brin `d` est plongé dans la carte `m`.

Classification des brins

Nos choix d'implantation de l'algorithme incrémental nécessitent de distinguer trois catégories de brins (noirs, bleus et rouges) en fonction des coûtures de ces brins. Les *brins noirs* sont des brins isolés sans coûtures. Les *brins bleus* sont ceux qui ont exactement un prédecesseur à la dimension `one` et exactement un successeur à la dimension `zero`. Les *brins rouges* sont ceux qui ont exactement un prédecesseur à la dimension `zero` et exactement un successeur à la dimension `one`. Rappelons enfin que dans notre définition en Coq des hypercartes, un brin ne peut pas avoir plus d'un successeur et d'un prédecesseur pour chaque dimension. Cette classification des brins est présentée dans la figure 2.8. Les brins noirs sont décrits par des traits pleins, les brins bleus par des tirets et les brins rouges par des pointillés.

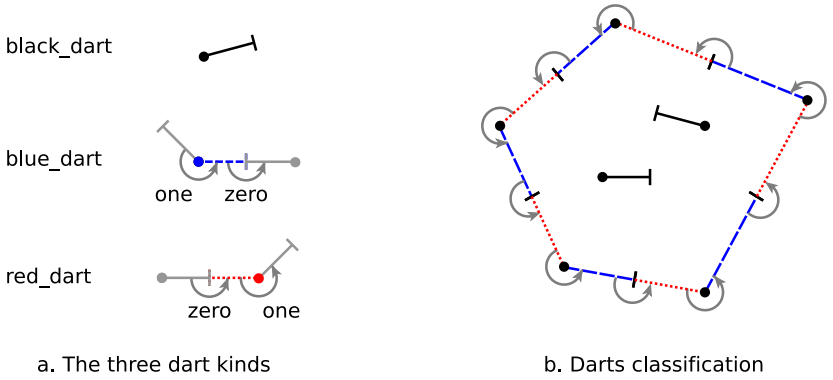


FIGURE 2.8 – Les trois catégories de brins et leur rôle dans la description de l'enveloppe convexe

En Coq, les prédicats `black_dart`, `blue_dart` et `red_dart` expriment qu'un brin `x` est *noir*, *bleu* ou *rouge* dans un carte donnée `m` :

```
Definition black_dart (m:fmap)(d:dart) : Prop :=
  ~ succ m zero d /\ ~ succ m one d /\ ~ pred m zero d /\ ~ pred m one d.

Definition blue_dart (m:fmap)(d:dart) : Prop :=
  succ m zero d /\ ~ succ m one d /\ ~ pred m zero d /\ pred m one d.

Definition red_dart (m:fmap)(d:dart) : Prop :=
  ~ succ m zero d /\ succ m one d /\ pred m zero d /\ ~ pred m one d.
```

On montre que ces trois propriétés sont décidables et que l'on dispose donc des fonctions `black_dart_dec`, `blue_dart_dec`, `red_dart_dec` qui permettent d'écrire des expressions conditionnelles dans le code Coq. Nous montrerons plus tard dans ce chapitre que, dans le cas où l'enveloppe construite forme bien un polygone avec des brins isolés, tous les brins sont soit bleus, soit rouges, soit noirs.

Postconditions

L'algorithme doit produire une carte combinatoire orientée, bien plongée, qui forme un polygone et qui est convexe. Nous présentons ici uniquement la formalisation en Coq de la propriété de convexité énoncée dans la définition 4, qui s'appuie sur le prédicat d'orientation `ccw`. Cette définition ne considère que les points de l'ensemble initial dont on calcule l'enveloppe convexe, comme cela est fait dans la définition mathématique 5 de l'enveloppe convexe :

```
Definition convex (m:fmap) : Prop := forall (x:dart)(y:dart),
  exd m x -> exd m y -> blue_dart m x ->
  let px := (fpoint m x) in let py := (fpoint m y) in
  let x0 := (A m zero x) in let px0 := (fpoint m x0) in
  px <> py -> px0 <> py -> ccw px px0 py.
```

Brins visibles, extrémités gauche et droite

Nous avons mis en évidence dans la section 2.3.2 l'importance du concept de *visibilité* d'une arête depuis un point ainsi que le rôle des sommets extrémaux gauche et droit. Comme nous travaillons uniquement avec des brins, la visibilité d'une arête peut être exprimée sur n'importe lequel de ses brins et les sommets extrémaux sont remplacés par les brins extrémaux correspondants comme le montre la figure 2.9.

Les prédicats `visible` et `invisible` sont définis en s'appuyant sur la classification des brins et le prédicat d'orientation `ccw`. Le prédicat `visible` est défini ainsi :

```
Definition visible (m:fmap)(d:dart)(p:point) : Prop :=
  if (blue_dart_dec m d) then (ccw (fpoint m d) p (fpoint m (A m zero d)))
                         else (ccw (fpoint m (A_1 m zero d)) p (fpoint m d)).
```

Le prédicat `invisible` est exactement la négation du prédicat `visible`. Une fois la décidabilité de ces propriétés établies (`visible_dec` et `invisible_dec`), nous spécifions deux prédicats `left_dart` et `right_dart` qui expriment respectivement qu'un brin est le brin extrémal gauche (resp. droit) par rapport au point `p` :

```
Definition left_dart (m:fmap)(p:point)(d:dart) : Prop :=
  blue_dart m d /\ invisible m (A_1 m one d) p /\ visible m d p.

Definition right_dart (m:fmap)(p:point)(d:dart) : Prop :=
  red_dart m d /\ visible m d p /\ invisible m (A m one d) p.
```

Nous prouvons facilement la décidabilité de ces deux propriétés. Par convention, nous considérons systématiquement que le brin extrémal gauche est un brin bleu et que le brin extrémal droit est un brin rouge. Enfin nous montrons l'équivalence de l'existence des brins extrémaux gauche et droite ainsi que leur unicité, s'ils existent.

Implantation de l'ajout d'un point dans l'enveloppe convexe

Nous décrivons maintenant notre programme d'insertion `CHID` d'un point, représenté par un brin `x` dans une enveloppe convexe déjà construite, représentée par une hypercarte `m` (de type `fmap`). Cette fonction procède par récursion structurelle sur les hypercartes libres, et étudie donc chaque brin et chaque lien séparément. Les brins sont traités dans un ordre arbitraire (suivant la structure du terme de type `fmap`) pendant la phase de construction du polygone (plutôt que

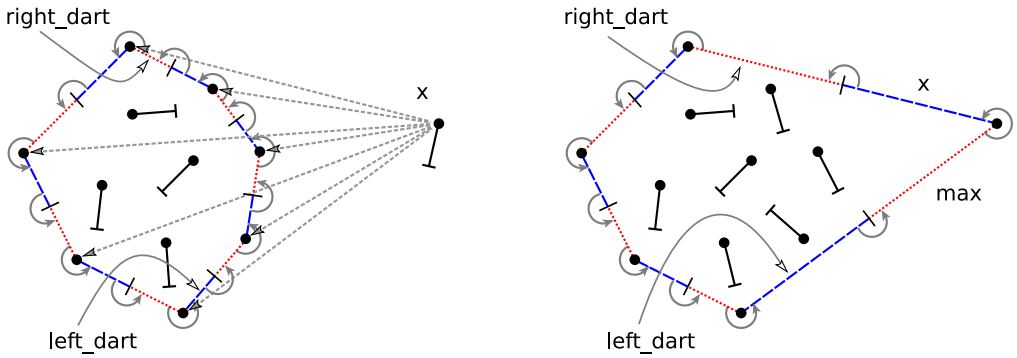


FIGURE 2.9 – Entrée et sortie de la fonction d'insertion d'un brin x dans l'enveloppe convexe déjà construite

d'être traité successivement en parcourant le polygone dans le sens direct). Comme la carte `m` est modifiée à chaque étape, la fonction `CHID` maintient une carte de *référence* `mr`, qui correspond à la carte `m` avant le premier appel à `CHID`. Cette carte de référence est utilisée pour tester des propriétés et n'est jamais modifiée par la fonction. A chaque étape de calcul, m est une sous-carte (*submap*) de la carte de référence mr :

```
Fixpoint submap (m:fmap)(mr:fmap) {struct m} : Prop :=
  match m with
    V => True
  | I m0 x p => submap m0 mr /\ exd mr x /\ (fpoint mr x) = p
  | L m0 k x y => submap m0 mr /\ (A mr k x) = y /\ (A_1 mr k y) = x
  end.
```

Initialement, m est égal à mr et à chaque appel récursif, nous montrons formellement en Coq la propriété `submap` m mr.

Deux cas de figure sont possibles à chaque appel de la fonction `CHID`. Si le nouveau point se trouve à l'intérieur de l'enveloppe convexe déjà calculée, la fonction insère simplement le brin dans la carte `m` sans aucune couture. Si le point se trouve à l'extérieur de l'enveloppe convexe, la fonction `CHID` supprime les arêtes du polygone qui sont visibles depuis le nouveau point et crée deux nouvelles arêtes reliant ce point aux sommets extrémaux gauche et droit.

On peut dire que la fonction `CHID` travaille *constructivement* plutôt que destructivement. En effet, elle reconstruit toujours l'hypercarte du début en (r)ajoutant un à un les brins et les nouveaux liens. Si un lien ne doit pas être remis dans la carte, il est simplement oublié. Une étape de calcul récursif sur (`CHID m mr x p max`) se déroule ainsi (le code se trouve dans la figure 2.10) :

— Si `m` est la carte vide (ligne 04), la fonction `CHID` retourne le brin `x` sans couture.

— Si `m` est de la forme (`I m0 x0 p0`) (ligne 05), la fonction `CHID` détermine la catégorie du brin `x0` dans `mr`.
- Si `x0` est un brin *bleu* dans `mr` (ligne 06), on teste si `x0` appartient à une arête de `mr` qui est invisible depuis le point `p` (ligne 07) en utilisant le prédicat `invisible_dec`. Si l'arête de `x0` est invisible depuis `p` (ligne 08), le brin est simplement conservé. Sinon, la fonction `CHID` teste (ligne 09) si `x0` est le nouveau brin extrémal gauche de `mr` par rapport à `x`. Si `x0` est le brin extrémal gauche (ligne 10), il est conservé dans la carte. De plus, un nouveau brin `max` (plongé sur `p`) est inséré et cousu avec `x` à la dimension `one`. Enfin, `x0` et `max` sont

```
01: Fixpoint CHID (m:fmap)(mr:fmap)(x:dart)(p:point)
02:   (max:dart) {struct m} : fmap :=
03:   match m with
04:     V => I V x p
05:   | I m0 x0 p0 =>
06:     if (blue_dart_dec mr x0) then
07:       if (invisible_dec mr x0 p) then
08:         (I (CHID m0 mr x p max) x0 p0)
09:       else if (left_dart_dec mr p x0) then
10:              (L (L (I (I (CHID m0 mr x p max) x0 p0)
11:              max p) one max x) zero x0 max)
12:            else (I (CHID m0 mr x p max) x0 p0)
13:     else if (red_dart_dec mr x0) then
14:            if (invisible_dec mr x0 p) then
15:              (I (CHID m0 mr x p max) x0 p0)
16:            else if (right_dart_dec mr p x0) then
17:            (L (I (CHID m0 mr x p max) x0 p0) zero x x0)
18:                 else (CHID m0 mr x p max)
19:          else (I (CHID m0 mr x p max) x0 p0)
20:   | L m0 zero x0 y0 =>
21:     if (invisible_dec mr x0 p) then
22:       (L (CHID m0 mr x p max) zero x0 y0)
23:     else (CHID m0 mr x p max)
24:   | L m0 one x0 y0 =>
25:     if (invisible_dec mr x0 p) then
26:       (L (CHID m0 mr x p max) one x0 y0)
27:     else if (invisible_dec mr y0 p) then
28:            (L (CHID m0 mr x p max) one x0 y0)
29:          else (CHID m0 mr x p max)
30:   end.
```

FIGURE 2.10 – CHID function in Coq

cousus à la dimension `zero`. Sinon, `x0` est simplement conservé dans la carte (ligne 12).
- Si `x0` est un brin *rouge* dans `mr`, un calcul similaire est effectué (lignes 13-18).
- Si `x0` est un brin *noir* dans `mr`, il est conservé dans la carte (ligne 19).

— Si `m` est de la forme (`L m0 zero x0 y0`) (ligne 20), la fonction `CHID` teste si l'arête formée par `x0` et `y0` est invisible depuis `x`, qui est plongé sur `p` (ligne 21). Si elle est invisible depuis `p`, le lien à la dimension `zero` entre `x0` et `y0` est conservé (ligne 22). Sinon, il n'est pas ajouté dans la carte résultat (ligne 23).

— On procède de même dans le cas où `m` est de la forme (`L m0 one x0 y0`) (lignes 24-29).

Cette fonction récursive structurelle est facile à implanter en Coq, mais elle est éloignée des implantations habituelles par les géomètres. En effet, elle ne tire pas avantage des propriétés géométriques de l'enveloppe convexe déjà construite. C'est pourquoi nous présentons dans la section suivante une implantation de l'algorithme de calcul de l'enveloppe convexe d'un ensemble de points, qui est cette fois non struturellement récursive, mais qui suit l'intuition géométrique du calcul de l'enveloppe convexe d'un ensemble de points du plan.

2.4.2 Calcul par récursion en suivant la forme de l'enveloppe convexe

La fonction présentée dans cette section cherche les extrémités gauche et droite où l'enveloppe convexe doit être modifiée lorsque le point inséré se trouve à l'extérieur de l'enveloppe déjà construite. Une fois ces extémités détectés, elle coupe la carte au niveau de ces extrémités et relie les extrémités au brin correspondant au nouveau point inséré.

Opérations de haut niveau sur les brins : les fonctions Merge et Split

Nous introduisons ici deux opérations de haut niveau sur les hypercartes, telles qu'elles ont été proposées par Dufourd dans [DB10]. Ces opérations proposent un niveau d'abstraction plus élevée que les constructeurs des cartes et facilite ainsi la description des opérations de calcul de l'enveloppe convexe basées sur les hypercartes. Pour chaque dimension `k` = 0 ou 1, l'opération `Merge` fusionne deux k-orbites. Pour cela, il faut choisir un brin `x` dans la première orbite et un brin `y` dans la seconde, de sorte que le k-successeur de `x` sera `y` dans la nouvelle orbite (Fig. 2.11.a). La précondition de `Merge` exige que les brins `x` et `y` ne soient pas dans la même k-orbite.

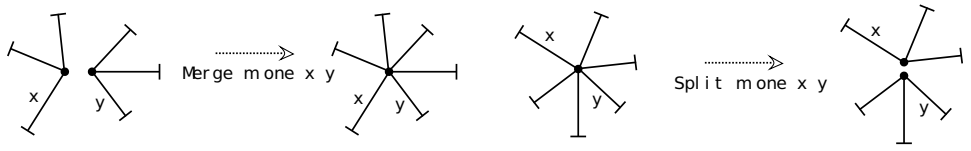


FIGURE 2.11 – (a). Fusion de deux k-orbites (b). Scission d'une orbite

L'opération `Split` sépare une k-orbite en deux morceaux par rapport à deux brins `x` et `y`. La précondition de `Split` exige que les brins `x` et `y` soient différents et appartiennent à la même k-orbite (Fig. 2.11.b).

Il est facile de démontrer que, sous réserve que les préconditions soient bien vérifiées, ces opérations conservent l'invariant des hypercartes `inv_hmap`. D'autres propriétés topologiques et géométriques sont établies au fur et à mesure de la preuve de correction du calcul de l'enveloppe convexe.

Spécification du programme

Comme dans la version précédente, l'enveloppe convexe est la face interne de l'hypercarte courante. Pour l'identifier plus facilement, nous définissons une structure de données `mapdart` qui contient à la fois l'hypercarte correspondant à l'enveloppe convexe et un brin de l'enveloppe convexe permettant de distinguer les faces.

```
Definition mapdart := fmap * dart.
```

Préconditions

L'hypercarte initiale doit vérifier les quatre préconditions déjà présentées à la section précédente :

```
Definition prec_CH (m:fmap) : Prop :=
 inv_hmap m /\ linkless m /\ is_neq_point m /\ is_noalign m.
```

L'hypercarte libre `m` doit évidement vérifier l'invariant `inv_hmap`. Le prédicat `linkless` exprime que les brins ne sont pas cousus entre eux. Le prédicat `is_neq_point` exprime que tous les brins de l'hypercarte initial ont des plongements distincts (cela correspond au prédicat `well_emb` de la description précédente). Enfin, le prédicat `is_noalign` exprime que trois brins avec des plongements distincts ne peuvent pas être plongés sur trois points alignés (cela correspond exactement au prédicat `noncollinear` de la description précédente).

Brins extrémaux gauche et droit

Nous définissons deux fonctions `search_left` et `search_right` qui parcourent récursivement l'ensemble des brins pour chercher respectivement les brins extrémaux gauche et droit dans la face (`m`,`d`) par rapport au point `p`. Ces fonctions sont récursives non-structurelles et sont définies en Coq en utilisant la commande `Function`. Une *mesure strictement décroissante* doit être fournie à Coq pour garantir la terminaison. Le mot-clé `measure` attend deux arguments, une fonction de calcul de la mesure et son argument. Informellement, ces deux fonctions utilisent un entier `i` qui est incrémenté de 1 à chaque appel récursif (en démarrant à 0). Si le degré de la face (c'est-à-dire son nombre de brins), défini en Coq par le prédicat `degreef`, est plus petit ou égal à `i`, alors tous les brins de la face ont été étudiés et il n'y a pas de brin extrémal gauche (resp. droit). Sinon, la fonction teste si le `i`-ème successeur dans la face contenant `d` correspond au brin extrémal gauche (resp. droit).

```
Function search_left (m:fmap)(d:dart)(p:point)(i:nat)
  {measure (fun n:nat => (degreef m d) - n) i} :=
  if (le_lt_dec (degreef m d) i) then nil
  else let di := Iter (cF m) i d in
    if (left_dart_dec m di p) then di
    else search_left m d p (i+1).
```

La fonction d'insertion `CHID` d'un brin dans une enveloppe déjà construite

Cette fonction calcule l'enveloppe convexe d'un polygone convexe `md := (m,d)` et d'un nouveau brin `x`. Elle calcule le brin extrémal gauche `l` et le brin extrémal droit `r` dans `m` par rapport au brin `x` et son plongement `p`. Si `l` est `nil`, alors `r` est aussi `nil` et cela signifie que `p` est à l'intérieur de l'enveloppe convexe déjà construite. Dans ce cas, la fonction insère simplement le brin `x` dans l'hypercarte libre. Sinon, la fonction effectue les six étapes suivantes, illustrées à la figure 2.12.

1. Elle sépare (`split`) le brin extrémal gauche `l` de son 0-successeur `l0 := cA m zero l`.
2. Si le brin `l0` est différent de `r`, elle découût `r` de son 0-prédecesseur `r_0 := cA_1 m zero r`.
3. Elle insère le brin `x` et un *nouveau* brin `max`, tous les deux plongés sur `p`.
4. Elle lie (`merge`) `max` et `x` à la dimension `one`.
5. Elle lie `l` et `max` à la dimension `zero` pour créer une nouvelle arête.
6. Elle lie `x` et `r` à la dimension `zero` pour fermer l'enveloppe convexe.

Cette description formelle est beaucoup plus proche des implantations usuelles de l'enveloppe convexe par les géomètres [dBCvKO08, Ede87] et elle tire avantage des opérations de haut niveau sur les orbites `merge` et `split` pour permettre une meilleure structuration et donc une meilleure lisibilité des opérations effectuées par la fonction `CHID`.

```
Definition CHID (md:mapdart)(x:dart)(p:point)(max:dart):mapdart :=
  let m := fst md in let d := snd md in
  let l := search_left m d p 0 in
  if (eq_dart_dec l nil) then (I m x p, d)
  else let r := search_right m l p 0 in
     let l0 := cA m zero l in let r_0 := cA_1 m zero r in
     let m1 := Split m zero l l0 in                          (*1*)
     let m2 := if (eq_dart_dec l0 r) then m1                 (*2*)
               else Split m1 zero r_0 r in
     let m3 := (I (I m2 x p) max p) in                       (*3*)
     let m4 := Merge m3 one max x in                         (*4*)
     let m5 := Merge m4 zero l max in                        (*5*)
     let m6 := Merge m5 zero x r in (m6, x).                 (*6*)
```

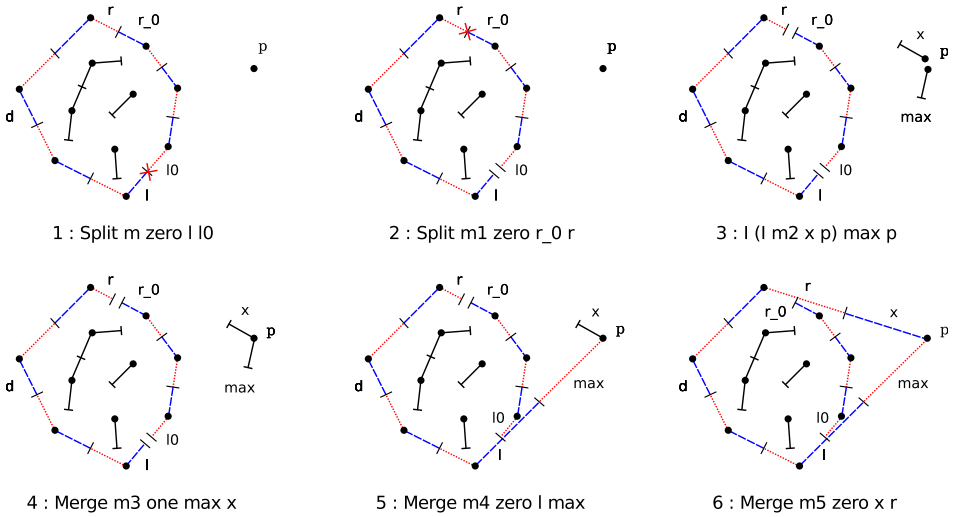


FIGURE 2.12 – Les six étapes de la fonction d'insertion `CHID`

Maintenant que nous avons décrit formellement dans Coq les fonctions de calcul de l'enveloppe convexe suivant deux approches différentes, nous allons montrer que celles-ci vérifient bien les propriétés attendues pour l'enveloppe convexe.

2.5 Propriétés formellement démontrées en Coq

Montrer que les programmes fonctionnels décrits dans Coq calcule bien l'enveloppe convexe d'un ensemble de points du plan nécessite d'établir de nombreuses propriétés topologiques et géométriques. Ces propriétés ne sont pas indépendantes les unes des autres. Certaines propriétés topologiques nécessitent en effet de tenir compte de certains aspects géométriques. Néanmoins, nous les classons en deux catégories et présentons d'abord les propriétés topologiques, puis les propriétés géométriques que nous avons établies.

2.5.1 Propriétés topologiques de l'enveloppe convexe

Propriétés d'invariance topologique sur les cartes combinatoires

La première propriété à établir est la préservation de la structure d'hypercarte par l'algorithme. En effet, l'hypercarte construite doit contenir des brins tous différents, il ne doit y

avoir de coûture entre les brins que quand ils sont déjà présents dans l'hypercarte. Toutes ces propriétés sont regroupées dans l'invariant inv_hmap.

```
Theorem inv_hmap_CH : forall (m:fmap),
  prec_CH m -> inv_hmap (CH m).
```

Il faut également s'assurer que tous les brins initiaux (qui étaient tous *libres* et plongés sur les points dont on cherche l'enveloppe convexe) sont encore présents dans la carte obtenue en résultat.

L'enveloppe convexe est un polygone

Une des propriétés fondamentales de l'enveloppe convexe est d'être un polygone, cela se traduit topologiquement par le fait que chacun de ses sommets et de ses arêtes est de degré exactement 2. Cette propriété se décrit facilement avec les cartes combinatoires. Le prédicat `inv_gmap` exprime que pour tout brin `x` d'une hypercarte `m`, et pour toute dimension `k`, α_k est une involution.

```
Definition inv_gmap (m:fmap) : Prop :=
  forall (k:dim)(x:dart), exd m x -> cA m k (cA m k x) = x.

Theorem inv_gmap_CH : forall (m:fmap),
  prec_CH m -> inv_gmap (fst (CH m)).
```

Il faut également s'assurer que, pour chaque brin `x` appartenant à la même composante connexe que le brin de référence `d` dans l'hypercarte `m`, aucune des `k`-orbites n'admet de point fixe.

```
Definition inv_poly (m:fmap)(d:dart) : Prop :=
  forall (k:dim)(x:dart), eqc m d x -> x <> cA m k x.
```

Cette définition s'appuie sur la notion de composants connexes eqc qui est plus intuitive à manipuler que celle de face. Bien évidemment, si les brins `x` et `y` sont dans la même face, ils sont dans la même composante connexe.

```
Theorem inv_poly_CH : forall (m:fmap),
  prec_CH m -> inv_poly (fst (CH m)) (snd (CH m)).
```

L'enveloppe convexe est planaire

Nous démontrons ensuite que la carte obtenue est bien planaire, ce qui s'exprime par le fait qu'elle est de genre 0.

```
Definition planar (m:fmap) := genus m = 0.

Theorem planar_CH : forall (m:fmap),
  prec_CH m -> planar (CH m).
```

Il reste à établir que le nombre de composants connexes est bien égal à 1 plus le nombre de brins isolés. De même le nombre de faces est égal à 2 (intérieur et extérieur) plus le nombre de brins isolés. Nous n'avons démontré ces deux résultats que dans le cas du deuxième algorithme.

2.5.2 Propriétés géométriques

Une fois les principales propriétés topologiques établies, nous travaillons sur les propriétés géométriques de nos programmes de calcul de l'enveloppe convexe.

La première propriété géométrique à vérifier est que les brins sont plongés de manière cohérente dans le plan. Nous montrons ensuite que la carte construite vérifie bien la propriété de l'enveloppe convexe.

Correction du plongement

Il s'agit de vérifier que deux brins correspondants au même sommet sont bien plongés sur le même point et que deux brins appartenant à la même arête sont plongés sur des points distincts.

```
Definition is_well_emb (m:fmap) : Prop :=
  forall (x y : dart), exd m x -> exd m y -> x <> y ->
  let px := fpoint m x in let py := fpoint m y in
  (eqv m x y -> px = py) /\ (eqe m x y -> px <> py).
```

Nous vérifions également qu'une de nos préconditions géométriques, à savoir qu'il n'y a jamais trois points alignés dans les données initiales, est bien toujours vérifiée dans la carte obtenue en résultat.

Propriété de convexité

La dernière propriété est aussi la plus importante du point de vue géométrique : elle affirme que la carte que nous avons construit est bien convexe. Une face (identifiée par une carte m

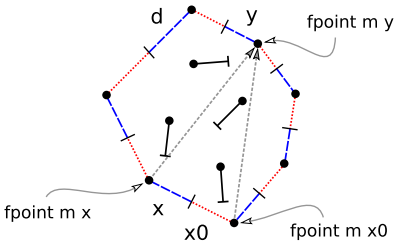


FIGURE 2.13 – La propriété de convexité

et un brin d) est convexe si et seulement si pour chaque brin x de la face (`eqf m d x`) et pour chaque brin y dont le plongement est différent de ceux de x et du 0-successeur `x0 := cA m zero x` de x, alors le triplet de (px,$px0$,py) est orienté dans le sens inverse des aiguilles d'une montre, comme le montre la figure 2.13.

```
Definition is_convex (m:fmap)(d:dart) : Prop :=
  forall (x:dart)(y:dart), eqf m d x -> exd m y ->
  let px := fpoint m x in let py := fpoint m y in
  let px0 := fpoint m (cA m zero x) in
  px <> py -> px0 <> py -> ccw px px0 py.

Theorem convex : forall (md:mapdart)(x max :dart)(p:point),
  is_convex (fst (CHID md x p max)).
```

En regroupant toutes les propriétés de la section 2.5, nous obtenons une preuve formelle de correction de nos programmes de calcul de l'enveloppe convexe d'un ensemble de points du plan.

2.6 Implantation concrète et intégration à une bibliothèque de modélisation géométrique

Il est assez facile de transposer la structure de données inductive des cartes vers une structure en C++ à base de pointeurs. De plus le second programme proposé étant récursif terminal, il peut facilement être implanté de manière impérative en C++. Nous adaptons alors notre programme afin de l'intégrer dans la bibliothèque de modélisation géométrique CGoGN (Combinatorial and Geometric mOdeling with Generic N-dimensional Maps) [IGG]. La transformation est réalisée manuellement, la plupart des opérations restent les mêmes et sont très proches de celles écrites initialement en Coq.

Implantation des hypercartes en C++

Dans la bibliothèque CGoGN, les hypercartes sont représentées par des listes de brins doublement chaînées héritées de la STL (Standard Template Library) de C++. Un brin dans un hypercarte est un pointeur vers une structure contenant un tableau de brins permettant de décrire la topologie (c'est-à-dire les liens entre le brin et ses prédecesseurs et ses successeurs aux deux dimensions) et un pointeur vers une structure de point avec un compteur permettant de connaître le nombre de brins plongés sur ce point (voir figure 2.14). Les hypercartes ont le type `hmap`. On construit le type de données `mapdart` comme une paire contenant une carte et un brin. Nous implantons également les fonctions d'accès `fstmd` et `sndmd`, ainsi que le constructeur `pairmd`.

```
typedef struct mapdartstruct { hmap m; dart d; } mapdart;
```

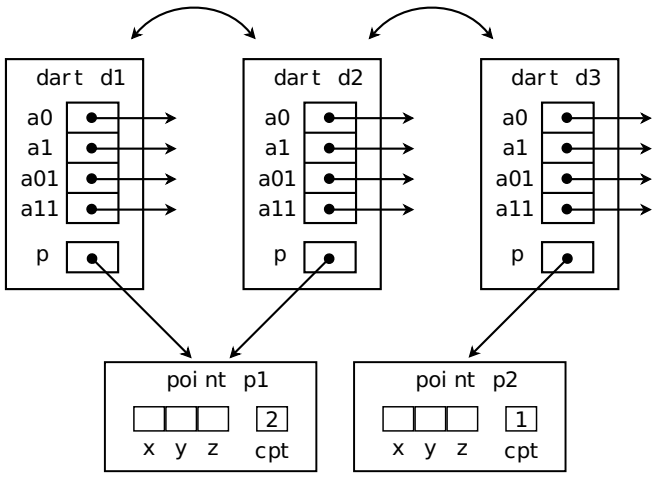


FIGURE 2.14 – Représentation des hypercartes en C++ dans CGoGN

Dérivation d'un programme C++ de calcul de l'enveloppe convexe

Nous programmons facilement les opérations atomiques `V`, `I` et `L` de notre bibliothèque ainsi que les opérations de plus haut niveau `Merge` et `Split`. Ces dernières sont programmées avec des effets de bord sur l'hypercarte `m` et réécrivent directement les liens déjà existants. Les deux fonctions Coq de calcul des extrémités gauche et droite (`search_left` et `search_right`) sont réécrites en deux fonctions récursives similaires en C++.

Les fonctions principales `CH` et `CHI` ne fonctionnent plus par filtrage sur la carte initiale `m` mais exigent d'extraire et de supprimer explicitement chaque brin en utilisant les fonctions adéquates de la bibliothèque CCoGN. Finalement, la fonction d'insertion `CHID` fonctionne ainsi : pour créer un brin dans la carte, on utilise la fonction `gendart` qui retourne un identifiant (un pointeur), et ce résultat est utilisé par la fonction `I` qui insère le brin et son plongement.

```
mapdart CHID (mapdart md, const point p) {
  hmap m = fstmd(md); dart d = sndmd(md);
  dart l = search_left(m,d,p,0);
  dart r = search_right(m,cA(m,zero,d),p,0);
  if (l==m->nil()) {
    dart x = gendart(m);
    m = I(m,x,p);
    md = pairmd(m,d);
  } else {
    dart l0 = cA(m,zero,l); dart r_0 = cA_1(m,zero,r);
    m = Split(m,zero,l,l0);
    if (l0!=r) m = Split(m,zero,r_0,r);
    dart x = gendart(m);
    m = I(m,x,p);
    dart max = gendart(m);
    m = I(m,max,p);
    m = Merge(m,one,max,x);
    m = Merge(m,zero,l,max);
    m = Merge(m,zero,x,r);
    md = pairmd(m,x); }
  return md; }
```

En complément, nous proposons aussi une interface graphique en C++ pour les programmes que nous dérivons de nos spécifications en Coq. Cela permet à la fois de tester nos programmes et également d'observer graphiquement leurs entrées et leurs sorties.

Complexité

La complexité de la fonction C++ d'insertion d'un brin dans une carte `CHI`, dérivée de sa description formelle en Coq est $O(n^2)$ dans le pire cas, avec n le nombre de points de l'ensemble initial. Dans le code des fonctions `search_left` et `search_right`, les appels à `Iter (cF m) i d` sont remplacés par un simple appel à la fonction ϕ qui retourne le successeur d'un brin dans la face de `d`. Cela évite de recalculer les `i` premiers successeurs de `d` à chaque étape. Par conséquent, la complexité globale de l'algorithme est $O(n^2)$, comme pour toute implantation de l'algorithme incrémental. En effet, obtenir une meilleure complexité nécessite de connaître *a priori* l'ensemble des points et donc d'utiliser d'autres algorithmes comme la marche de Jarvis (aussi appelé algorithme du papier cadeau) [Jar73] ou bien l'algorithme de Graham [Gra72].

2.7 D'autres représentations informatiques des cartes

En plus de la description inductive des cartes utilisée jusqu'ici, d'autres approches pour modéliser les cartes combinatoires sont possibles. Nous en présentons rapidement deux. La première a été étudiée au début des années 2010 et la seconde est actuellement à l'étude dans le cadre d'une collaboration avec Alain Giorgetti de l'Université de Franche-Comté.

2.7.1 Modélisation des cartes par liste de brins en C

Il est assez naturel de modéliser les cartes combinatoires par des listes de brins. C'est une structure de données bien adaptée pour décrire les cartes et leurs opérations en C. Au cours du stage de Yasmine Harbit en 2009, nous avons proposé une implantation impérative des cartes combinatoires et leurs opérations (ajout, suppression, coûture) en C et nous avons étudié comment prouver des propriétés comme la légitimité de la coûture (les deux brins cousus doivent obligatoirement être déjà présentés dans la carte) ou encore la cohérence des orbites lors de l'insertion ou la suppression de brins. Ces travaux ont été menés dans l'environnement de développement formel Frama-C [CKK+12].

2.7.2 Une approche basée sur les permutations et les types dépendants

Dans le cadre d'une collaboration récente avec Alain Giorgetti, nous étudions comment construire formellement des bijections entre des représentations non trivialement équivalentes des cartes en Coq. Nous travaillons actuellement sur une bijection entre les cartes ordinaires enracinées (décrites par le type inductif dépendant `rom`) et les cartes locales (décrites dans le type `map`), où la première permutation est vue comme une rotation transitive sur les sommets et où la seconde permutation est une involution locale - $p(2i) = 2i + 1$ et $p(2i + 1) = 2 * i$ - et n'a donc pas besoin d'être encodée.

```
Inductive rom : nat -> Type :=
| mty : rom 0                                              (* leaf node *)
| bin : forall e1 e2, rom e1 -> rom e2 -> rom (e1+e2+1) (* binary nodes *)
| unl : forall e k, k <= 2*e -> rom e -> rom (S e).     (* unary labeled nodes *)

Record map (e : nat) : Set := {
 rotation : permut (2*e);
 transitive : is_transitive rotation }.
```

Nous utilisons des techniques d'énumération combinatoire pour compter les objets d'une certaine taille dans ces deux représentations et s'assurer qu'il y en a bien le même nombre, condition nécessaire à l'existence d'une bijection entre les deux représentations. Nous définissons ensuite en Coq les fonctions de traduction d'une représentation dans l'autre. Avant de prouver formellement que les fonctions aller et retour sont bien des bijections réciproques, nous les testons partiellement avec l'outil `QuickChick` [LP18], ce qui nous permet de corriger les erreurs éventuellement présentes dans notre spécification. A ce jour, ces premières étapes sont terminées et nous sommes dans la phase de production de la preuve formelle en Coq du résultat recherché. A plus long terme, nous souhaitons démontrer formellement en Coq d'autres propriétés de correspondance entre structures de données combinatoires, notamment entre des familles de cartes et des familles de λ-termes (par exemple les cartes planaires enracinées et les λ-termes planaires, comme cela est fait dans [ZG15]).

2.8 Conclusions et perspectives

Nous avons présenté dans ce chapitre une méthodologie de modélisation en Coq d'objets géométriques en s'appuyant sur la notion d'hypercarte. Nous avons décrit formellement dans Coq deux variantes de l'algorithme d'insertion d'un point dans une enveloppe déjà construite : une première version purement récursive sur la structure inductive des cartes combinatoires et une seconde, récursive non-structurelle mais plus proche des implantations usuelles en géométrie

algorithmique, qui identifie les arêtes en conflit et parcourt l'enveloppe convexe existante pour éliminer les arêtes devenues inutiles et construire à partir des sommets extrémaux gauche et droit et du point inséré la nouvelle partie de l'enveloppe convexe. Nous avons ensuite prouvé formellement que ces deux algorithmes calculent bien l'enveloppe convexe de l'ensemble des points fournis en entrée. Pour cela, nous avons démontré que les propriétés topologiques et géométriques de l'enveloppe convexe sont bien conservées au cours de l'opération d'insertion.

Ces premières applications des méthodes formelles de spécification et de preuve à la géométrie algorithmique montrent que les cartes combinatoires et la modélisation géométrique à base topologique sont des approches bien adaptées pour décrire des algorithmes géométriques et en prouver formellement des propriétés. Les représentations basées sur les cartes semblent notamment bien adaptées à la modélisation d'algorithmes en dimensions supérieures (3D et plus). En effet, autant en 2D on pouvait se contenter de structures de données simples (des listes de points par exemple) comme l'on fait Meikle et Pichardie au début des années 2000 [MF06, PB01], autant en 3D et plus, l'utilisation de structures de données plus complexes comme les cartes combinatoires devient indispensable.

Parallèlement à nos travaux, Dufourd et Bertot [DB10] ont formalisé en Coq un algorithme de calcul d'une triangulation de Delaunay dans le plan en utilisant également les cartes combinatoires. Une poursuite naturelle de ces travaux pourrait être de démontrer formellement des algorithmes, tels que celui de Fortune [For86] pour calculer le diagramme de Voronoi d'un ensemble de points du plan.

Une perspective de recherche intéressante, bien qu'encore inexplorée à ce jour, serait de démontrer formellement la correction d'algorithmes géométriques en 3D. En effet, dans ce contexte, les cartes combinatoires et les outils associés présentés dans ce chapitre prendraient tout leur sens pour décrire la topologie d'objets géométriques en trois dimensions. Un premier exemple pourrait être de décrire formellement l'algorithme incrémental de calcul de l'enveloppe convexe d'un ensemble de points en 3D [dBCvKO08, Chap.11] et d'en prouver la correction.

Les travaux réalisés sur la formalisation d'algorithmes géométriques et notamment ceux sur l'enveloppe convexe ont mis en évidence deux écueils qui freinent le développement de bibliothèques de preuves formelles traitant d'algorithmes géométriques. D'une part, les preuves sont longues et parfois fastidieuses à écrire. Dans ce cadre, des outils d'aide à la preuve (automatisation partielle) seraient les bienvenus. D'autre part, la modélisation en Coq d'une représentation constructive des nombres réels, permettrait de s'affranchir des limitations causées par l'utilisation des nombres flottants.

Chapitre 3

Automatisation des preuves en géométrie d'incidence projective

3.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, formaliser et prouver la correction d'algorithmes géométriques nécessite un effort de preuve important. En effet, de tels développements formels combinent des aspects algorithmiques (récursivité, terminaison, conservation d'invariants) et des considérations géométriques (orientation, alignement, convexité). Les aspects géométriques des démonstrations sont souvent fastidieux, avec de nombreux cas particuliers à traiter. Certains peuvent paraître triviaux, mais sont souvent longs et pénibles à prouver : c'est notamment le cas pour les alignements que nous avons écartés dans le chapitre précédent. Notre objectif dans ce chapitre est de voir comment automatiser certains cas simples et d'évaluer jusqu'où l'automatisation peut aller. Nous cherchons à développer un générateur de preuves formelles, aisément vérifiables par un système de preuves comme Coq, dans le contexte de la géométrie.

Nous nous plaçons dans le cadre de la géométrie d'incidence projective, qui est une des théories les plus simples pour décrire de nombreux aspects de la géométrie. Elle est basée sur la relation d'incidence entre les points et les droites. Dans sa version projective, on suppose que deux droites co-planaires se coupent toujours. Un de ses principaux avantages est qu'elle peut être décrite avec un très petit nombre d'axiomes. Cela en fait un candidat bien adapté pour faire des expériences d'automatisation des démonstrations. De plus, elle se décrit facilement soit de manière synthétique, en énonçant les propriétés requises pour la relation d'incidence $\in$, soit de manière plus combinatoire en utilisant la notion de rang d'un ensemble de points. Par exemple le fait que toute droite contient au moins trois points distincts s'écrit dans un cas :

$$\forall l : \mathit{Line}, \exists ABC : \mathit{Point}, A \neq B \wedge B \neq C \wedge A \neq C \wedge A \in l \wedge B \in l \wedge C \in l$$

et dans l'autre cas :

$$\forall AB : \mathit{Point}, rk\{A, B\} = 2 \;\Rightarrow\; \exists C, rk\{A, B, C\} = rk\{B, C\} = rk\{A, C\} = 2.$$

Dans la deuxième approche, on ne manipule plus que des points et le raisonnement géométrique est remplacé par du calcul sur la combinatoire du matroïde sous-jacent à la notion de rang. L'ensemble de 2 points distincts A et B est de rang 2. L'ensemble de 3 points colinéraires A, B et C est également de rang 2. Le plan tout entier en dimension 2 est de rang 3. En dimension $n \geq 3$, le rang maximum d'un ensemble de points est $n + 1$. Dans le cas $n = 3$, un ensemble de rang 4 n'est pas un plan, et capture l'espace tout entier.

Nous commençons par formaliser dans Coq ces deux descriptions différentes de la géométrie projective : l'approche synthétique, telle qu'elle est habituellement présentée dans la littérature [Cox03, Bue95a] et l'approche combinatoire basée sur les notions de matroïdes et de rangs, proposée par Michelucci et Schreck [MS06]. Ces deux approches sont complémentaires et en fonction du problème considéré, il est préférable de choisir l'une ou l'autre de ces formalisations.

Nous nous attachons dans un premier temps à démontrer formellement l'équivalence entre ces deux modélisations de la géométrie projective. Nous étudions ensuite les propriétés de quelques plans et espaces projectifs de petite taille pour déterminer l'approche la plus adaptée en fonction du problème considéré. Cela permet notamment d'observer expérimentalement que la propriété de Desargues est généralement plus facile à démontrer en suivant l'approche combinatoire, alors que simplement vérifier qu'un ensemble de points et de droites vérifient bien les axiomes de la géométrie projective est plus simple à faire dans le cadre synthétique. Enfin, nous étudions comment tirer profit des règles de matroïde de l'approche combinatoire pour construire un prouveur automatique et nous le validons en générant automatiquement des preuves en Coq de plusieurs théorèmes emblématiques de la géométrie projective.

Les travaux décrits dans ce chapitre ont débuté en 2008 en collaboration avec Julien Narboux et Pascal Schreck [MNS08, MNS09, MNS12] et se sont poursuivis pendant la thèse de David Braun que j'ai co-encadré avec Pascal Schreck de 2015 à 2019. Les résultats obtenus durant cette thèse sont décrits dans les articles suivants : [BMS19, BMS18, BMS20].

3.2 Système d'axiomes pour la géométrie projective

La géométrie est un domaine très riche où de nombreux objets (points, droites, plans, hyperplans) coexistent avec de nombreuses propriétés (distance, angle, continuité, incidence, etc.). La géométrie d'incidence ne retient que les points et les droites et ne conserve que la notion d'incidence entre un point et une droite.

3.2.1 Géométrie d'incidence

La géométrie d'incidence peut être modélisée par une structure constituée d'un ensemble de points Ω, d'un ensemble de droites Δ et d'une relation binaire Φ où $\Phi(x, l)$ exprime que le point x est sur la droite l. On ajoute ensuite les règles minimales suivantes [Bat97] :

— Il existe toujours une droite passant par 2 points.

— Sur une droite, il y a au moins 2 points.

— Il existe 3 points qui ne sont pas colinéaires.

— Deux droites distinctes se coupent en *au plus* un point.

Cette description minimaliste permet déjà de prouver un certain nombre de théorèmes géométriques [Bue95c]. La géométrie projective est construite en ajoutant une propriété supplémentaire, à savoir que : *deux droites coplanaires se coupent toujours*.

3.2.2 Un système d'axiomes pour la géométrie projective plane

Nous présentons un système d'axiomes pour la géométrie projective plane, proposé initialement par Veblen et Young [VY18], puis repris par Coxeter [Cox03]. Il se compose de 5 axiomes dont les énoncés mathématiques sont donnés dans la figure 3.1. Une illustration géométrique est également présentée dans la figure 3.2.

Les 2 premiers axiomes (A1P2, A2P2 [1]) servent à construire des points et des droites. Pour rester général, nous n'imposons pas que les points (ou les droites) mis en jeu soient distincts. Dans le cas où ces éléments seraient égaux, ces axiomes resteraient corrects. En effet, il existe toujours une droite passant par un point, et il existe toujours un point sur une droite.

L'axiome (A3P2) établit l'unicité des objets définis. L'axiome (A4P2) exprime que toute droite contient au moins trois points. Cette propriété plus forte que la règle informelle de la section précédente permet d'éviter que certaines configurations dégénérées soient considérées comme des plans projectifs.

L'axiome (A5P2) exprime qu'il existe toujours deux droites distinctes, ce qui impose que la dimension soit au moins 2. L'axiome (A2P2) *Point-Existence* imposant que la dimension soit au plus 2, la dimension est donc exactement 2.

(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) 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

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

FIGURE 3.1 – Système d'axiomes usuel pour la géométrie projective plane

Ce système d'axiomes se modélise très facilement dans Coq. Il en existe de nombreuses variantes [Bue95c, Cox03]. Nous avons par exemple montré dans [MNS08] que l'on peut remplacer les axiomes (A4P2, A5P2) par un unique axiome (A4P2') exprimant qu'il existe au moins 4 points et qu'aucun sous-ensemble de 3 points n'est sur la même droite.

(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))

1. Dans la suite de ce document, nous utiliserons la convention de nommage suivante : $A\#YN$ pour les axiomes. A signifie axiome, $\#$ correspond à son numéro, Y vaut soit P = projectif ou bien R =rang. Enfin le dernier élément précise la dimension.

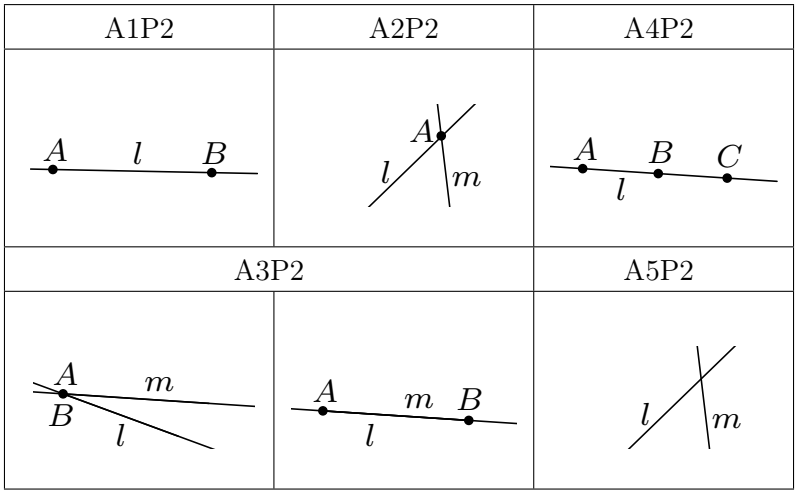


FIGURE 3.2 – Vue géométrique des axiomes usuels de la géométrie projective plane

3.2.3 Un système d'axiomes pour la géométrie projective dans l'espace

Le système d'axiomes pour les plans projectifs se transforme facilement en un système d'axiomes pour les espaces projectifs de dimension supérieure ou égale à 3. L'axiome d'existence d'un point par intersection de 2 droites (A2P2) est remplacé par l'axiome (A2P3), habituellement appelé axiome de *Pasch* exprimant que deux droites **coplanaires** se coupent toujours. L'axiome (A5P2) *Lower-Dimension* est remplacée par un nouvel axiome exprimant qu'il existe 2 droites qui ne se coupent pas (capturant ainsi l'aspect 3D et plus). Enfin, si l'on souhaite limiter la dimension à exactement 3, on ajoute l'axiome (A6P3) exprimant qu'il existe toujours une droite coupant 3 autres droites (coplanaires ou non).

Les énoncés mathématiques des axiomes sont regroupés dans la figure 3.3 et un aperçu géométrique est donné dans la figure 3.4.

3.3 Approche combinatoire avec la notion de rang

Après avoir présenté une approche synthétique de la géométrie projective dans la section précédente, nous allons voir comment décrire cette même géométrie en utilisant une approche combinatoire basée sur la théorie des matroïdes et la notion de rang.

3.3.1 Propriétés des matroïdes

La théorie des matroïdes [Oxl06] a été introduite en 1935 par Whitney pour capturer l'essence des dépendances des colonnes de matrices. Les matroïdes permettent en effet de décrire et d'étendre les propriétés principales de la dépendance linéaire dans les espaces vectoriels. La définition de Whitney recouvre cependant un nombre plus large de structures combinatoires. Cette théorie, très générale, a des applications dans différents domaines de l'informatique, notamment l'algorithmique et la théorie des graphes [Edm71, Tut59].

Utilisée avec des ensembles finis de points, la théorie des matroïdes permet de capturer les propriétés d'incidence (collinéarité, coplanarité, etc.) entre ces points. Dans ce contexte, nous introduisons la notion de rang d'un ensemble de points, permettant d'en capturer aisément la dimension (point, droite, plan, espace tout entier). La notion de rang permet de ne travailler

(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, $P_1 \in l_1 \wedge P_1 \in l_4 \wedge$
$P_2 \in l_2 \wedge P_2 \in l_4 \wedge P_3 \in l_3 \wedge P_3 \in l_4$

FIGURE 3.3 – Axiomes usuels pour la géométrie projective dans l'espace

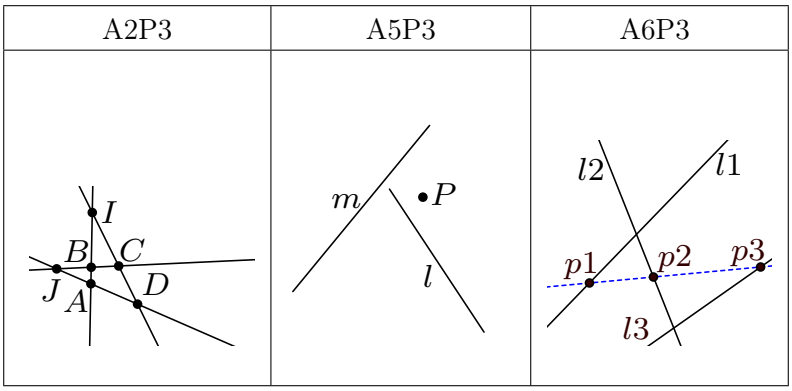


FIGURE 3.4 – Vue géométrique des axiomes de la géométrie projective dans l'espace

qu'avec des points, ce qui est très avantageux pour automatiser les preuves. Nous n'avons en effet plus qu'un type d'objets à manipuler. Notre système ne traite plus de manière explicite les droites, les plans et les éventuels objets de dimension supérieure. Comme nous allons le voir, nous pouvons décrire tous les concepts de la géométrie d'incidence avec la fonction de rang.

Une fonction à valeurs entières *rk* , définie sur un ensemble E fini est une fonction de rang pour un matroïde si et seulement elle vérifie les conditions de la figure 3.5 .

Il existe de multiples manières de définir les matroïdes et nous avons retenu cette approche parce qu'elle apparaît bien adaptée à notre cadre d'étude.

(A1R2-R3) Non-Negative and Subcardinal : $\forall$ X $\subseteq$ E, 0 $\leq$ rk(X) $\leq \left|\mathrm{X}\right|$

(A2R2-R3) Non-Decreasing : $\forall$ X $\subseteq$ Y, rk(X) $\leq$ rk(Y)

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

FIGURE 3.5 – Propriétés de matroïde de la fonction de calcul du rang

3.3.2 Notion de rang pour décrire la géométrie projective

Dans le cadre de la géométrie projective, nous définissons une fonction de rang sur des ensembles finis de points. Cette fonction vérifie les propriétés de la figure 3.5.
Considérons M un matroïde sur un ensemble fini E avec une fonction de rang rk comme ci-dessus. La clôture cl d'un sous-ensemble F de E est l'ensemble $cl(F) = \{x \in E \mid rk(F) = rk(F \cup \{x\})\}$. Un ensemble dont la clôture est égale à lui-même est appelé un *plat* [Bue95b]. Un ensemble est un plat s'il est maximum pour le rang, ce qui signifie que l'ajout d'un élément à l'ensemble augmente son rang. En d'autres termes, le rang d'un plat F est le cardinal du plus petit ensemble générateur de F. La fonction usuelle de rang pour la géométrie d'incidence est définie en identifiant les plats avec les sous-espaces linéaires. Quelques exemples d'ensembles de points et leurs rangs respectifs sont présentés dans la figure 3.6.

rk{A,B} = 1	A = B
rk{A,B} = 2	A $\neq$ B
rk{A,B,C} = 2	A,B,C sont alignés avec au moins 2 points distincts
rk{A,B,C} $\leq$ 2	A,B,C sont alignés
rk{A,B,C} = 3	A,B,C ne sont pas alignés
rk{A,B,C,D} = 3	A,B,C,D sont coplanaires, et ne sont pas alignés
rk{A,B,C,D} = 4	A,B,C,D ne sont pas coplanaires

FIGURE 3.6 – Quelques égalités sur les rangs et leur interprétation géométrique

A partir de cette définition, on peut vérifier que tout espace projectif a une structure de matroïde. Par contre, la réciproque n'est pas vraie. Pour qu'une structure de matroïde représente fidèlement la géométrie projective, il est nécessaire d'ajouter des axiomes supplémentaires. C'est ce que nous allons étudier dans les deux prochaines sections pour la 2D, puis pour la 3D et plus.

3.3.3 Un système d'axiomes basé sur les rangs en 2D

Afin de capturer la géométrie projective avec notre système de rang, il est nécessaire d'ajouter cinq axiomes modélisant les aspects géométriques en plus des propriétés matroïdales de la fonction de rang. La figure 3.7 présente les axiomes à ajouter pour décrire la géométrie projective plane. Les 2 premiers axiomes assurent que la fonction de rang n'est pas dégénérée. Les suivants sont des traductions plus ou moins directs des axiomes de l'approche synthétique.

3.3.4 Un système d'axiomes basé sur les rangs en 3D et plus

Le système d'axiomes pour la 3D et plus est présentée dans la figure 3.8. Les axiomes (A6R3) *Pasch*, (A7R3) *Rk-Three-Points* et (A8R3) *Lower-Dimension* sont les traductions des axiomes

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

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

FIGURE 3.7 – Une extension du système d'axiomes sur les rangs pour capturer la géométrie projective plane

correspondants (A2P3), (A4P3) et (A5P3). L'axiome (A9R3) permet de borner la dimension et donc de se placer si nécessaire dans le cadre de la 3D exactement.

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

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

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

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

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

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

FIGURE 3.8 – Une extension du système d'axiomes sur les rangs pour capturer la géométrie projective dans l'espace

La description en Coq de ce système d'axiomes s'appuie sur les classes de type pour partager au maximum les parties communes indépendantes de la dimension.

3.3.5 Un exemple d'application : preuve du théorème de Desargues

En utilisant le système d'axiomes basé sur les rangs, nous avons prouvé formellement en Coq la correction du théorème de Desargues en géométrie projective [MNS12]. Cette preuve très calculatoire, qui est basée sur un relevage en 3D de la figure 2D, a nécessité de trouver les bons

sous-ensembles de points à considérer pour aboutir au fait que les points α, β et γ sont bien alignés (voir Fig. 3.9).

Dans un premier temps, nous démontrons le théorème dans le cas où les deux triangles mis en jeu ne sont pas coplanaires :

Théorème 1 (Desargues 3D)**.** *Considérons 2 triangles non dégénérés ABC et abc qui soient en perspective depuis le point o :*

$$rk\{A, B, C\} = rk\{a, b, c\} = 3$$
$$rk\{a, A, o\} = rk\{b, B, o\} = rk\{c, C, o\} = 2$$

On suppose que cela forme une figure non plane (plongée dans un espace 3D) :

$$rk\{A, B, C, a, b, c\} \geq 4$$

et on définit trois points α, β, γ tels que :

$$rk\{A, B, \gamma\} = rk\{a, b, \gamma\} = 2$$
$$rk\{A, C, \beta\} = rk\{a, c, \beta\} = 2$$
$$rk\{B, C, \alpha\} = rk\{b, c, \alpha\} = 2$$

Sous ces hypothèses, on a la propriété : $rk\{\alpha, \beta, \gamma\} \leq 2$.

Dans le cas où les deux triangles mis à jeu sont coplanaires, il suffit de relever un des triangles, ici le triangle $A'B'C'$ vers un triangle abc, non-coplanaire avec le triangle ABC en utilisant un point P, extérieur au plan de la configuration 2D. Il ne reste plus qu'à vérifier que les points α, β et γ construits comme les intersections des côtés semblables des triangles sont bien les mêmes dans les 2 cas. Le théorème en 2D s'énonce alors de la manière suivante :

Théorème 2 (Desargues 2D)**.** *Considérons 2 triangles non dégénérés ABC and $A'B'C'$ qui soient en perspective depuis le point O :*

$$rk\{A, B, C\} = rk\{A', B', C'\} = 3$$
$$rk\{A', A, O\} = rk\{B', B, O\} = rk\{C', C, O\} = 2$$

On suppose que cela forme une figure plane :

$$rk\{A, B, C, A', B', C', O\} = 3$$

et on définit trois points α, β, γ tels que :

$$rk\{A, B, \gamma\} = rk\{A', B', \gamma\} = 2$$
$$rk\{A, C, \beta\} = rk\{A', C', \beta\} = 2$$
$$rk\{B, C, \alpha\} = rk\{B', C', \alpha\} = 2$$

Sous réserve que les conditions de non-dégénérescence suivantes sont vérifiées :

$$rk\{A, B, O\} = rk\{A, C, O\} = rk\{B, C, O\} = 3$$
$$rk\{A, A'\} = rk\{B, B'\} = rk\{C, C'\} = 2$$

On a la propriété : $rk\{\alpha, \beta, \gamma\} \leq 2$.

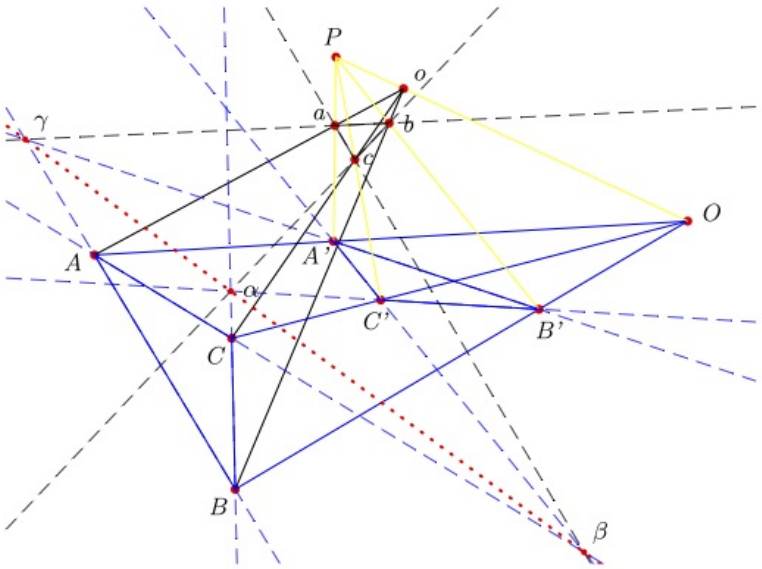


FIGURE 3.9 – Le théorème de Desargues (extrusion 3D) : A,B,C,A',B',C',O,α,β and γ sont co-planaires. En supposant que P est le soleil, l'ombre du triangle abc est le triangle A'B'C'.

Le développement complet de la preuve a nécessité l'écriture de 3 600 lignes de spécification et de 10 400 lignes de preuve dans l'assistant de preuve Coq. Tout cela a été fait de manière manuelle, même si nous avons exploité quelques symétries pertinentes afin de factoriser certaines parties de démonstration.

Cette première preuve formelle en Coq a permis de valider la pertinence de l'approche basée sur les rangs pour prouver formellement des théorèmes de la géométrie projective dans Coq. Ces travaux se sont naturellement poursuivis dans deux directions que nous présentons dans la suite de ce chapitre. Il s'agit tout d'abord d'établir une correspondance formelle entre les descriptions synthétique et combinatoire de la géométrie projective. Cela permet d'utiliser l'approche la plus adaptée à chaque problème et également de passer d'une approche à l'autre quand on le souhaite. En effet, l'approche synthétique propose des énoncés lisibles, mais est peu adaptée pour la démonstration automatique alors que l'approche combinatoire propose des énoncés très numériques et moins lisibles, mais est très bien adaptée pour la démonstration automatique. Ensuite nous montrons comment l'approche combinatoire basée sur les rangs peut être utilisée pour construire un prouveur automatique capable de démontrer des propriétés géométriques comme le théorème de Desargues modulo la construction des points auxiliaires.

3.4 Equivalence entre les deux systèmes d'axiomes

Nous établissons maintenant le chaînon manquant entre la représentation synthétique et la représentation combinatoire de la géométrie projective. Afin de pouvoir utiliser indifféremment les deux approches, nous prouvons formellement que pour chaque dimension pertinente (2D, $\geq$3D, et 3D), le système d'axiomes de l'approche synthétique est équivalent au système d'axiomes basé sur les matroïdes et les rangs.

Théorème 3. *Pour chaque dimension pertinente (2D, $\geq$3D, et 3D), le système d'axiomes de l'approche synthétique est équivalent au système basé sur les matroïdes et les rangs.*

3.4.1 De la description matroïdale vers la description synthétique

En prenant comme hypothèses les axiomes de la description matroïdale, nous montrons que toutes les propriétés énoncées comme axiomes dans la description synthétique sont prouvables formellement en Coq. Les preuves sont similiaires quelle que soit la dimension considérée.

Nous définissons en termes de points et de rangs uniquement les notions de base de la description synthétique, à savoir les notions de points (Point) et de droites (Line) et la relation d'incidence Incid. Les définitions en Coq sont présentées dans la figure 3.10. Une droite est construite à partir de 2 points distincts. Un point P est incident à une droite $l = (AB)$ si le rang du triplet de points constitué des points caractérisant la droite (A et B) et du point P vaut exactement 2. Enfin deux droites l=AB et m=CD sont égales si le rang du quadruplet A,B,C,D vaut exactement 2.

```
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(triple (fstP l)(sndP l) P) = 2.

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

FIGURE 3.10 – Modélisation de la description synthétique en termes de rangs

Nous pouvons maintenant prouver formellement, pour chaque dimension, les 5 axiomes de la description synthétique de la géométrie projective (il s'agit des propriétés de la figure 3.1 pour le plan et des propriétés de la figure 3.3 pour l'espace).

Techniques de preuve

Nous présentons quelques techniques de preuves souvent utilisées.

Premièrement, les axiomes de matroïdes prenant la forme d'inégalités, la plupart des propriétés d'égalité entre rangs ($rk(a) = rk(b)$) sont démontrées en 2 étapes $rk(a) \leq rk(b)$, puis $rk(a) \geq rk(b)$. Afin d'éviter des démonstrations inutiles, il est utile de vérifier avant d'énoncer un lemme d'égalité si l'on ne peut pas se contenter d'une seule des inégalités correspondantes.

Deuxièmement, la propriété de sous-modularité (A3R2-R3) est une propriété clé de la théorie des matroïdes et elle est utilisée intensivement dans les démonstrations. Cette propriété nécessite de calculer l'intersection de 2 ensembles. Considérons que l'on cherche à montrer que

$$rk\{A, B, C, D, I\} + rk\{I\} \leq rk\{A, B, I\} + rk\{C, D, I\},$$

on pourrait être tenté d'utiliser la sous-modularité avec $X := \{A, B, I\}$ et $Y := \{C, D, I\}$. Malheureusement cet énoncé n'est pas une conséquence directe de la propriété de sous-modularité. Les points A et C peuvent être égaux et par conséquent $\{A, B, I\} \cap \{C, D, I\} = \{A, I\}$. Déterminer l'intersection de deux ensembles finis de points nécessite de faire des distinctions de cas sur l'égalité des points, ce qui conduit à des étapes de preuve pénibles en Coq. Nous considérons donc une version affaiblie de l'intersection, qui nous appellons intersection littérale.

Définition 7 (Intersection littérale)**.** *Soient L_1 et L_2 deux ensembles de points. $L_1 \sqcap L_2$ est l'intersection des deux ensembles de points, du point de vue syntaxique.*

Comme $X \sqcap Y \subseteq X \cap Y$, nous avons, pour tout I, la propriété suivante :

Lemme 1 (A3R2-R3-alt)**.**

$$\forall X\ Y\ I, I \subseteq X \cap Y \Rightarrow rk\{X \cup Y\} + rk\{I\} \leq rk\{X\} + rk\{Y\}$$

Cette variante technique de la propriété (A3R2-R3) est bien plus adaptée et sera utilisée intensivement dans les preuves en Coq.

Troisièmement, le rang d'un ensemble de points n'est pas toujours connu de manière exacte. On ne dispose souvent que d'un encadrement du rang. Dans ce cas, on est amené à faire du raisonnement par cas sur sa valeur. C'est ce qui se passe notamment dans la preuve de l'axiome (A6P3) **Upper-Dimension**. Les trois droites qui sont coupées par une quatrième doivent avoir pour unique contrainte d'être différentes deux à deux. Pour faire cette démonstration, on doit systématiquement distinguer si les couples de droites sont coplanaires ou non.

Un exemple de preuve : la propriété d'unicité

Nous traitons une propriété parmi les 5 à prouver : la propriété (A3P2-P3) *Uniqueness* dont nous rappelons l'énoncé ci-après.

Lemme 2. *(A3P2-P3) Uniqueness*[2]*, $\forall$ 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 par distinguer les cas $A = B$ et $A \neq B$.
Si $A = B$, le lemme est démontré.
Si $A \neq B$, en dépliant les définitions de `Line` et de `Incid`, nous obtenons :

We have $A <> B$ Let P ∈ l, Q ∈ l and $P <> Q$ Let R ∈ m, S ∈ m and $R <> S$ Incid A l ⇒ rk{P, Q, A} = 2 Incid B l ⇒ rk{P, Q, B} = 2 Incid A m ⇒ rk{R, S, A} = 2 Incid B m ⇒ rk{R, S, B} = 2	} Hypothèses
rk{P Q R S} = 2 ⇒ l = m	} Conclusion

Nous commencons par montrer les deux inégalités suivantes : $rk\{P, Q, A, B\}$ <= 2 et $rk\{R, S, A, B\}$ <= 2 en utilisant la sous-modularité.
Pour la première propriété, le raisonnement est le suivant :
rk({P, Q, A} ∪ {P, Q, B}) + rk({P, Q, A} ∩ {P, Q, B})
≤ rk{P, Q, A} + rk{P, Q, B}
⇒ rk{P, Q, A, B} + rk{P, Q} ≤ rk{P, Q, A} + rk{P, Q, B}
⇒ rk{P, Q, A, B} ≤ 2

A partir des deux inégalités obtenues, nous montrons que rk{P, Q, R, S, A, B} ≤ 2.

2. Le lemme `uniqueness` se trouve dans le fichier `rk_equiv_to_pp.v`.

rk({P, Q, A, B} ∪ {R, S, A, B}) + rk({P, Q, A, B} ∩ {R, S, A, B})
≤ rk{P, Q, A, B} + rk{R, S, A, B}
⇒ rk{P, Q, R, S, A, B} + rk{A, B} ≤ rk{P, Q, A, B} + rk{R, S, A, B}
⇒ rk{P, Q, R, S, A, B} ≤ 2

Comme {P, Q, R, S} ⊂ {P, Q, R, S, A, B}, nous en déduisons, en utilisant la propriété A2R2-R3, que rk{P, Q, R, S} ≤ rk{P, Q, R, S, A, B} ≤ 2
Etant donné que l'ensemble {P, Q, R, S} possède au moins 2 éléments distincts, nous pouvons conclure que $rk\{P, Q, R, S\} = 2$. □

Des raisonnements similaires permettent de montrer, un à un, toutes les propriétés de la figure 3.1 à partir de la description basée sur les rangs.

3.4.2 De la description synthétique vers la description matroïdale

Nous nous intéressons maintenant à l'autre sens de l'équivalence. Il faut tout d'abord définir la fonction de rang dans la description synthétique de la géométrie projective.

La définition de la fonction de calcul du rang `rkl` [3] (voir figure 3.11) s'inspire de la notion de plat en théorie des matroïdes. Un ensemble de points est soit vide, soit il représente un point, une droite, un plan ou bien l'espace tout entier.

```
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.
```

FIGURE 3.11 – Définition de la fonction de rang dans le description synthétique de la géométrie projective

Les trois prédicats apparaissant dans la Fig. 3.12 (`contains_four_non_coplanar_points` [4], `contains_three_non_collinear_points` et `contains_two_distinct_points`) testent chacune des dimensions possibles pour l'ensemble de points considéré. On teste d'abord la coplanarité : soit il existe un quadruplet de points non coplanaires et l'ensemble considéré est l'espace tout entier, soit on poursuit l'analyse en testant la collinéarité.

```
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.
```

FIGURE 3.12 – Définition récursive du prédicat `contains_four_non_coplanar_points`. Le prédicat `coplanar_with_all` teste si le premier point est coplanaire avec les autres points.

3. La définition `rkl` se trouve dans le fichier `psoh_equiv_rk_lemmas.v`.

4. La définition `contains_four_non_coplanar_points` se trouve dans le fichier `psoh_equiv_c4p.v` et celle de `coplanar_with_all` utilisée dans la définition de `contains_four_non_coplanar_points` se trouve dans le fichier `psoh_equiv_copwa.v`.

Une fois cette fonction définie, il suffit de prouver, à partir des axiomes de la description synthétique, les neuf propriétés de la description basée sur les rangs.

Techniques de preuves

La plupart des démonstrations procèdent par cas sur les différentes valeurs possibles pour la fonction de rang. Ce sont des entiers naturels compris de 0 à 4 dans le cas de l'espace projectif. L'écriture de tactiques comme `my_rank` [5] présentée dans la figure 3.13 permet de gérer aisément les conditionnelles imbriquées dans la définition de la fonction de rang et tous les cas dégénérés associés.

Une propriété fondamentale à démontrer est que la fonction de rang est bien un morphisme par rapport à l'égalité d'ensembles. En effet, il est très utile de disposer d'un théorème montrant que deux ensembles égaux (i.e. inclus l'un dans l'autre) ont le même rang :

Lemme 3 (rank_morph)**.** $\forall$ *x y : list Point, equivlist x y* $\rightarrow$ `rkl` *x* = `rkl` *y*

```
Ltac my_rank :=
repeat match goal with
  |[H : _ |- _] => [progress intros|progress intro]
  |[H : _ |- _] => solve[intuition]
  |[H : _ |- _] => progress contradiction
  |[H : _ = _ |- _] => solve[inversion H]
  |[H : ?X[==]?X -> False |- _] => apply False_ind;apply H;reflexivity
  |[H : _ |- (if if ?X then _ else _ then _ else _) = _ \/ _] => case_eq X
  ...
  end.
```

FIGURE 3.13 – Un extrait d'une tactique traitant automatiquement les cas dégénérés

Un exemple de preuve : la propriété de sous-modularité

La propriété de sous-modularité (A3R2-R3) est la plus complexe à prouver, notamment à cause des opérations d'union et d'intersection qu'elle met en jeu.

Lemme 4 ((A3R2-R3) **Submodular**)**.** $\forall$ *X,Y* $\subseteq$ *E, rk(X*$\cup$*Y) + rk(X*$\cap$*Y)* $\leq$ *rk(X) + rk(Y)*

De nombreux lemmes, faciles à montrer dans la description synthétique de la géométrie projective, mais très techniques dans la description basée sur les rangs doivent être prouvés. Nous présentons ci-dessous deux exemples représentatifs (où `list_inter` dénote l'intersection littérale sur les listes) :

— Si l'intersection de deux droites l et m est une droite, alors tous les points incidents à ces deux droites sont sur la même droite.

$$\forall l\ m,\ \texttt{rkl}\ l = 2\ \rightarrow\ \texttt{rkl}\ m = 2\ \rightarrow\ \texttt{rkl}(\texttt{list\_inter}\ l\ m) = 2\ \rightarrow\ \texttt{rkl}(\texttt{union}\ l\ m) = 2$$

— Si une droite m est incluse dans le plan p, alors l'ensemble des points de m et p sont tous dans le plan p.

$$\forall l\ m,\ \texttt{rkl}\ p\ =\ 3\ \rightarrow \texttt{rkl}\ m\ =\ 2\ \rightarrow \texttt{rkl}(\texttt{list\_inter}\ p\ m)\ =\ 2\ \rightarrow\ \texttt{rkl}(\texttt{union}\ p\ m) = 3$$

Les preuves des autres propriétés de l'axiomatique basée sur les rangs suivent la même approche.

5. Le code la tactique Ltac `my_rank` se trouve dans le fichier `psoh_equiv_rk_lemmas.v`.

Bilan

Au total, les preuves des deux sens de l'équivalence pour chacune des dimensions pertinente représentent environ 17 000 lignes de preuve en Coq. Le tableau 3.14 présente quelques chiffres-clés en fonction du sens prouvé et de la dimension considérée. Il est à noter que, comme attendu, le travail de spécification et de preuve est beaucoup plus conséquent dès que l'on se confronte à la 3D.

	Des rangs vers la description synthétique		De la description synthétique 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

FIGURE 3.14 – Structure et taille des preuves d'équivalence suivant la dimension considérée

Nous disposons maintenant de deux approches différentes mais complémentaires pour décrire et prouver formellement en Coq des théorèmes de la géométrie projective.

3.5 Etude de cas sur les plans et espaces projectifs finis

Dans cette section, nous évaluons nos deux approches avec le contexte des plans et espaces projectifs finis. Les premiers exemples de géométrie d'incidence sont construits à partir de corps. Les plans affines connus proviennent généralement de F^2, où F est un corps, avec un système de coordonnées et des plans projectifs peuvent être construits à partir de F^3 en utilisant un système de coordonnées homogènes. Les espaces de Fano proviennent du corps $\mathbb{Z}/2\mathbb{Z}$.

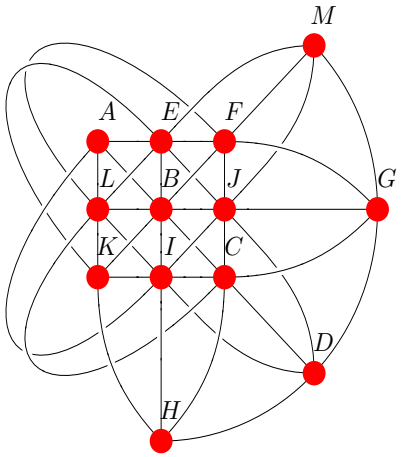


FIGURE 3.15 – Une configuration de $pg(2,3)$: 13 points et 13 droites.

Les corps finis de cardinalité $n = p^k$, notés by $GF(n)$ sont des corps de Galois et ils sont isomorphes au corps $\mathbb{Z}_p[X]/f(X)$ où p est un nombre premier, où $\mathbb{Z}_p$ est l'abbréviation de $\mathbb{Z}/p\mathbb{Z}$ et où f est un polynôme irréductible sur $\mathbb{Z}_p[X]$ de degré k. On en déduit que chaque droite de l'espace affine (resp. projectif) correspondant est de cardinalité n (resp. $n+1$). Les espaces projectifs finis, provenant d'un corps $GF(n)$ sont nommés $pg(d,n)$ où d est la dimension de l'espace

	point(s)	droite(s)	plan(s)
`pg(2,2)`	7	7	1
`pg(2,3)`	13	13	1
`pg(2,4)`	21	21	1
`pg(2,5)`	31	31	1
`pg(3,2)`	15	35	15

FIGURE 3.16 – Nombres de points, droites et plans pour les plus petits plans et espaces projectifs.

et n l'ordre du corps sous-jacent. La figure 3.16 synthétise ces cardinalités et la figure 3.15 donne une vue géométrique du plan projectif $pg(2,3)$.

Nous utilisons les espaces $pg(d,n)$ comme des *benchmarks* pour éprouver nos stratégies de mécanisation des preuves en Coq. Ces géométries sont bien adaptées parce qu'elles sont décrites précisement et que le nombre d'objets mis en jeu croît très vite avec n. Bien que nous travaillons dans le contexte des espaces $pg(d,n)$, nous prenons uniquement en compte leurs caractéristiques géométriques, ce qui signifie notamment que nous n'utilisons pas de coordonnées. Nous établissons facilement que les axiomes de la géométrie projective sont vérifiés pour $pg(2,2)$, $pg(2,3)$ et $pg(2,5)$. De même, nous montrons que les axiomes de la théorie basée sur les rangs sont vérifiés pour $pg(2,2)$ et $pg(2,3)$. Nous nous intéressons également à la propriété de Desargues dans le cadre des plans finis. Tous ces exemples nous servent à analyser et mieux comprendre la complexité des preuves en Coq.

Propriété de Desargues

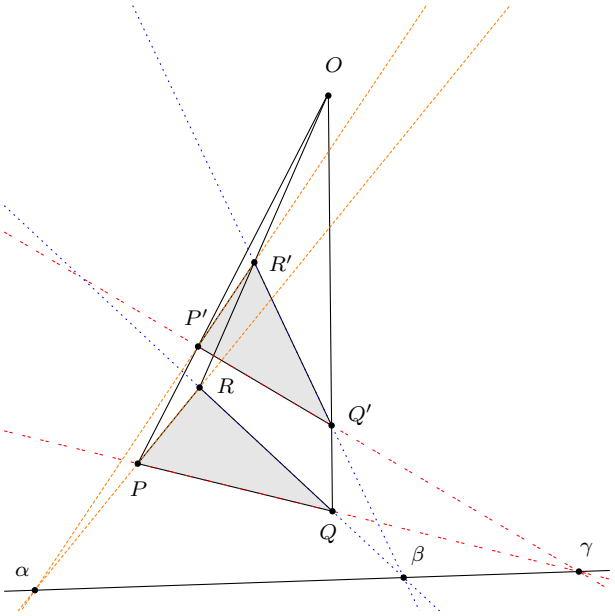


FIGURE 3.17 – Une configuration de la propriété de Desargues.

La propriété de Desargues, dont une configuration est présentée dans la figure 3.17, est vérifiée dans tout espace projectif de dimension au moins égale à 3. Nous avons d'ailleurs formalisé ce résultat en Coq dans la section précédente. Cependant dans le cas des plans projectifs, cette

propriété est indépendante du système d'axiomes de la géométrie projective plane (voir figure 3.1). Ainsi il existe des plans dits Désarguésiens et des plans dits non-Désarguésiens. Le plan de Moulton [Mou02, MNS08] et les plans de Hall [Hal43] d'ordre 9 sont non-Désarguésiens.

3.5.1 Représentation des données et techniques de preuve en 2D

La manière la plus simple de décrire un plan projectif est de modéliser son ensemble de points par un type inductif avec un constructeur par point, son ensemble de droites par un type inductif avec un constructeur par droite et d'écrire comme une fonction booléenne la relation d'incidence. Cette approche basique est plus efficace que d'utiliser un type fini comme (`FinType`, tel qu'il est défini et utilisé dans la bibliothèque *Mathematical Components* [MT16]. En effet, faire des raisonnements par cas est coûteux avec un tel type, notamment parce qu'il faut déconstruire une structure de la forme $\{i : nat | i < n\}$ à chaque étape.

Cependant, l'appproche basée sur des types inductifs simples n'est possible que quand les ensembles considérés sont petits. Déjà, pour pg(2,5) par exemple, le nombre de points et de droites est déjà égal à 31. Il est alors nécessaire de disposer d'outils de génération automatique de la spécification en plus d'outils pour faire les preuves.

Un modélè du plan projectif : pg(2,3)

Prenons l'exemple de $pg(2,3)$. Ce plan contient 13 points et 13 droites (voir figure 3.15) et peut être décrit en Coq comme suit :

```
Inductive ind_Point : Set := A | B | C | ... | K | L | M.

Inductive ind_line : Set := ABCD | AEFG | AIJM | AHKL | BEHI | BGJL
| BFKM | CELM | CFHJ | CGIK | DEJK | DGHM | DFIL.

Definition Incid_bool (P:Point) (l:Line) : bool := match P with
  | A => match l with
           | ABCD | AEFG | AIJM | AHKL => true
           | _ => false
         end
  [...]
end.
```

La description des modèles finis peut être facilement générée algorithmiquement en donnant pour chaque droite les points qu'elle porte. La relation d'incidence est alors dérivée automatiquement. La taille de la spécification de $pg(2,n)$ croît rapidement avec n, en effet $pg(2,n)$ a n^2+n+1 points et autant de droites.

Raisonnement par cas

Dans les modèles finis, toutes les propriétés sont démontrées en faisant du raisonnement par cas sur tous les points et toutes les droites apparaissant dans l'énoncé de la propriété. Pour que la preuve soit gérable par Coq, il faut faire preuve d'un peu de finesse pour éviter d'avoir trop de cas à traiter simultanément. Considérons par exemple, la preuve de l'axiome (A3P2) *Uniqueness* dans le cadre de $pg(2,3)$:

```
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.
```

Comme le plan $pg(2,3)$ contient 13 points et 13 droites, une analyse par cas exhaustive produit $13^4 = 28\ 051$ cas. Cela devient plus compliqué quand on aborde $pg(2,5)$ avec ses 31 points et ses 31 droites, où 923 521 cas doivent être traités. Pour un n donné, le plan projectif $pg(2,n)$ a $(n^2+n+1)^4$ cas à traiter, ce qui fait que la preuve n'est faisable que pour de petites valeurs de n.

Choix d'une description adaptée

Le choix de la description (synthétique ou combinatoire) influe sur la complexité des preuves à faire. En effet, deux descriptions différentes d'une même propriété mathématique (ici l'existence d'un point) n'ont pas le même nombre de quantificateurs.

```
Lemma point_existence : forall (l1 l2 :Line),
   exists A : Point, Incid A l1 /\ Incid A l2.


Lemma rk_inter : forall A B C D : Point,
   exists J, rk(triple A B J) = 2 /\ rk(triple C D J) = 2.
```

Dans le plan pg(2,3), la première formulation `point_existence` conduit à $13^2 = 169$ cas à traiter, pour chacun desquels il faut fournir un témoin. La deuxième formulation `rk_inter` conduit à $13^4 = 28\ 051$ cas à traiter. C'est une des raisons pour lesquelles l'approche combinatoire est moins bien adaptée pour démontrer qu'un ensemble de points et de droites muni d'une relation d'incidence vérifie bien les axiomes de la géométrie projective. Ce phénomène, bien connue dans le domaine des SAT solvers [AFG+11, Sut09] se retrouve ici en Coq. L'ordre des hypothèses et leur nombre ainsi que la taille des formules mises en jeu peuvent également avoir un impact significatif sur le temps de preuve.

Techniques de preuve

Une propriété comme celle d'unicité présentée ci-dessus est démontrée par induction sur les quatre variables suivantes : A, B, l et m. Il est pertinent de choisir de faire l'induction d'abord sur A, puis sur l, puis sur B, puis sur m. Dans ce cas, on tire plus rapidement avantage des contradictions dans les hypothèses. En effet, les variables A et l sont reliées dans l'hypothèse *Incid A l* et il est possible d'exploiter des contradictions dès que les valeurs A et l seront connues.

Dans les preuves avec de multiples traitements par cas, on utilise l'associativité à droite de la séquence dans les tactiques (`tac1;(tac2;tac3)`) ainsi que la tactique `abstract` pour décomposer et structurer la preuve.

3.5.2 Automatisation de la preuve de la propriété de Desargues

Toutes les techniques présentées jusqu'ici pour montrer que les plans considérés sont bien des modèles de la géométrie projective sont encore utilisables. Nous exploitons également des propriétés comme les symétries pour factoriser certains aspects de la preuve.

```
Lemma Desargues : forall O P Q R P' Q' R' X Y Z X' Y' Z'
X'' Y'' Z'' alpha beta gamma,
all_distinct O X Y Z X' Y' Z' X'' Y'' Z'' ->
rk(O,X,Y,Z)=2 -> rk(O,X',Y',Z')=2 -> rk(O,X'',Y'',Z'')=2 ->
rk(P,Q,gamma)=2 -> rk(P',Q',gamma)=2 -> rk(P,R,beta)=2 ->
rk(P',R',beta)=2 -> rk(Q,R,alpha)=2 -> rk(Q',R',alpha)=2 ->
rk(P,O,X,Y,Z)=2 -> rk(P',O,X,Y,Z)=2 ->
```

```
rk(Q,O,X',Y',Z')=2 -> rk(Q',O,X',Y',Z')=2 ->
rk(R,O,X'',Y'',Z'')=2 -> rk(R',O,X'',Y'',Z'')=2 ->
rk(O,P,P')=2 -> rk(O,Q,Q')=2 -> rk(O,R,R')=2 -> rk(O,P,Q)=3 ->
rk(O,P,R)=3 -> rk(O,Q,R)=3 -> rk(P,Q,R)=3 -> rk(P',Q',R')=3 ->
( rk(P,P')=2 \/ rk(Q,Q')=2 \/ rk(R,R')=2 ) ->
rk(alpha,beta,gamma)=2.
```

Nous montrons formellement que le plan $pg(2,2)$ est Désarguésien en suivant l'approche synthétique. Néanmois, pour démontrer un théorème comme celui de Desargues, il est clair que l'approche basée sur les rangs est plus efficace pour traiter les innombrables configurations à vérifier. Il s'agit maintenant de prendre en compte les symétries de la propriété de Desargues pour simplifier la preuve.

Automatisation basée sur des arguments géométriques

Nous prouvons le théorème dans le cas où le centre de perspective est un point particulier du modèle $pg(2,x)$. Il suffit alors de montrer que la permutation des points du modèle reste un modèle, cela évite une induction (celle sur le point perspective). Il est également possible de fixer les droites passant par le point perspective et qui forment les deux triangles. Il suffit ensuite de montrer que toute permutation de ces droites conserve la propriété. Finalement, les conditions de non-dégénérescence peuvent être exploitées pour réduire le nombre de cas à traiter, en imposant par exemple que les deux triangles en perspective aient au maximum un point en commun.

Ingénierie de la preuve

Du point de vue de la spécification, les rangs permettent de décrire de manière homogène toutes les incidences. Cela permet d'éviter les traitements par cas sur les droites (6 dans le cas du théorème de Desargues), sans pour autant augmenter le nombre de points.

De plus, les tactiques de simplifications sont faciles à écrire, ne manipulant qu'un seul type d'objets et uniquement les propriétés de l'égalité et de l'ordre sur les entiers. La plupart des formules s'écrivent sous la forme $rk(E) = n$ où E est un ensemble de points et n un entier naturel donnant intuitivement la dimension de l'ensemble.

Quand les preuves deviennent difficiles à traiter par Coq, il faut affiner précisement les tactiques construites afin de les rendre le plus efficace possible en remplaçant par exemple des appels à `omega` par l'application de lemmes, notamment pour le motif de raisonnement suivant :

$$\forall n, \forall E, rk\{E\} \leq n \rightarrow rk\{E\}) \geq n \rightarrow rk\{E\} = n.$$

Identifier les goulots d'étranglement dans l'exécution des tactiques se fait facilement grâce au profileur de tactiques `Ltac profiler` [TG15].

3.5.3 Représentation des données et techniques de preuve en 3D

Les techniques présentées jusqu'ici sont encore plus pertinentes quand on s'intéresse aux petits espaces projectifs $pg(3,2)$ [6] et $pg(3,3)$. L'espace $pg(3,2)$ contient 15 points et 35 droites. L'espace $pg(3,3)$ contient 40 points et 130 droites. De même que dans le plan, nous prouvons que les axiomes synthétiques de la géométrie projective sont bien vérifiés par les espaces $pg(3,2)$ et $pg(3,3)$. Néanmoins, cela est nettement plus difficile à démontrer. En effet, pour écrire et

6. Voir `http://demonstrations.wolfram.com/15PointProjectiveSpace/` pour une représentation interactive de $pg(3,2)$.

transmettre les preuves à Coq, nous rencontrons des limitations importantes pour ce qui concerne l'utilisation de la mémoire. Les tactiques utilisées doivent être particulièrement bien construites et la décomposition doit être clairvoyante et éviter d'avoir plusieurs milliers de millions de sous-buts à générer au même niveau. Considérons par exemple l'énoncé de l'axiome de *Pasch* dans $pg(3,2)$:

```
Lemma pasch : forall A B C D : Point, forall lAB lCD lAC lBD : Line,
              all_distinct A B 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).
```

Comme l'espace fini $pg(3,2)$ contient 15 points et 35 droites, les raisonnements par cas successifs conduisent à $15^4 \times 35^4 = 75\ 969\ 140\ 625$ cas à traiter. Il est donc essentiel de limiter la taille de la preuve en éliminant le plus de cas possibles le plus tôt possible. L'ordre dans lequel les raisonnements par cas sont faits ne suffit plus pour s'assurer que la preuve sera gérable par le système.

De nombreuses parties des preuves, démontrées automatiquement avec des tactiques sans intervention de l'utilisateur, doivent être factorisées dans des lemmes intermédiaires et la preuve complète doit être décomposée soigneusement avec des lemmes intermédiaires pertinents. Dans la preuve de la propriété de *Pasch*, nous utilisons le lemme intermédiaire suivant qui, étant donné deux points distincts T et Z, fournit la droite portant ces deux points. La fonction `l_from_points` calcule effectivement la droite qui passe par les points T et Z (et qui est unique quand $T \neq Z$).

Ici, le paradigme *proofs-as-programs* est bien mis en évidence. La fonction est décrite comme une version simplifiée (non-dépendante) de la propriété (A1P3) *Line-existence*, que l'on peut utiliser directement comme un programme. Cette fonction permet d'éviter de générer de nouvelles branches de preuve quand on fait un raisonnement par cas sur les droites. A chaque étape, une seule droite est correcte. De la même manière, un programme qui fournit les points appartenant à la droite l peut être extrait de la preuve du théorème (A4P3) *Three-Points*.

```
Lemma points_line : forall T Z : Point, forall x : Line,
Incid T x -> Incid Z x -> T<>Z -> x=(l_from_points(T,Z)).
```

Nous réduisons ainsi le nombre de cas total à vérifier à $15^4 = 50625$ cas, avant d'éliminer l'hypothèse existentielle de l'axiome de *Pasch* : `exists I:Point,(Incid I lAB /\ Incid I lCD)`.

Pour pg(3,2), nous avons décrit la spécification à la main. Néanmoins les preuves ont dû être adaptées partiellement afin que le système Coq puisse les traiter convenablement.

Pour pg(3,3), nous avons écrit un générateur de spécification qui, à partir de la liste, pour chaque droite, des points de cette droite, construit la relation d'incidence ainsi que tous les points ou droites nécessaires dans des quantifications existentielles. La construction longue de la spécification ainsi que toutes les étapes de recherche de preuve sont donc faites en amont de la preuve Coq. Dans ce cas, les spécifications et les preuves traitées par Coq frôlent les limites de capacité du système. Pour être acceptées par Coq, les preuves doivent être parfaitement optimisées, avec pour objectif d'éviter tout débordement de la capacité de la mémoire ou de la pile, ou bien, simplement que le temps de calcul ne soit trop long (plusieurs jours).

3.5.4 Résultats obtenus

Nous avons décrit formellement à l'aide du système Coq des plans et des espaces finis de plus en plus grands. Nous avons débuté par le plan de Fano $pg(2,2)$ et nous avons établi qu'il s'agit bien d'un plan projectif et qu'il vérifie bien la propriété de Desargues. Nous avons poursuivi le processus jusqu'au plan $pg(2,5)$. Nous avons ainsi validé la plus-value à utiliser la notion de rangs dans de telles preuves, notamment quand on cherche à établir la propriété de Desargues. Nous avons également étudié les espaces projectifs $pg(3,2)$ - qui contient 15 points et 35 droites - et $pg(3,3)$ - qui contient 40 points et 130 droites - . Nous avons prouvé que ces espaces respectent bien tous les deux les axiomes des espaces projectifs.

Cela représente environ 5 000 lignes de spécification et 2 500 lignes de preuve. Tous ces résultats sont synthétisés dans la figure 3.18. Pour chaque formalisation, nous présentons trois données-clés : le nombre de lignes de spécification, le nombre de lignes de preuve, et le temps nécessaire pour vérifier avec Coq (= compiler) la preuve.

Cela a nécessité des optimisations non seulement au niveau de la structure de la preuve (identification de symétries) mais aussi au niveau des pas élémentaires de raisonnement qui doivent être les plus efficaces possibles aussi bien en termes de temps de calcul et de consommation mémoire.

<table>
<tr><td></td><td colspan="6">Formalisation de la géométrie d'incidence projective</td></tr>
<tr><td></td><td colspan="3">avec la géométrie synthétique</td><td colspan="3">en utilisant les rangs</td></tr>
<tr><td></td><td>Spéc.</td><td>Preuve</td><td>T. E.</td><td>Spéc.</td><td>Preuve</td><td>T. E.</td></tr>
<tr><td>$pg(2,\,2)$ est un modèle</td><td>108</td><td>44</td><td>2s</td><td>110</td><td>42</td><td>16s</td></tr>
<tr><td>$pg(2,\,3)$ est un modèle</td><td>150</td><td>44</td><td>7s</td><td>297</td><td>77</td><td>2055s</td></tr>
<tr><td>$pg(2,\,4)$ est un modèle</td><td>206</td><td>44</td><td>35s</td><td colspan="3" rowspan="6">E. C.</td></tr>
<tr><td>$pg(2,\,5)$ est un modèle</td><td>276</td><td>44</td><td>90s</td></tr>
<tr><td>$pg(2,\,7)$ est un modèle</td><td>458</td><td>44</td><td>7337s</td></tr>
<tr><td>$pg(2,\,8)$ est un modèle</td><td colspan="3">E. C.</td></tr>
<tr><td>$pg(3,\,2)$ est un modèle</td><td>10490</td><td>192</td><td>1440s</td></tr>
<tr><td>$pg(3,\,3)$ est un modèle</td><td>420130</td><td>250</td><td>7623s</td></tr>
<tr><td><code>Desargues</code> vérifié pour $pg(2,\,2)$</td><td>188</td><td>205</td><td>37s</td><td>297</td><td>162</td><td>26s</td></tr>
<tr><td><code>Desargues</code> vérifié pour $pg(2,\,3)$</td><td colspan="3">E. C.</td><td>2089</td><td>386</td><td>10700s</td></tr>
<tr><td><code>Desargues</code> vérifié pour $pg(2,\,4)$</td><td colspan="6" rowspan="3">E. C.</td></tr>
<tr><td>...</td></tr>
<tr><td><code>Desargues</code> vérifié pour $pg(3,\,3)$</td></tr>
</table>

FIGURE 3.18 – Tests de performance pour plusieurs preuves en géométrie finie réalisés sur notre machine standard [8]. E. C. signifie *Explosion Combinatoire*.

Ces expériences ont permis de tester exhaustivement les capacités de preuve du système Coq. Elles ont notamment permis de détecter des régressions dans la capacité de Coq (au passage de 8.5 à 8.6) à faire des preuves et à en vérifier l'exactitude par *type-checking* au moment du `Qed`. Ces régressions ont été corrigées depuis.

Pour chaque avancée d'un ordre de magnitude dans les plans et espaces que nous arrivons à traiter, nous avons dû mettre en place des optimisations de plus en plus sophistiquées. Le prochain objectif serait de traiter quelques uns des plans de Hall qui contiennent 91 points et 91 droites [Hal43].

A titre de comparaison, notamment pour la preuve du théorème de Desargues, nous avons mené des expériences avec l'infrastructure TPTP framework [Sut09]. Pour le plan de Fano $pg(2,2)$, seuls 3 prouveurs parviennent à prouver le théorème de Desargues (en moins de 300

secondes). Il s'agit des prouveurs iprover, Vampire [KV13] et Z3 [dMB08]. Nous prouvons le théorème de Desargues dans Coq en 37 secondes pour $pg(2,2)$. La performance est donc comparable, même si nous avons dû écrire la structure de la preuve et les tactiques démontrant le théorème à la main. L'avantage de notre approche reste que nous fournissons en plus de l'affirmation que le théorème est vérifiée, une preuve formelle qui peut être revérifiée par Coq ou un autre vérificateur de preuve comme Dedukti [ABC+16], où des preuves Coq peuvent être importées en utilisant l'outil CoqInE [BB12].

3.6 Mise en œuvre d'un prouveur automatique et interaction avec Coq

La notion de rang se prête bien par son côté combinatoire à la preuve automatique par recherche exhaustive d'un chemin des hypothèses vers la conclusion d'un théorème. Tout ensemble de points de l'espace projectif de dimension 3 a un rang compris entre 0 (ensemble vide) et 4 (l'espace tout entier). L'idée est de représenter un ensemble (par son indicatrice binaire) ainsi que ses rangs minimum et maximum. On utilise pour cela un entier machine 32 bits. Cela permet d'encoder 28 points distincts avec un bit par point, les 4 autres bits servent à encoder le rang minumum et le rang maximum de l'ensemble considéré. A partir d'une configuration initiale (on suppose que le rang de chaque sous-ensemble est compris entre 0 et 4 et on spécialise en fonction des hypothèses du théorème), on applique systématiquement les règles du matroïde associé aux rangs, afin de réduire l'intervalle de valeurs possibles pour le rang de chaque sous-ensemble, et on itére jusqu'à ce que le système se stabilise.

Cet outil automatique a permis de générer automatiquement des scripts de preuve Coq pour démontrer formellement des théorèmes mettant en jeu un nombre de plus en plus important de points distincts : le théorème de Desargues en 3D (10 points), le théorème du conjugué harmonique (14 points) ainsi que la preuve de l'implication de la propriété de Pappus vers la propriété de Dandelin-Gallucci (19 points).

Ces travaux sont en cours de soumission dans une revue internationale [BMS20] et ont été présentés par David Braun dans sa thèse [Bra19] soutenue en septembre 2019.

3.6.1 Principe général

L'algorithme prend en entrée un énoncé géométrique (la conclusion du théorème) et des hypothèses. Tous sont exprimés sous forme d'égalités sur les rangs : rk{P,Q,R...}=n. L'algorithme encode ces propriétés sur les ensembles et les rangs dans un treillis d'inclusion avec des données incomplètes pour les rangs minimum et maximum. Dans le pire des cas, si aucune information n'est connue, le rang est compris entre 0 et 4 (en 3D).

On lance ensuite un algorithme de saturation sur ce treillis. Cet algorithme applique les propriétés de base des matroïdes (voir figure 3.5) sous forme de règles qui peuvent réduire la taille des intervalles bornant le rang de chaque sous-ensemble de points de la configuration initiale. Chaque fois que le rang d'un sous-ensemble est modifié, le treillis est mis à jour en conséquence et les nouvelles contraintes sont alors propagées vers les autres sous-ensembles de points.

Le prouveur poursuit ses itérations jusqu'à ce que l'intervalle de valeurs (entre le rang minimum et le rang maximum) ne puisse plus être réduit pour aucun sous-ensemble de points de l'ensemble initial E. Le processus de saturation s'arrête alors. Si l'énoncé était faux, le prouveur atteint un état incohérent avec $rkMin > rkMax$ pour un ensemble de points donné. Si le résultat attendu n'est pas obtenu, cela signifie que le prouveur a échoué. Sinon il reste à vérifier que la preuve obtenue est bien correcte.

Comme chaque application d'une règle de réécriture est enregistrée avec les sous-ensembles X et Y correspondants, le prouveur reconstruit alors la liste des déductions effectuées pour atteindre l'état final et la transforme en un script de preuve Coq. Ce script est envoyé à Coq qui vérifie que la démonstration est bien correcte et valide (ou non) la preuve proposée.

Le processus complet est synthétisé dans la figure 3.22. Il est important de noter que la phase de saturation et la phase d'enregistrement de la trace se déroulent en parallèle.

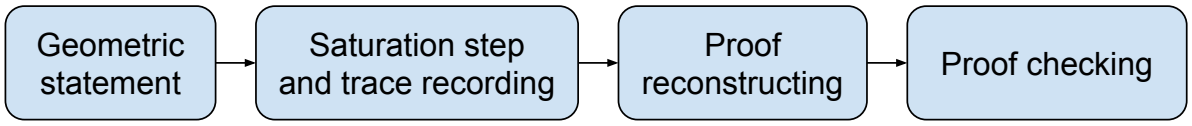


FIGURE 3.19 – Le prouveur procède en quatre phases successives.

Le prouveur est construit pour générer à la fois des configurations en géométrie affine et en géométrie projective. En effet, son moteur ne s'appuie que sur les propriétés de base des matroïdes, qui sont indépendantes des propriétés géométriques ajoutées pour décrire la géométrie projective. La distinction entre les deux cadres repose uniquement sur l'existence (ou non) d'un point d'intersection entre deux droites coplanaires. Ainsi, toutes les déductions faites par le prouveur le sont dans le cadre de la géométrie d'incidence *pure* et les résultats sont corrects à la fois en géométrie projective et en géométrie affine.

Notre prototype de prouveur ne crée pas de nouveaux points au cours de ses étapes de démonstration. Il est en effet plus raisonnable que de nouveaux points soient créés uniquement sous le contrôle de l'utilisateur afin d'éviter une éventuelle explosition combinatoire ou bien l'apparition de boucles infinies dans l'algorithme.

3.6.2 Règles de réécriture

Les déductions faites par le prouveur s'appuyent uniquement sur l'application des règles de matroïdes vérifiées par la fonction de rang et présentées dans la figure 3.5. Les autres axiomes permettent uniquement de créer de nouveaux points ou bien de borner la dimension. Comme notre prouveur ne gère pas la création de nouveaux points, nous ignorons tous les axiomes avec une quantification existentielle, notamment (A6R3) et (A7R3).

La propriété **(A1R2-R3)** sert à l'initialisation du treillis. Nous transformons les propriétés **(A2R2-R3)** et **(A3R2-R3)** en 8 propriétés sur le rang minimum et le rang maximum. Ces propriétés permettent de maximiser le rang minimum et de minimiser le rang maximum à chaque étape.

Soient X et Y deux sous-ensembles de points de l'ensemble E de tous les points, les ensembles X, Y, $X \cup Y$ and $X \cap Y$ doivent obligatoirement vérifier les propriétés énoncées dans la figure 3.20.

Dans l'algorithme de saturation 3.1 , ces propriétés sont transformées en règles de réécriture, qui sont utilisées pour mettre à jour l'intervalle des rangs possibles pour les ensembles de points considérés X et Y. Chaque règle de réécriture de la figure 3.21 est liée à la propriété de même numéro dans la figure 3.20. Cet ensemble de règles peut bien sûr être étendu avec de nouvelles règles. Il pourrait s'agir des règles transcrivant la propriété de Pappus, ou bien de règles accélérant la réduction des intervalles en exploitant la propriété que l'intersection de deux plans et une droite.

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

FIGURE 3.20 – Propriétés utilisées jusqu'à saturation.

(RS1) **if** $X \subseteq Y$ and $rkMin(X) > rkMin(Y)$ **then** $rkMin(Y) \leftarrow rkMin(X)$

(RS2) **if** $Y \subseteq X$ and $rkMin(Y) > rkMin(X)$ **then** $rkMin(X) \leftarrow rkMin(Y)$

(RS3) **if** $X \subseteq Y$ and $rkMax(Y) < rkMax(X)$ **then** $rkMax(X) \leftarrow rkMax(Y)$

(RS4) **if** $Y \subseteq X$ and $rkMax(X) < rkMax(Y)$ **then** $rkMax(Y) \leftarrow rkMax(X)$

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

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

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

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

FIGURE 3.21 – Règles de réécriture utilisées pour réduire l'intervalle des valeurs acceptables pour le rang.

3.6.3 Implémentation effective du prouveur

Dans cette section, nous présentons quelques caractéristiques importantes de l'implantation du prouveur automatique en C. Nous commençons par présenter la structure de données utilisée. Nous décrivons ensuite les principales subtilités de l'algorithme. Enfin, nous expliquons comment le script de preuve Coq est généré. Etant donné la complexité en temps et en mémoire de l'algorithme de saturation, nous avons choisi délibérément d'implanter notre prototype de prouveur comme un programme indépendant plutôt que de l'intégrer directement à l'assistant de preuve Coq.

Initialisation de l'algorithme

Les données fournies en entrée à l'algorithme sont la configuration géométrique exprimée sous forme d'égalités sur les rangs ainsi que le nombre de points total n de la configuration. La première étape est de construire l'ensemble des parties (de taille 2^n) dont chaque élément représente un sous-ensemble de points de la configuration géométrique initiale.

Nous encodons chacun de ces sous-ensembles dans un mot de 32 bits où les 28 premiers bits servent à représenter la fonction caractéristique de l'ensemble des points de la configuration initiale. Les points sont encodés dans l'ordre lexicographique.

Dans l'exemple de la figure 3.22, les bits 1, 2, 5, et 8 sont à 1, et ils représentent les points A, B, E, et H. Les 4 derniers bits représentent le rang minimum (2 bits) et le rang maximum (2 bits) de cet ensemble de points. La fonction de rang ne prenant que 4 valeurs différentes (de 1 à 4) pour les ensembles non-vides, nous pouvons encoder sur 2 bits chacun les rangs minimum et maximum. En utilisant les 4 valeurs 00, 01, 10 et 11, notre structure de données contient alors la valeur $\mathsf{rk} - 1$.

Notre prototype ne traite donc que des configurations dont le nombre de points est inférieur à 28, ce qui s'avère largement suffisant en pratique. Néanmoins, on pourrait sans difficulté adapter cette représentation et utiliser des mots de 64 bits. Dans ce cas, nous pourrons avoir des configurations géométriques d'au maximum 60 points.

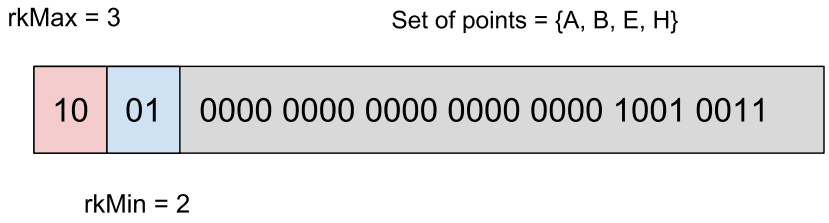


FIGURE 3.22 – Représentation en mémoire d'un (sous-)ensemble de points et de son rang.

Au départ, nous utilisons les axiomes **(A1R2-R3)** et **(A9R3)** pour borner le rang de chaque sous-ensemble non-vide X de E avec la valeur 1 pour le rang minimum et $min\{|X|, 4\}$ pour le rang maximum. Ces notions de rang minimum et maximum sont ensuite affinées grâce aux hypothèses de l'énoncé géométrique.

Boucle de saturation

Une étape de saturation applique toutes les règles de réécriture de la figure 3.21 à toutes les sous-parties de l'ensemble initial de points. Les règles sont appliquées successivement en utilisant l'ordre arbitraire pré-déterminé suivant RS1 RS3 RS2 RS4 RS5 RS7 RS8 [9]

Pour chaque sous-ensemble, on vérifie que les valeurs extrêmes pour le rang sont cohérentes avec les règles. Par exemple, considérons $X = \{A, B\}$ et $Y = \{A, B, C\}$ avec l'hypothèse que $rkMin(X) \geq 2$. L'étape d'initialisation conduit aux bornes suivantes : $rkMin(Y) \geq 1$ et $rkMax(Y) \leq 3$. Ces bornes sont en contradiction avec la propriété **(PS1)** pour les ensembles X et Y. En effet, le rang minimum de Y est inférieur au rang minimum de X alors que l'on a l'inclusion $X \subseteq Y$. Par conséquent, le rang minimum de Y est mis à jour pour être plus grand que celui de X conformément à l'exigence exprimée par la règle **(RS1)**.

L'algorithme 3.1 présenté ci-dessous applique tous les règles de réécriture à chaque couple de sous-ensembles de points. Si une règle n'est pas vérifiée, la valeur du rang minimum ou maximum à l'origine du conflit est corrigée. Pour garantir la correction de l'algorithme, nous imposons que les seules mises à jour possibles consistent à augmenter la valeur du rang minimum ou à réduire la valeur du rang maximum tout en s'assurant que l'invariant $RkMin < RkMax$ est conservé.

Dans le cas où les rangs minimum et maximum sont égaux, nous avons trouvé le rang effectif pour ce sous-ensemble et il ne sera plus modifié. Si la valeur du rang minimum devient strictement plus grande que celle du rang maximum, nous sommes en présence d'hypothèses contradictoires et l'algorithme échoue. Dans tous les autres cas, l'algorithme répète la boucle de saturation

9. Nous avons vérifié expériementalement que l'ordre d'application des règles n'a pas d'impact significatif sur le temps de calcul et ne modifie pas la complexité en temps de l'algorithme.

et applique de nouveau les règles à tous les sous-ensembles jusqu'à ce qu'aucune mise à jour n'ait été possible lors de la dernière itération. Quand l'algorithme s'arrête, toutes les déductions possibles ont été faites.

Algorithme 3.1 : 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
```

Enregistrement des déductions effectuées

En parallèle de la phase de saturation, le prouveur enregistre chacune des déductions effectuées afin de pouvoir produire une trace de la preuve qui sera ensuite transmise pour validation à l'assistant de preuve Coq.

Nous construisons un graphe orienté acyclique appelé graphe des déductions (GD). Chaque nœud de ce graphe représente un sous-ensemble avec ses rangs minimum et maximum ainsi que la règle ayant servi à établir les valeurs de ces rangs. Initialement, nous avons un nœud pour chaque sous-ensemble et les valeurs des rangs minimum et maximum sont ceux fournis par les hypothèses. L'application d'une règle conduit à la création d'un nouveau nœud avec des indications de rang plus précises. Ce nouveau nœud est connecté aux nœuds déjà existants ayant permis de faire la nouvelle déduction. Chaque sous-ensemble ne peut apparaître au maximum que 4 fois dans le graphe. En effet, la distance entre rang minimum et rang maximum est d'au plus 3 et à chaque création d'un nouveau nœud cette distance diminue de 1.

La figure 3.23 donne un extrait du graphe de déductions dans le cas où la règle RS5 est appliquée. Géométriquement cette étape de raisonnement exprime que si l'intersection de 2 droites est un point, alors l'ensemble formé par l'union des points de ces deux droites est au plus un plan. L'algorithme de saturation 3.1 est alors adapté en l'algorithme 3.2 pour prendre en compte la construction du graphe de déductions en parallèle de la saturation.

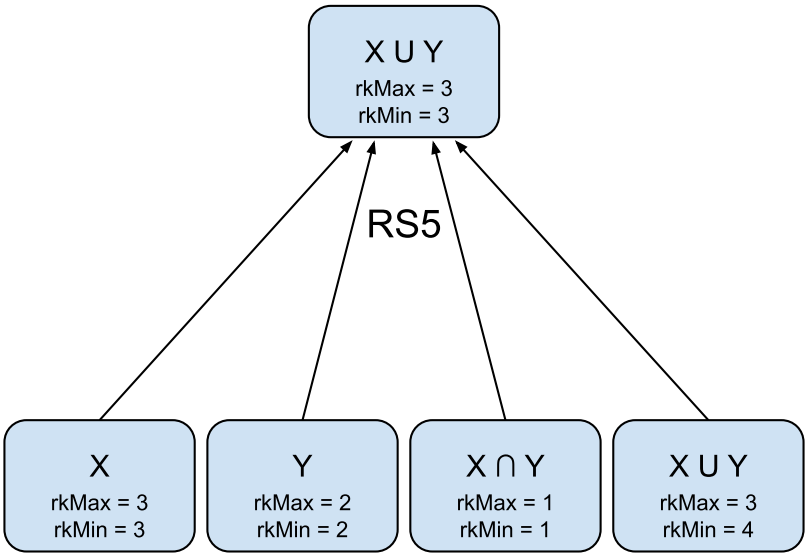


FIGURE 3.23 – Un extrait du graphe de déductions et l'application de la règle RS5.

Algorithme 3.2 : Étape de saturation et construction du Graphe de déductions.

Entrée(s) : Graphe de déductions initialisé
Sortie(s) : Graphe de déductions 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 nœud dans le Graphe de déductions mémorisant
8                      l'application de 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
```

Génération d'un script Coq vérifiable

L'algorithme procède via un parcours récursif postfixe du graphe de déductions, en débutant par le nœud représentant la conclusion de l'énoncé à prouver. Tous les nœuds du graphe ne sont pas utiles pour reconstruire une trace de la preuve. De plus la trace n'est pas unique et dépend notamment de l'ordre dans lequel les nœuds sont traités ainsi que de l'ordre d'application des règles dans la phase de saturation. Pour des raisons d'efficacité, nous ajoutons un mécanisme d'étiquetage des nœuds afin de ne parcourir que ceux qui sont utiles à la preuve et afin également de ne parcourir ces nœuds qu'une seule fois. Nous invitons le lecteur intéressé à consulter la thèse de David Braun [Bra19] pour plus de détails.

Chaque règle de réécriture de la figure 3.21 est traduite en une séquence de tactiques Coq. Ce code est englobé dans un bloc (voir figure 3.1) qui regroupe toutes les étapes de preuve nécessaires

pour obtenir les (nouvelles) valeurs des rangs minimum et maximum du nœud reconstruit. On vérifie à ce moment-là que toutes les hypothèses nécessaires pour appliquer la règle conduisant au nouveau nœud ont bien déjà été prouvées et se trouvent dans le contexte.

Les blocs, au sens Coq du terme, implantent une forme de localité, dans le style des espaces de noms en C/C++ : ils permettent de contrôler la portée de noms d'hypothèses, d'éviter les collisions de noms et de simplifier le déroulement des démonstrations en allégeant le contexte. Les hypothèses locales à un bloc disparaissent systématiquement à la fin de celui-ci. Seules les hypothèses initiales du théorème persistent.

```
(* 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 3.1 – Illustration d'un bloc correspondant à l'application d'une règle.

Les blocs sont assemblés en un fichier Coq, contenant l'énoncé formel du théorème, suivi d'un script de preuve démontrant le théorème. Une fois ce fichier validé (= compilé avec succès) par Coq, nous disposons d'une preuve formelle du théorème en question.

3.6.4 Fonctionnement du prouveur sur un exemple simple

Nous détaillons le fonctionnement du prouveur sur un théorème de la géométrie projective. Il s'agit d'un théorème très simple, avec seulement 4 points. Cela va nous permettre de visualiser tous les sous-ensembles et les étapes de preuve réalisées automatiquement par le prouveur.

Théorème 4 (Un exemple simple)**.** *Considérons un plan ABD et construisons un nouveau point C distinct de A et de D sur la droite AD. L'ensemble des points A, B et C forment alors un plan, i.e. rk{A,B,C}=3.*

L'énoncé exact de ce théorème en Coq est donné dans la figure 3.24 et une description géométrique est donnée dans la figure 3.25.

L'énoncé comportant 4 points, la première couche du graphe de déduction contient $2^4 - 1$ nœuds, décrivant un sous-ensemble non vide et les valeurs de ses rangs maximum et minimum, comme le montre la figure 3.26.

Nous procédons à une première itération du processus de saturation, ce qui donne le graphe de déductions [10] de la figure 3.27. Notons que dans le cas où plusieurs règles sont applicables

10. Les étiquettes adjacentes aux nœuds indiquent l'ordre dans lequel les nœuds ont été créés.

```
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.
```

FIGURE 3.24 – Un exemple d'énoncé géométrique pour notre prouveur.

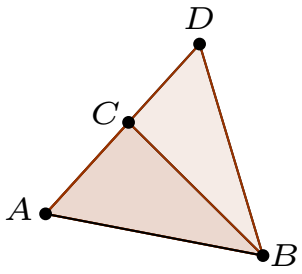


FIGURE 3.25 – La configuration géométrique du théorème 3.24.

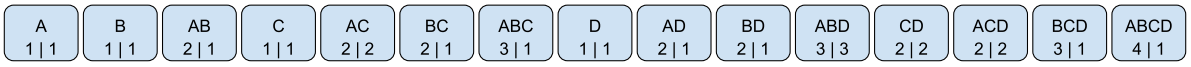


FIGURE 3.26 – Le graphe de déduction après initialisation avec les hypothèses.

pendant la même étape de saturation, plusieurs nœuds peuvent être créés pour le même sous-ensemble (par exemple ici pour les ensembles BCD et ABCD).

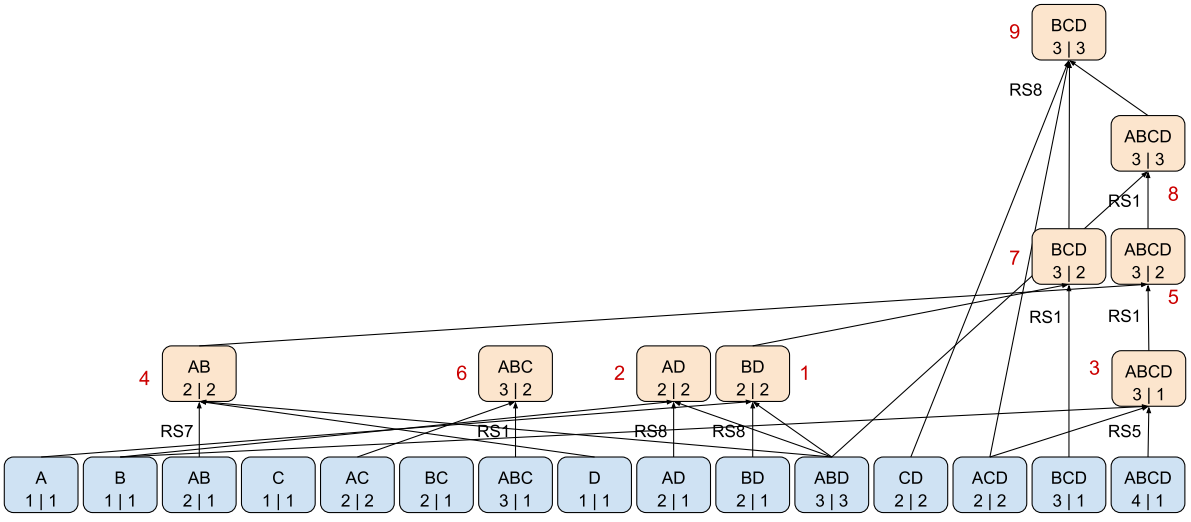


FIGURE 3.27 – Graphe de déductions partiellement saturé.

Une seconde phase de saturation est ensuite effectuée par l'algorithme et conduit au graphe de déductions de la figure 3.28. De nouveaux nœuds (pour les sous-ensembles ABC et BC) sont

construits. La troisième itération de la saturation ne fait plus aucune modification et l'algorithme s'arrête. Dans cet exemple, les rangs minimum et maximum sont égaux pour tous les sous-ensembles de points. En effet, le problème est bien contraint. Quand le problème est sous-contraint, l'algorithme ne peut pas trouver un rang exact pour tous les sous-ensembles.

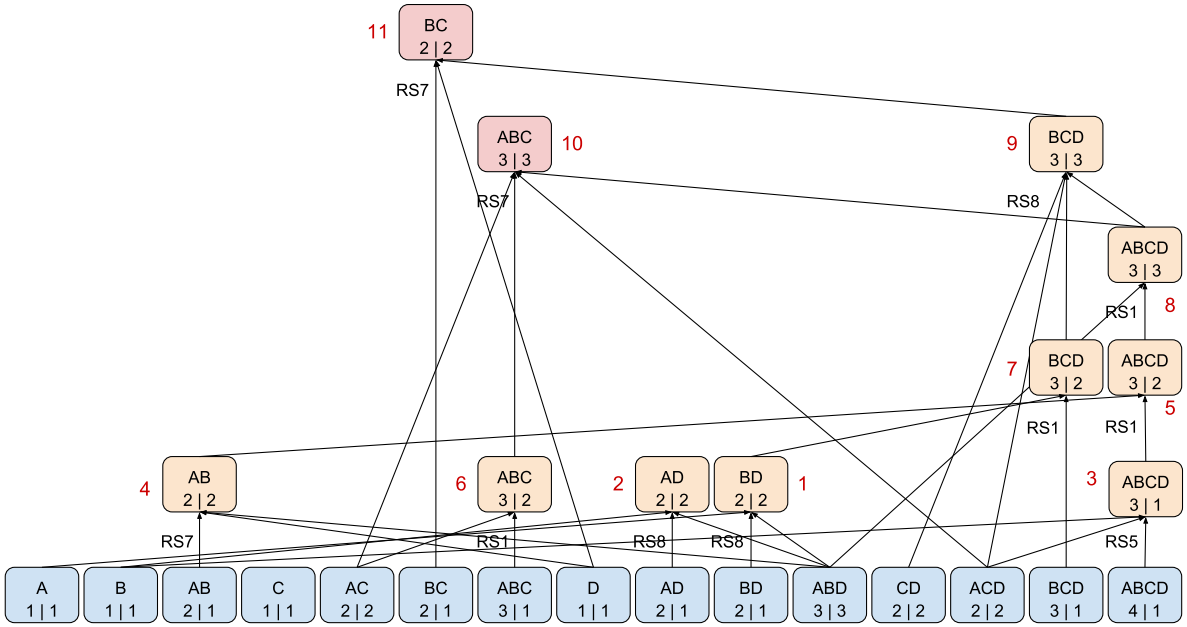


FIGURE 3.28 – Graphe de déductions complètement saturé.

Une fois la saturation terminée, le prouveur effectue un parcours récursif postfixe en partant du nœud dont on cherchait le rang. Dans notre cas, il s'agit du nœud correspond à l'ensemble ABC, numéroté 14 dans la figure 3.29. Cela permet de générer la trace de déduction, qui est constituée de tous les nœuds numérotés en bleu dans cette figure. Cette trace est ensuite convertie en un script de preuve Coq, où chaque étape de la saturation est exprimée en tactiques Coq. Enfin, le script de preuve obtenu est exécuté par Coq afin de vérifier la correction de la preuve proposée.

Bien que nous soyons capables de faire valider par Coq des scripts de plusieurs milliers de lignes avec des centaines d'hypothèses, il est souvent nécessaire de savoir structurer en couches les différents niveaux de raisonnement, de décomposer automatiquement une preuve en plusieurs lemmes intermédiaires et de les nommer convenablement. Cela permet d'obtenir des preuves plus simples et aussi de réutiliser *gratuitement* certains lemmes déjà prouvés. Le lecteur intéressé pourra trouver une description plus détaillée de l'outil de preuve automatique dans la thèse de David Braun [Bra19].

3.7 Preuves automatiques de théorèmes classiques en géométrie projective

Dans cette section, nous présentons trois résultats emblématiques de la géométrie projective, que nous avons pu démontrer automatiquement avec notre prouveur. Il s'agit du theorème de Desargues, du théorème du conjugué harmonique et du théorème de Dandelin-Gallucci.

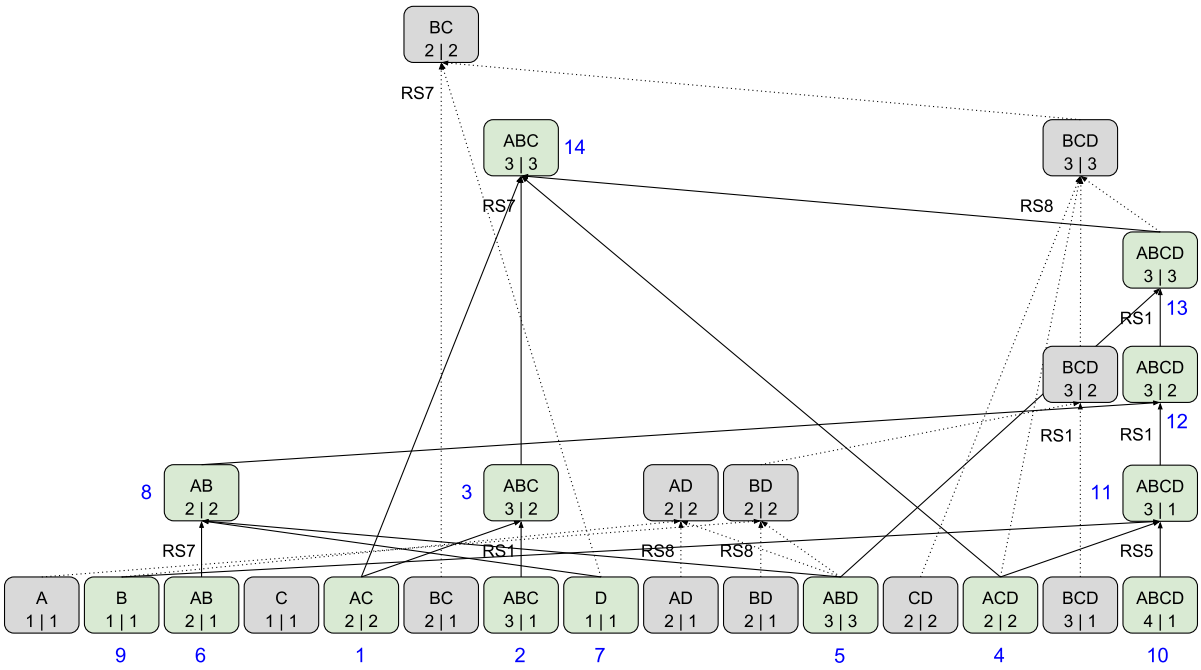


FIGURE 3.29 – Trace de la reconstruction de la preuve dans le graphe de déductions.

3.7.1 Théorème de Desargues

Le premier exemple de preuve complètement automatisée est le théorème de Desargues. La propriété de Desargues n'est pas toujours vérifiée en deux dimensions. On peut en effet construire des plans projectifs particuliers comme le plan de Moulton [Mou02], dans lesquels cette propriété n'est pas vérifiée. Néanmoins, dès que l'on se place dans un espace de dimension au moins 3, cette propriété est vérifiée.

Preuve de la version 3D du théorème de Desargues

Théorème 5 (Théorème de Desargues)**.** *Soit E un espace projectif et P, Q, R, P', Q', R' des points de cet espace. On considére les triangles PQR and $P'Q'R'$, que l'on suppose non dégénérés. Si les droites (PP'), (QQ') et (RR') se coupent en un point perspective O, alors les points $\alpha = (PR) \cap (P'R')$, $\beta = (QR) \cap (Q'R')$ et $\gamma = (PQ) \cap (P'Q')$ sont alignés.*

Comme nous l'avons décrit dans la section 3.3.5 de ce document, nous avons prouvé ce théorème formellement avec Coq de manière interactive il y a quelques années [MNS12]. En résumé, la démonstration se déroule ainsi : on traite d'abord la propriété dans le cas où les 2 triangles en question sont non coplanaires. Cette configuration est appelée Desargues 3D (voir Fig. 3.30). A partir de cette preuve en 3D, nous montrons ensuite que cette propriété est vérifiée quand les 2 triangles sont coplanaires (voir Fig. 3.31). Pour cela, on se ramène du cas 2D au cas 3D en relevant le triangle $P'Q'R'$ vers le triangle $P''Q''R''$, qui est non coplanaire avec le triangle PQR et néamoins perspective depuis le point O situé sur la droite SO'.

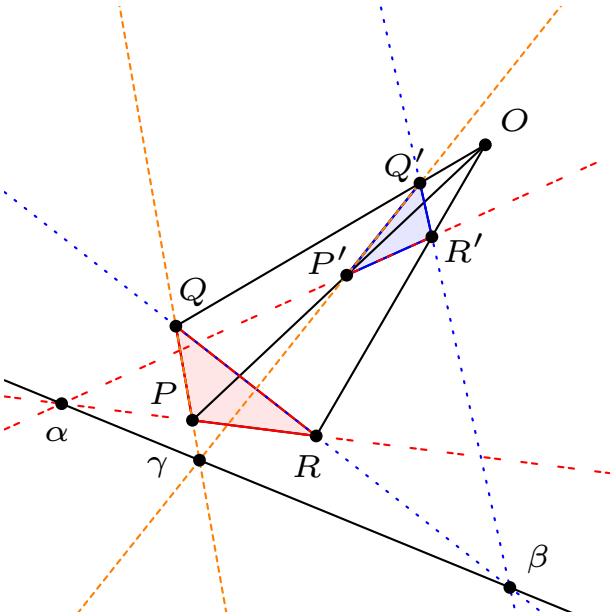


FIGURE 3.30 – Le théorème de Desargues (5), dans sa version 3D.

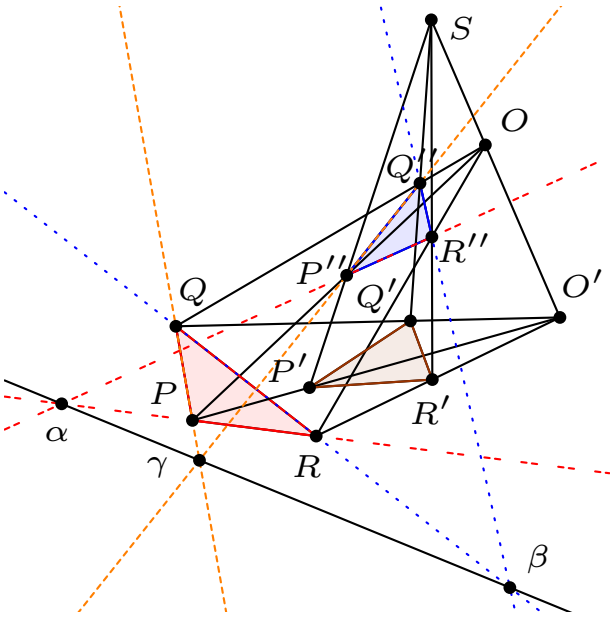


FIGURE 3.31 – La version 2D du théorème de Desargues (5) et son relèvement en 3D.

Résultats obtenus

Les énoncés des deux versions du théorème (2D et 3D) sont prouvés rapidement et automatiquement par notre prouveur. La version la plus simple (3D) compte 10 points, alors que la version 2D incluant le relèvement en 3D compte 15 points. Prouver l'énoncé `Desargues_2D` prend environ 3min (2min pour générer le script de preuve Coq et 1min pour vérifier les 6 000 lignes de preuve générées) et le niveau d'utilisation de la mémoire reste raisonnable.

```
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.
```

FIGURE 3.32 – Enoncé du théorème de Desargues en 3D (5).

```
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.
```

FIGURE 3.33 – Enoncé en Coq du théorème de Desargues avec l'extrusion d'un des triangles pour obtenir une figure 3D.

3.7.2 Conjugué harmonique

Le deuxième exemple que nous considérons est celui du conjugué harmonique. L'énoncé comporte 14 points et est donc similaire en complexité avec la propriété de Desargues en 2D.

Dans un plan euclidien, cette configuration géométrique s'intéresse au birapport entre quatre points disposés sur une même droite.

Définition 8. *(Birapport) Soient A, B, C, D 4 points appartenant à la même droite, le birapport r de (A, B) et (C, D) est le rapport des mesures algébriques :* $r = \frac{\frac{\overline{CA}}{\overline{CB}}}{\frac{\overline{DA}}{\overline{DB}}}$.

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.

En géométrie projective, le conjugué harmonique se définit ainsi :

Théorème 6 (Théorème du conjugué harmonique)**.** *Soit E un espace projectif. Soient A, B, C 3 points collinéaires de cet espace. Soit R un point quelconque, en dehors de la droite AB. On considère une droite passant par C et coupant RA (resp. RB) en Q (resp. P). Si les droites AP et BQ se coupent en U et si RU coupe AB en D, alors D ne dépend ni du choix de R, ni de celui de la droite CPQ. Le point D est appelé le conjugué harmonique de C par rapport à A et B.*

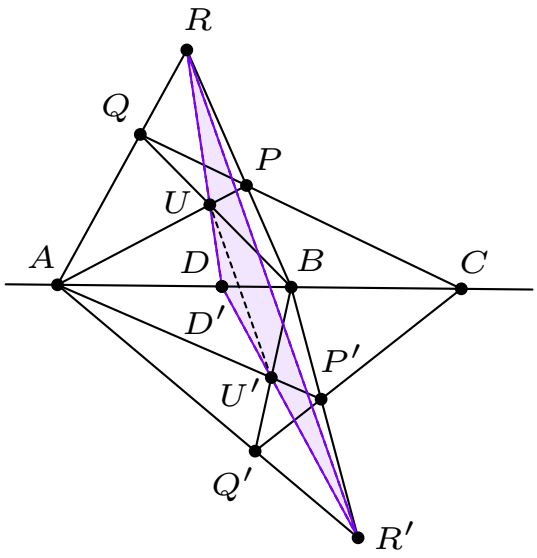


FIGURE 3.34 – Une configuration géométrique du conjugué harmonique (Théorème 6).

```
Lemma harmonic_conjugate : forall A B C P Q R P' Q' R' U U' D D' O : Point
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.
```

FIGURE 3.35 – Enoncé du théorème du conjugué harmonique (6).

Résultats obtenus

La complexité de cette preuve est du même ordre de grandeur que pour la version 2D du théorème de Desargues. En effet, cette configuration met également en jeu 14 points, en incluant les points auxiliaires, hors du plan initial. Le script de preuve Coq généré par le prouver est d'environ 10 000 lignes et l'utilisation de la mémoire reste inférieure à 2GB (12% de la mémoire totale disponible sur notre machine test équipée de 16 GB). Avec toutes les optimisations, notamment le mécanisme de couches d'abstraction facilitant la vérification de la preuve par Coq, la vérification de la preuve par Coq se fait en moins d'une minute.

3.7.3 Théorème de Dandelin-Gallucci

Pour notre troisième exemple, nous nous intéressons à une propriété peu connue de la géométrie projective en 3D la propriété de Dandelin-Gallucci. Cette configuration met en jeu plusieurs droites transversales non coplanaires.

Définition 9 (Droite transversale)**.** *Une droite qui coupe un ensemble de droites est dite transversale pour cet ensemble.*

Quand on prend trois droites quelconques en dimension 3, une telle droite existe et cela provient directement de l'axiome (A6P3) **Upper-Dimension** (et de son homologue exprimé en termes de rangs (A9R3) **Rk-Upper-Dimension**). Notons qu'en dimension supérieure à 3, il n'existe pas toujours une droite transversale à un ensemble de plus de 2 droites. La propriété de Dandelin-Gallucci s'exprime de la manière suivante :

Propriété 2 (Propriété de Dandelin-Gallucci)**.** *Considérons 3 droites quelconques a, b, c qui ne se coupent pas 2 à 2 et qui coupent trois autres droites e, f, g. Toute droite transversale d au premier ensemble $\{a, b, c\}$ coupe toute droite transversale h du deuxième ensemble $\{e, f, g\}$.*

Le théorème de Dandelin-Gallucci établit ce résultat comme un théorème et est connu depuis longtemps du point de vue mathématique. Les démonstrations présentes de la littérature utilisent le concept de birapport, le théorème de Desargues ou la propriété de Pappus [BP15]. Nous nous intéressons à une preuve basée sur la propriété de Pappus et présentée par H. F. Baker dans *Principles of Geometry* [Hen25].

Dans notre contexte, le théorème de Dandelin-Gallucci exprime que la propriété de Pappus et la propriété de Dandelin-Gallucci sont équivalentes.

Preuve de la propriété de Dandelin-Gallucci (en supposant Pappus)

On montre, en supposant que la propriété de Pappus est vérifiée, que la propriété de Dandelin-Gallucci est vérifiée.

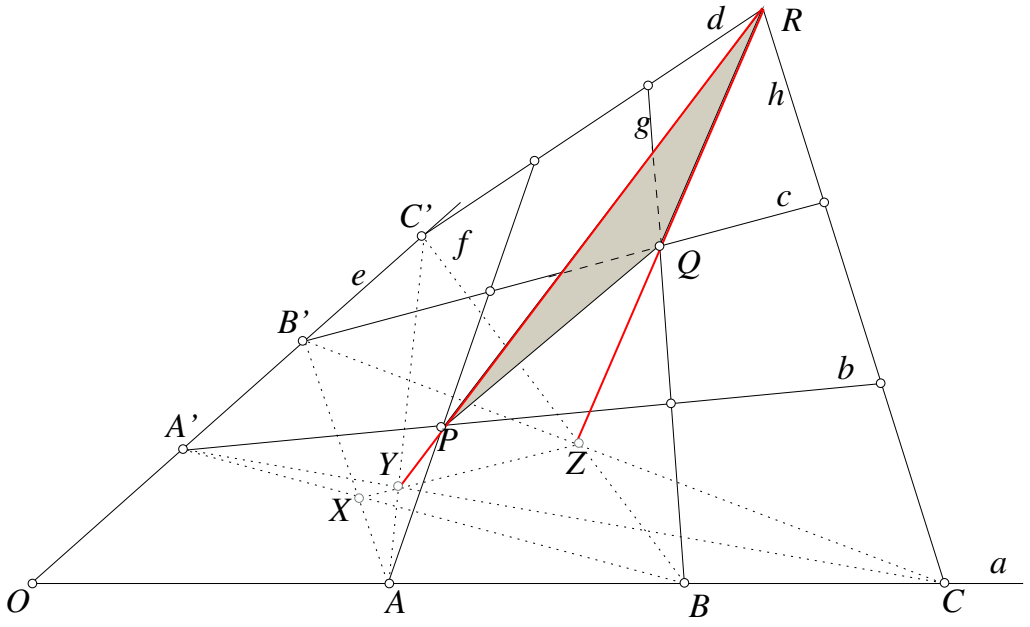


FIGURE 3.36 – De la configuration de Pappus à celle de Dandelin-Gallucci .

Prouver ce théorème nécessite de construire manuellement les points auxiliaires de la configuration de Pappus à utiliser : $\alpha = (NL') \cap (N'L)$, $\beta = (M'L) \cap (ML')$ et $\gamma = (NM') \cap (N'M)$. Ce travail difficile à automatiser est de la responsabilité de l'utilisateur. Le système peut ensuite automatiquement identifier cette configuration de Pappus et en déduire l'alignement recherché afin de pouvoir terminer la preuve du théorème de Dandelin-Gallucci.

En effet, le prouveur que nous avons construit offre la possibilité d'ajouter des axiomes sans quantification existentielle sous la forme de règles de réécriture supplémentaires dans notre algorithme de saturation. Dès qu'une configuration de Pappus est identifiée, le prouveur ajoute, comme nouveau nœud dans le graphe de déduction, le fait que les points α, β et γ sont alignés (c'est-à-dire que rk$\{\alpha, \beta, \gamma\}$=2).

```
Lemma dg : forall A A' B B' C C' R P Q N M L N' M' L' O alpha beta gamma: Point,
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 -> /* first point Pappus */
rk(M', L, beta) = 2 -> rk(M, L', beta) = 2 -> /* second point Pappus */
rk(N, M', gamma) = 2 -> rk(N', M, gamma) = 2 -> /* third point Pappus */

rk(A, C', Q, M') = 2. /* M' begongs to b' */
```

FIGURE 3.37 – Enoncé en Coq du théorème 2.

Résultats obtenus

Il s'agit du théorème le plus difficile que notre prouveur arrive à démontrer. En effet, il met en jeu 19 points (16 pour la configuration géométrique initiale et 3 points supplémentaires correspondant à l'application de la propriété de Pappus). Le prouveur met 16h pour construire le graphe de déduction complet et ainsi obtenir un script de preuve, structurée par couches de déduction, que Coq peut valider. Pour atteindre ce résultat, de nombreuses autres optimisations doivent être mises en œuvre. Elles sont décrites en détail dans la thèse de David Braun [Bra19].

Une des optimisations les plus significatives consiste à réaliser l'étape de saturation par couches. Pour la preuve du théorème de Dandelin-Gallucci, on utilise 3 couches successives. La première comporte 9 points. Il s'agit de toutes les intersections des droites a, b, c et e, f, g. La deuxième inclut les points d'intersection des droites transversales d et h. Enfin, la troisième et dernière couche inclut les points formant la configuration de Pappus. La structuration en couches n'a que peu d'impact sur le temps de calcul, elle permet néanmoins de rendre la preuve plus

modulaire et ainsi acceptable par le système Coq.

Grâce à toutes ces optimisations, le script de preuve Coq produit contient 152 lemmes, répartis en 3 couches. La couche 1 contient 82 lemmes, la couche 2 contient 69 lemmes et enfin la couche 3 contient un seul lemme : le théorème de Dandelin-Gallucci. Dans cette configuration, le système Coq vérifie le script de preuve, d'une longueur d'environ *50 000* lignes en 5 min et n'utilise au maximum que 7.5% de la mémoire disponible.

3.8 Conclusions et perspectives

Nous avons présenté dans ce chapitre deux approches différentes mais complémentaires pour modéliser des propriétés de la géométrie projective et les démontrer formellement avec un outil comme Coq. L'étude des modèles finis a montré que l'approche basée sur les rangs se généralise et passe plus facilement à l'échelle que celle basée sur la description synthétique quand il s'agit de démontrer des théorèmes. Nous avons également tiré profit du caractère combinatoire de l'approche basée sur les rangs pour construire un prouveur automatique, générant des preuves Coq et capable de prouver des théorèmes emblématiques de la géométrie projective.

3.8.1 Pousser Coq à ses limites avec les modèles finis

Les travaux initiés sur les modèles finis vont se poursuivre, notamment dans le cadre de l'étude des espaces projectifs. De nombreux résultats ont déjà été obtenus dans le cadre du plus petit espace projectif pg(3,2), composé de 15 points et 35 droites. Nous allons étudier comment notre approche se généralise pour prouver des résultats similaires sur des espaces plus grands : pg(3,3), composé de 40 points et 130 droites pour commencer.

Concernant pg(3,2), nous travaillons également à démontrer formellement en Coq que tous les *spreads*[11] (56) sont isomorphes. De même, nous chercherons ensuite à adapter notre méthodologie de preuve pour l'appliquer à des ensembles plus grands, en commençant par pg(3,3).

Deux approches co-existent actuellement pour décrire les espaces projectifs : la première est d'utiliser des ensembles finis tels que ceux définis dans la bibliothèque *Mathematical Components* [MT16]. Malheureusement, cela a un coût au niveau des démonstrations. Faire du raisonnement par induction sur un type fini (`FinType`) est lourd et coûteux. L'autre approche plus pragmatique consiste à utiliser un programme pour générer automatiquement la spécification de ces espaces (le type inductif des points, le type inductif des droites ainsi que la relation d'incidence). Cela permet de travailler avec des types inductifs très simples et donc d'avoir des structures de raisonnement par cas les plus simples possibles. Idéalement, il faudrait arriver à tirer partie du meilleur de ces deux mondes.

A terme, l'objectif est de pouvoir reproduire des résultats de recherche contemporains sur les espaces projectifs et leurs décompositions comme par exemple ceux présentés dans les articles suivants [Bet16, TZ17, TZ19].

3.8.2 De nouvelles preuves automatiques

Afin de confirmer les capacités de notre prouveur automatique, nous étudions actuellement comment prouver automatiquement la réciproque du théorème de Dandelin-Gallucci avec notre prouveur. Il s'agit à partir de l'hypothèse que la propriété de Dandelin-Gallucci est vérifiée, de démontrer la propriété de Pappus. La démonstration mathématique est présentée dans [Hor17]. Dans le cadre de notre prouveur, cela nécessite d'ajouter une règle de réécriture pour la propriété

11. Un *spread* de pg(3,q) est une partition des points de pg(3,q) en $q^2 + 1$ droites.

de Dandelin-Gallucci comme cela a déjà été fait pour la propriété de Pappus. D'autres théorèmes comme celui des tétraèdres homologiques présenté dans [RR06] ainsi que des variantes du théorème de Desargues en dimension $n > 3$ sont également à l'étude.

3.8.3 Outils et intégration du prouveur au système Coq

Nos travaux confirment que la géométrie projective est un bon terrain d'expérimentation pour faire de la preuve automatique. L'approche combinatoire basée sur les rangs permet facilement d'automatiser les démonstrations de propriétés mais cela se fait au prix d'une lisibilité moindre. De plus, elle nécessite pour l'instant l'utilisation d'un prouveur externe, qui produit une trace de la preuve à faire valider par Coq, et qui ne peut donc pas être utilisé de manière interactive.

Allers-retours entre les 2 formalismes de description de la géométrie projective

Les prochaines étapes de développement consistent à proposer des outils pour travailler indifférement et de manière la plus transparente possible dans les deux descriptions. Cela nécessiterait, d'une part, de pouvoir traduire automatiquement un énoncé d'une description vers l'autre et, d'autre part, d'intégrer complètement notre prouveur à Coq pour en faire une tactique utilisable au sein du système. Avec ces outils à disposition, l'utilisateur énoncerait la propriété qu'il cherche à démontrer dans la description synthétique, puis le système convertirait l'énoncé sous forme de rangs et tenterait de le prouver automatiquement. En cas de succès, le théorème serait alors démontré et le travail terminé. Dans le cas contraire, l'utilisateur reprendrait la main pour décomposer la preuve en sous-cas, ou bien pour créer de nouveaux points. A chaque étape de ce travail interactif, le prouveur automatique pourrait être appelé afin de terminer la preuve. En cas d'ajout de points par l'utilisateur, il faudrait avoir conservé la saturation partielle déjà effectuée et simplement la compléter avec les nouveaux sous-ensembles mis en jeu.

Cette approche permettrait de combiner efficacement automatisation pour les parties de preuve à dominante combinatoire et raisonnement humain pour la construction des points par exemple.

Storytelling sur les preuves

Le prouveur produit des scripts de preuve Coq longs et techniques. Ces scripts n'offrent pas le niveau d'abstraction attendu par un mathématicien qui souhaiterait saisir l'essence même de la démonstration et en observer les principales étapes. La seule chose facilement observable est l'existence ou non de la preuve (c'est-à-dire l'acceptation ou non du script de preuve par Coq). Or, un bon moyen de convaincre un mathématicien sceptique que la preuve est bien correcte serait de pouvoir lui raconter la preuve en mettant en avant la structure du raisonnement et les étapes principales tout en laissant de côté les aspects les plus techniques. En expliquant, avec un haut niveau d'abstraction, la preuve à l'utilisateur, on contribue à le convaincre un peu plus que l'énoncé prouvé est bien celui qu'il recherchait et que la preuve obtenue est bien correcte.

Chapitre 4

Calcul réel exact pour la géométrie

La plupart des algorithmes géométriques sont conçus en s'appuyant sur une arithmétique réelle exacte [KMP+04]. L'utilisation d'une arithmétique à virgule flottante à la place d'une arithmétique exacte peut rendre incorrecte l'implantation effective de tels algorithmes. Calculer avec des nombres flottants permet certes de gagner en efficacité, mais au prix d'approximations dont les conséquences peuvent parfois être catastrophiques, comme lors de l'échec de la mission de l'anti-missile Patriot [BOB92]. Dans cet exemple précis, l'algorithme utilisait un incrément de temps de 0.1 s. Cette constante décimale est non-représentable de manière exacte en binaire. Cela a causé une (petite) erreur initiale de 10^{-7}, qui s'est accumulée au fil des calculs et a conduit à l'échec de l'interception du missile venant en face.

En géométrie, comme dans d'autres domaines, la plupart des calculs numériques s'effectuent néanmoins avec des nombres à virgule flottante. Même si ces nombres flottants sont seulement des approximations des nombres réels, la norme IEEE-754 [IEE19] décrit un cadre strict dans lequel on peut prédire et analyser rigoureusement le comportement de nombreux algorithmes numériques [JR17]. En revanche, les nombres à virgule flottante ont une précision limitée et ils ne respectent pas certaines propriétés pourtant élémentaires de l'arithmétique des réels comme l'associativité de l'addition. Des bibliothèques de calcul multiprécision en virgule flottante avec arrondi correct comme MPFR [Zim10] permettent de travailler à une précision arbitraire. Cependant ces bibliothèques ne permettent pas de garantir la précision des nombres durant toute l'exécution d'un programme.

La question de la représentation des réels sur ordinateur est étudiée depuis de nombreuses décennies, avec notamment l'introduction de l'arithmétique des intervalles [Moo62] dans les années 1960. Plusieurs systèmes disposant de capacités de calcul réel exact co-existent actuellement, parmi lesquels on peut citer IRRAM [Mül00], AERN [DFKT14], Core 2 [YYD+10].

Dans notre contexte, celui des preuves formelles et plus spécifiquement de l'écosystème Coq, il existe plusieurs approches distinctes pour décrire, calculer et raisonner avec les nombres réels. La bibliothèque standard de Coq, développée principalement par Micaela Mayero [May01], propose une description axiomatique des réels comme un corps ordonné, archimédien et complet. Cette bibliothèque utilise de plus comme axiome une version spécifique du principe d'omniscience qui exprime que l'ordre sur les réels est total. Cet axiome, par nature classique, permet notamment de décider de l'égalité à 0, ce que nous sommes incapables de faire sur les réels dans un contexte purement intuitionniste. Une autre bibliothèque, C-CoRN (Constructive Coq Repository at Nijmegen), développée initialement par Milad Niqui [GN00] propose, quant à elle, une axiomatisation des nombres réels intuitionnistes. Elle utilise comme notion de base l'*apartness* qui peut être vue comme une forme constructive de l'inégalité, plutôt que la relation d'égalité. Par son caractère constructif, il est possible, en réalisant les axiomes intuitionnistes, d'obtenir des algorithmes de calcul sur les réels [KO09, KS11b]. Une synthèse de la plupart des formalisations

des réels disponibles dans les assistants de preuve contemporains est présentée dans [BLM16].

Dans notre contexte, influencés par la géométrie, nous choisissons de reprendre un point de vue calculatoire un peu oublié pour décrire formellement les nombres réels et le continu : la droite d'Harthong-Reeb. Intuitivement, il s'agit de faire des calculs uniquement sur des entiers (en *zoomant* aussi fortement que nécessaire pour n'effectuer les calculs intermédiaires qu'avec des entiers, et en *normalisant* éventuellement à la fin du calcul pour obtenir le résultat réel recherché).

La construction de la droite d'Harthong-Reeb a pour fondation une arithmétique non-standard des entiers qui n'était malheureusement pas explicitement construite par ses inventeurs. En partant d'une suite naïve d'entiers (1,2,3 ...), Reeb affirme qu'il existe un entier ω plus grand que tous les entiers naïfs. En s'appuyant sur le théorème de compacité de la théorie des modèles [HR04], l'existence de ce grand entier ω suffit pour déduire qu'il existe un modèle non-standard de l'arithmétique entière et que ce modèle peut être utilisé pour construire la droite d'Harthong-Reeb.

En géométrie discrète, ce modèle est à l'origine d'importants développements sur la droite analytique discrète [Rev91]. Nos travaux ont permis de décrire formellement ce modèle en Coq et de nous assurer que la droite d'Harthong-Reeb vérifie bien tous les axiomes de Bridges [Bri99] et est donc bien un modèle des nombres réels constructifs. Pour cela, en première approche, nous montrons qu'en fondant la description de la droite d'Harthong-Reeb sur un modèle axiomatique des entiers non-standards, on obtient une théorie qui vérifie bien tous les axiomes de Bridges, y compris la propriété de la borne supérieure. Dans un second temps, nous construisons un modèle de ces entiers non-standards basés sur les entiers de Laugwitz-Schmieden (qui sont en fait des suites à valeurs entières) et nous établissons les propriétés de ce modèle. La plupart des axiomes de Bridges sont vérifiés, mais certains axiomes mettant en jeu la relation d'ordre ou bien la propriété de la borne supérieure ne sont pas vérifiés tels quels. Nos collègues mathématiciens Agathe Chollet et Guy Wallet ont proposé une variante de ces axiomes que nous décrivons formellement en Coq. Nous étudions comment prouver que ces axiomes sont bien vérifiés par la description constructive de la droite d'Harthong-Reeb.

Nous définissons ensuite en Coq le procédé d'arithmétisation d'Euler permettant de discrétiser des fonctions continues et d'en calculer des approximations numériques à différentes échelles. Ces définitions sont extraites automatiquement de cette description formelle en Coq vers du code certifié (un programme en OCaml), permettant ainsi de tracer les approximations à différents niveaux de précision sur une grille discrète.

Enfin, nous reprenons les travaux de Valérie Ménissier-Morain sur la représentation des nombres réels par les nombres B-approximables [MM94, MM05], qui ont des caractéristiques assez proches des Ω-entiers utilisés pour construire la droite d'Harthong-Reeb. Nous complétons la description formelle en Coq, initialement développée par Jérôme Créci [Cré02], des nombres B-approximables et de leurs propriétés en Coq. L'objectif, à terme, est de relier formellement entre elles ces deux représentations des nombres réels.

Formaliser en Coq une théorie comme celle de la droite d'Harthong-Reeb est non seulement un challenge pour le système d'aide à la preuve, mais aussi un excellent moyen de préciser toutes les étapes potentiellement ambiguës de la preuve. Ici, dans les preuves manuscrites, un *entier* x pouvait appartenir soit à $\mathbb{N}$, soit à $\mathbb{Z}$, soit à l'ensemble des suites à valeurs entières, ou bien même à $\mathbb{R}$. La description formelle de la théorie nous a forcé à lever ces ambiguités et à éventuellement modifier la rédaction de la démonstration pour qu'elle soit correcte. De même, la formalisation nous a permis d'identifier correctement les parties de démonstration indépendantes de la représentation des entiers non-standards choisie et de préciser certaines étapes de raisonnement notamment dans la preuve de la propriété de la borne supérieure.

Ces travaux ont été initiés au cours du projet ANR Galapagos (2008-2011) et ils se pour-

suivent dans le cadre d'une collaboration avec des chercheurs de l'équipe IMAGeS de l'Université de Strasbourg (Marie-Andrée Da Col et Loïc Mazo) et de l'Université de Poitiers (Laurent Fuchs et Gaëlle Largeteau-Skapin). Les principaux résultats sont présentés dans l'article suivant [MCF14].

4.1 Un système d'axiomes pour l'arithmétique non standard

L'idée centrale de la droite d'Harthong-Reeb est d'introduire un mécanisme de changement d'échelle non-trivial sur l'ensemble des entiers afin de modéliser sous une forme discrète le continu. Pour cela, nous devons d'abord construire une arithmétique non-standard. Dans cette section, nous formalisons dans Coq une telle arithmétique non-standard, sur laquelle nous construirons ensuite la droite d'Harthong-Reeb.

4.1.1 Description axiomatique des entiers

Nous décrivons d'abord les nombres entiers (non-standards), leurs opérations et leurs propriétés. Nous encapsulons tous ces objets dans une structure abstraite dont la signature est décrite via un *module type* dans Coq. Du point de vue de la programmation, cela peut être vu comme une interface qui, d'une part, regroupe toutes les propriétés des nombres non-standards dont nous aurons besoin pour construire la droite d'Harthong-Reeb, et qui, d'autre part, peut être implantée par un type concret et des opérations sur ce type concret accompagnés des preuves des axiomes de l'interface comme nous le ferons dans la section 4.3.

Ce *module type* contient les déclarations des objets de la théorie (A est le type de données abstrait dénotant les entiers non-standards). Dans Coq, le mot-clé `Parameter` permet de déclarer un objet abstrait (un type, une opération ou même une propriété) que l'on pourrait implanter plus tard.

```
Parameter A:Type.

Parameter a0 a1 : A.
Parameter plusA multA divA modA : A -> A -> A.
Parameter oppA absA : A -> A.

Parameter leA ltA : A -> A -> Prop.
```

Nous plaçons le type A dans la sorte `Type`, qui est la sorte la plus élevée de Coq. En effet, le système Coq est muni de trois sortes `Prop`, `Set` et `Type`. La sorte `Prop` correspond au type des propositions, alors que la sorte `Set` correspond au type des calculs. De plus, la sorte `Type` est le type des sortes `Prop` et `Set` et contient une hiérachie d'univers ($\text{Type}_i : \text{Type}_{i+1}$). Par souci de simplicité et de cohérence, nous choisissons d'avoir A : `Type`, comme c'était le cas des réels axiomatiques de Coq jusqu'à la version 8 de Coq.

Afin de rendre la description formelle plus agréable à lire, nous introduisons en Coq des notations aussi proches que possible de celles utilisées habituellement en mathématiques.

```
Notation "x + y " := (plusA  x y).
Notation "x * y " := (multA  x y).
Notation "x / y " := (divA  x y).

Notation "0" := (a0).
```

```
Notation "1" := (a1).
Notation "- x" := (oppA x).
Notation "| x |" := (absA x) (at level 60).
Notation "x <= y" := (leA x y) (at level 50).
Notation "x < y" := (ltA x y) (at level 50).
```

Les propriétés des fonctions sur A sont exprimées comme des paramètres de la théorie en Coq. D'une certaine manière, ces paramètres peuvent être considérés comme les axiomes de base de la théorie développée.

```
Parameter plus_neutral : forall x,0 + x = x.
Parameter plus_comm : forall x y,  x + y =  y + x.
Parameter plus_assoc : forall x y z,  x + (y + z) = (x + y) + z.
Parameter plus_opp : forall x, x + (- x) = 0.

Parameter mult_absorb : forall x, x * 0 = 0.

Parameter abs_pos : forall x, 0 <=|x|.
Parameter abs_pos_val : forall x, 0 <=x -> |x|=x.
Parameter abs_neg_val : forall x, x <=0 -> |x|=-x.

[...]
```

Dans la suite, nous ajoutons comme paramètre que l'ensemble des entiers non-standards, dénoté par A avec les opérations $+$, $\times$, ... est muni d'une structure d'anneau. Cela permet de démontrer automatiquement des égalités algébriques et également d'automatiser certaines simplications d'expressions qui peuvent être laborieuses à réaliser à la main. Nous décrivons également une opération de division euclidienne ainsi une opération de valeur absolue ($x \mapsto |x|$).

Enfin, nous supposons que les relations d'ordre $\leq$ et $<$ ont bien leurs propriétés usuelles, notamment la transitivité, la régularité vis-à-vis des opérations comme l'addition, etc. Nous supposons également que ces relations sont décidables en ajoutant l'axiome de décidabilité suivant : (pour tout $x : A$, $x < 0$ ou bien $x = 0$ ou bien $0 < x$), qui est décrit en Coq par l'axiome `A0_dec` :

```
Axiom A0_dec : forall x : A, {x < 0}+{x = 0}+{0 < x}.
```

4.1.2 Aspects non-standards

Même s'il existe des descriptions axiomatiques de l'analyse non-standard comme par exemple IST (Internal Set Theory) [Nel77], nous utilisons ici un système d'axiomes plus faible, dans la lignée des travaux de Nelson ou de Lutz [Nel87, Lut92], et bien adapté à nos objectifs. Nous introduisons un nouveau prédicat `lim` sur les nombres entiers : `lim`(x) "signifie" que l'entier x est limité. Ce prédicat permet de distinguer les entiers qui ne sont pas infiniment grands. Nous introduisons également le nombre ω, noté dans Coq `w`. Ce nombre est plus grand que tous les entiers naïfs, comme l'a affirmé initialement Georges Reeb dans [DR89]. Cette propriété peut se justifier en utilisant le théorème de compacité de la théorie des modèles [Lal90].

```
Parameter lim : A -> Prop.
Parameter w : A.
```

Ce prédicat est externe à la théorie des entiers usuels et ses propriétés sont décrites par les axiomes ANS1, ANS2, ANS3, ANS4 (et ANS5 qui sera introduit un peu plus tard) :

ANS1. *L'entier* 1 *est limité.*

```
Parameter ANS1 : lim 1.
```

ANS2. *La somme et le produit de deux entiers limités sont limités.*

```
Parameter ANS2a : forall x y, lim x -> lim y -> lim (x + y).
Parameter ANS2b : forall x y, lim x -> lim y -> lim (x * y).
```

ANS3. *Il existe des entiers non-limités.*

```
Parameter ANS3 : ~ lim w.
```

Nous supposons simplement que w n'est pas limité (en Coq, `~` dénote la négation logique).

ANS4. *Pour tout couple* $(y, x) \in A^2$ *tel que* y *est limité et* $|x| \leqslant |y|$*, l'entier* x *est limité.*

```
Parameter ANS4 :
  forall x, (exists y, lim y /\ | x | <= | y |)-> lim x.
```

Par commodité d'écriture et de lecture des énoncés mathématiques, nous utilisons les notations suivantes :

— $\forall^{lim} x\ F(x)$ est une abréviation pour $\forall x\ (lim(x) \Rightarrow F(x))$ et peut être lue comme "pour tout x limité, on a $F(x)$".

— $\exists^{lim} x\ F(x)$ est une abréviation pour $\exists x\ (lim(x) \wedge F(x))$ et peut être lue comme "il existe un nombre limité x tel que $F(x)$".

En suivant la terminologie de la théorie IST [Nel77], nous qualifions une formule ou une propriété P d'*externe* quand le prédicat `lim` apparaît dans P et *interne* dans le cas contraire. Cette distinction est nécessaire pour déterminer quand les propriétés connues pour les entiers standards peuvent être généralisées aux entiers non-standards. En fait, quand une propriété P est interne, c'est-à-dire quand elle n'utilise pas le prédicat `lim`, l'extension de P aux nombres infiniment grands [1] est immédiate. Cette propriété est exprimée par le principe de permanence (*Overspill principle*) énoncé ci-après.

Proposition 1. ***(Overspill principle)*** *Soit* $\mathcal{P}(x)$ *une formule interne telle que* $\mathcal{P}(n)$ *est vraie pour tout* $n \in A$, `lim` $n \wedge n \geqslant 0$*. Alors, il existe un nombre infiniment grand* $\nu \in A, \nu \geqslant 0$ *tel que* $\mathcal{P}(m)$ *est vraie pour tout* m *tel que* $0 \leqslant m \leqslant \nu$*.*

```
Parameter overspill_principle : forall P:A -> Prop,
 (forall n:A, lim n /\ 0<=n -> P n) ->
  (exists v:A, ~lim v /\ 0<=v /\ (forall m:A, 0<=m /\ m <=v -> P m)).
```

Il n'est pas possible d'utiliser ce principe de permanence pour les propriétés *externes*. Nous avons besoin d'introduire un nouveau principe d'extension sous la forme d'un axiome, noté ANS5. Ce principe établit qu'une formule qui contient des propriétés externes peut être étendue aux nombres infiniment grands, mais que nous ne savons pas si ces nombres infiniment grands vérifient ou non ces propriétés. Le principe énoncé ci-dessous, qui est notre dernier axiome, s'applique à la fois aux formules *internes* et *externes* P.

ANS5. ***(Principe d'induction externe) :*** *Supposons que*

1. Un nombre *infiniment grand* est la même chose qu'un nombre non-limité positif.

1. *il existe* $x_0 \in A^p$ *tel que* $\mathcal{P}((x_0))$ *;*
2. *pour tout* $n \in A, n \geqslant 0 \wedge lim(n)$ *et pour toute suite* $(x_k)_{0\leqslant k\leqslant n}$ *de* A^p *telle que* $\mathcal{P}((x_k)_{0\leqslant k\leqslant n})$ *il existe* $x_{n+1} \in A^p$ *tel que* $\mathcal{P}((x_k)_{0\leqslant k\leqslant n+1})$.

Alors, il existe une suite interne $(x_k)_{k\in A,k\geqslant 0}$ *dans* A^p *telle que, pour tout* n *qui vérifie lim* $n \wedge n \geqslant 0$*, on a* $\mathcal{P}((x_k)_{0\leqslant k\leqslant n})$.

Ce principe de raisonnement exprime qu'une suite de valeurs x_k vérifiant $\mathcal{P}$ avec k limité peut être prolongée en une suite $(x_k)_{k\in A,k\geqslant 0}$ définie pour tous les entiers. Affirmer que cette suite est interne signifie qu'elle a toutes les propriétés usuelles des suites de nombres. En particulier, si $\mathcal{Q}(x)$ est une formule interne, l'ensemble $\{k \in A, k \geqslant 0 \mid \mathcal{Q}(x_k)\}$ est alors une partie interne de $\{k \in A \mid k \geqslant 0\}$.

En Coq, nous adoptons une définition légèrement différente et plus facile à utiliser pour ANS5. Nous ne considérons que le cas $p = 1$, qui s'avère suffisant dans la suite de notre formalisation. De plus, nous choisissons d'avoir un prédicat P dont l'arité est fixe. Dans la définition de `ANS5` en Coq, le prédicat P s'applique à un seul élement de la suite plutôt qu'à toute la suite. On peut voir cela comme une projection du prédicat de la propriété ANS5 $\mathcal{P}$ sur le k-ième élément de la suite. Cela signifie qu'une propriété telle que $\mathcal{P}((x_k)_{0\leqslant k\leqslant n})$ est traduite par $\forall k, 0 \leqslant k \leqslant n \rightarrow P(x_k)$. Enfin, il faut s'assurer que la suite dont nous montrons l'existence coïncide bien avec la suite initiale pour tout k tel que $0 \leqslant k \leqslant n$. Nous obtenons alors un principe plus faible, mais suffisant pour nos besoins et plus pratique à utiliser dans Coq :

```
Parameter ANS5 :
forall P :A -> Prop,
(forall u : forall n:A, lim n /\ 0 <= n -> A,
  P (u 0 H0) ->
  (forall n:A, forall Hn : lim n /\ 0 <= n,
     (forall k:A, forall Hk:0 <= k /\ k <= n,
      forall Hn1 : lim (n+1)/\0 <=(n+1),
        P (u k (ANS4_special n k Hn Hk)) -> P(u (plusA n 1) Hn1))) ->
          {v:A->A | forall n:A, forall Hn:lim n /\ 0 <= n,
                    forall k:A,  forall Hk:0 <= k /\ k <= n,
                    P (v k) /\ v k = u k (ANS4_special n k Hn Hk)}).
```

Ici, la suite u a deux arguments : un élément n de A et une preuve `H` que `lim` $n \wedge n \geqslant 0$. Dans ce contexte, `H0` est une preuve de `lim` $0 \wedge 0 \geqslant 0$ et `(ANS4_special n k Hn Hk)` [2] est une preuve de `lim` $k \wedge k \geqslant 0$. Nous rappelons au lecteur que `{x:A|P x}` est une notation de Coq utilisée pour définir des Σ-types (`sig P`) existentiels, et qu'elle permet de décrire des ensembles en compréhension. Dans notre cas, il s'agit de l'ensemble des éléments de A qui vérifient P. Un Σ-type étant défini de manière inductive dans Coq, deux projections lui sont automatiquement associées : `proj1_sig`, qui retourne `a` et `proj2_sig` qui retourne une preuve `H` de `P a`.

4.2 La droite d'Harthong-Reeb, vue de manière axiomatique

4.2.1 Définitions et opérations

Nous donnons maintenant la définition du système $\mathcal{HR}_\omega$. Ce système, proposé par Marc Diener [Die92], est une version formelle de la droite dite d'Harthong-Reeb. Nous prouvons que

2. `ANS4_special` est une variante de la propriété ANS4 sans valeur absolue. Son énoncé en Coq est le suivant : `Lemma ANS4_special : forall n k:A, (lim n /\ 0<=n) -> (0<=k /\ k<=n) -> lim k /\ 0<=k .`

ce système peut être considéré comme un modèle de la droite réelle, et qu'il est partiellement constructif. En effet, nous ne disposons pas pour l'instant de moyens de construire un nombre infiniment grand ω. Dans un certain sens, $\mathcal{HR}_\omega$ est équivalent à $\mathbb{R}$ (pour une justification de cette assertion, nous recommandons au lecteur l'article suivant : [CWF+09]).

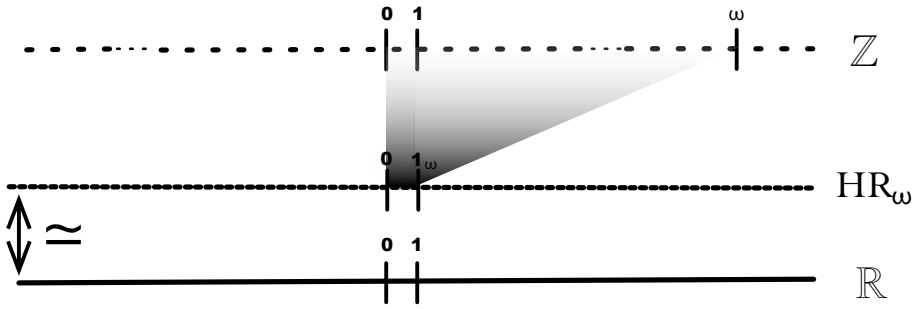


FIGURE 4.1 – Une représentation intuitive de la droite d'Harthong-Reeb $\mathcal{HR}_\omega$.

Au moyen de l'axiome ANS3, la construction débute en considérant un entier positif infiniment grand (c'est-à-dire non-limité) $\omega \in A$. L'objectif est de définir un nouveau système numérique où tous les éléments sont des entiers et où ω est la nouvelle unité. Considérons l'ensemble sous-jacent à ce système.

Définition 10. *L'ensemble $\mathcal{HR}_\omega$ des entiers admissibles à l'échelle ω est défini par :*
$\mathcal{HR}_\omega = \{x \in A \ ; \ \exists^{lim} n \in A, n \geqslant 0 \ \text{ tel que } |x| < n\omega\}$.

Cette définition se traduit facilement en Coq :

```
Definition P := fun (x:A)=> exists n:A, (lim n /\ 0 < n /\  (|x| <= n*w)).
Definition HRw := {x:A | P x}.
```

L'ensemble $\mathcal{HR}_\omega$ est un ensemble externe. De plus, c'est un sous-groupe additif de A. Cet ensemble $\mathcal{HR}_\omega$ est équipé des opérations $+_\omega$ et $\times_\omega$, de l'égalité à l'échelle ω, de l'inégalité à l'échelle ω (notées $=_\omega$ et $\neq_\omega$) ainsi que d'une relation d'ordre $>_\omega$:

Définition 11 (Opérations sur $\mathcal{HR}_\omega$)**.** *Soient X et Y des éléments de $\mathcal{HR}_\omega$.*

- *X et Y sont* égaux *à l'échelle ω et on écrit $X =_\omega Y$ quand* $\forall^{lim} n \in A \quad n > 0 \to n|X - Y| \leqslant \omega$.
- *Y est* strictement plus grand *que X à l'échelle ω et on écrit $Y >_\omega X$ quand* $\exists^{lim} n \in A \quad n > 0 \wedge n\,(Y - X) \geqslant \omega$.
- *X est* différent *de Y à l'échelle ω et on écrit $X \neq_\omega Y$ quand* $(X >_\omega Y$ *ou* $Y >_\omega X)$
- *La* somme *de X et Y à l'échelle ω est $X +_\omega Y := X + Y$ (comme la somme habituelle). Pour cette opération, l'*élément neutre *est $0_\omega = 0$ et l'*opposé *de chaque élément $Z \in \mathcal{HR}_\omega$ est $-_\omega Z := -Z$.*
- *Le* produit *de X et Y à l'échelle ω est $X \times_\omega Y := \lfloor \frac{X.Y}{\omega} \rfloor$ (qui est différent du produit habituel)*[3]*. L'*élément neutre *est $1_\omega := \omega$, et l'*inverse *de chaque élément $Z \in \mathcal{HR}_\omega$ tel que $Z \neq_\omega 0_\omega$ est $Z^{(-1)_\omega} := \left\lfloor \frac{\omega^2}{Z} \right\rfloor$.*

3. Dans la définition du produit, les opérations $t.u$, $\frac{t}{u}$ et $\lfloor u \rfloor$ sont les opérations définies précédemment sur le type A des entiers non-standards.

Les opérations algébriques sont définies sur les entiers de l'ensemble A sur lequel la construction de $\mathcal{HR}_\omega$ s'appuie. Par conséquent, nous devons nous assurer que leurs résultats appartiennent bien à $\mathcal{HR}_\omega$. Pour la somme, il suffit de montrer le lemme suivant :

```
Lemma Padd: forall x y:A, P x -> P y -> P ( x + y).
```

L'addition peut ensuite être définie dans $\mathcal{HR}_\omega$ de la manière suivante :

```
Definition HRwplus (x y: HRw) : HRw :=
match x with exist xx Hxx =>
match y with exist yy Hyy =>
exist P (xx + yy) (Padd xx yy Hxx Hyy)
end end.
```

Tous les lemmes et les définitions formelles en Coq des objets de la définition 9 sont regroupés dans la figure 4.2.

4.2.2 Preuves des axiomes de Bridges

A la fin des années 1990, Brigdes a proposé [Bri99] une description axiomatique de ce qu'est une droite réelle constructive. Cette description se décompose en trois groupes : un premier groupe traitant de la structure algébrique (R1), un deuxième des propriétés d'ensemble ordonné (R2) et un dernier groupe (R3) traitant de certaines propriétés spéciales présentées plus loin. Un corps qui respecte tous ces axiomes est appelé un corps ordonné de Bridges-Heyting. Une preuve, rédigée à la main, que *la droite d'Harthong-Reeb, munie de ces opérations et relations est un corps ordonné de Bridges-Heyting* est fournie dans [CWF$^+$09] et elle n'utilise que la logique intuitionniste.

Dans la suite de cette section, nous montrons comment les axiomes de Bridges peuvent être décrits dans Coq et nous prouvons formellement que notre description de la droite d'Harthong-Reeb vérifie bien tous ces axiomes.

R1. Structure algébrique

Le premier groupe d'axiomes (R1) décrit les propriétés algébriques des opérations $+_\omega$ et $\times_\omega$ attendues d'une droite réelle constructive.

1. $\forall x, y \in \mathcal{HR}_\omega, x +_\omega y =_\omega y +_\omega x$
2. $\forall x, y, z \in \mathcal{HR}_\omega, (x +_\omega y) +_\omega z =_\omega x +_\omega (y +_\omega z)$
3. $\forall x \in \mathcal{HR}_\omega, 0_\omega +_\omega x =_\omega x$
4. $\forall x \in \mathcal{HR}_\omega, x +_\omega (-_\omega x) =_\omega 0_\omega$
5. $\forall x, y \in \mathcal{HR}_\omega, x \times_\omega y =_\omega y \times_\omega x$
6. $\forall x, y, z \in \mathcal{HR}_\omega, (x \times_\omega y) \times_\omega z =_\omega x \times_\omega (y \times_\omega z)$
7. $\forall x \in \mathcal{HR}_\omega, 1_\omega \times_\omega x =_\omega x$
8. $\forall x \in \mathcal{HR}_\omega, x \times_\omega x^{(-1)_\omega} =_\omega 1_\omega$ if $x \neq_\omega 0_\omega$
9. $\forall x, y, z \in \mathcal{HR}_\omega, x \times_\omega (y +_\omega z) =_\omega x \times_\omega y +_\omega x \times_\omega z$

Il est facile de montrer que $\mathcal{HR}_\omega$ vérifie bien ces propriétés. Tous les axiomes de ce groupe peuvent être démontrés formellement en se fondant sur les définitions des opérations utilisées. La plupart des propriétés sont décrites en utilisant l'égalité de Leibniz de Coq (qui est plus forte que l'égalité $=_\omega$ définie ci-dessus). Les propriétés (6), (8) et (9) nécessitent l'utilisation de l'égalité $=_\omega$. A chaque fois, les preuves se font en décomposant les éléments de $\mathcal{HR}_\omega$ en un élément

```
Lemma Padd : forall x y, P x -> P y -> P (x + y).

Definition HRwplus (x y: HRw) : HRw :=
 match x with exist xx Hxx => match y with exist yy Hyy =>
  exist P (xx + yy) (Padd xx yy Hxx Hyy)
end end.

Lemma Popp : forall x, P x -> P (- x).

Definition HRwopp (x: HRw) : HRw :=
 match x with exist xx Hxx => exist P (- xx) (Popp xx Hxx) end.

Definition HRwminus (x y : HRw) : HRw := HRwplus x (HRwopp y).

Lemma Pprod : forall x y, P x -> P y -> P (( x * y) / w).

Definition HRwmult (x y: HRw) : HRw :=
 match x with exist xx Hxx => match y with exist yy Hyy =>
  exist P ((xx * yy) / w) (Pprod xx yy Hxx Hyy)
end end.

Definition HRwequal (x y : HRw) : Prop :=
 match x with exist xx Hxx => match y with exist yy Hyy =>
  (forall n, lim n ->0 < n -> ( (n*|xx + (- yy)|) <= w))
end end.

Definition HRwgt (y x : HRw) : Prop :=
 match y with exist yy Hyy => match x with exist xx Hxx =>
  (exists n, lim n /\ 0 < n /\ (w <= (n*(yy+ (-xx)))))
end end.

Definition HRwge (y x:HRw) : Prop :=
 match y with exist yy Hyy => match x with exist xx Hxx =>
  forall n, lim n /\ 0 < n -> n * (xx + - yy) <= w
end end.

Definition HRwdiff (x y : HRw) : Prop := HRwgt x y \/ HRwgt y x.

Lemma Pdiv : forall x ,  HRwdiff x HRw0 -> P ((w * w ) /(proj1_sig x)).

Definition HRwinv (x : HRw) (H: HRwdiff x HRw0) : HRw :=
 exist P ((w * w ) / (proj1_sig x)) (Pdiv x H).
```

FIGURE 4.2 – Définitions des opérations de $\mathcal{HR}_\omega$ dans Coq.

x de A et une preuve H que $P(x)$ est vérifiée. Nous résumons ici la preuve que $\mathcal{HR}_\omega$ vérifie la première propriété (la commutativité de l'addition). Montrer que les termes `HRwplus` x y et `HRwplus` y x sont égaux dans $\mathcal{HR}_\omega$ revient à montrer, non seulement que les deux témoins

(dans A) sont égaux, mais également que les preuves de leurs propriétés associées $P(x+y)$ et $P(y+x)$ sont égales. Comme ce qui importe est que P soit vérifiée pour l'élément considéré, nous utilisons le principe de *non-pertinence des preuves* pour montrer que toutes les preuves de la même propriété, ici $P(x)$ sont égales [4]. Ce principe peut être décrit en Coq par l'axiome suivant :

```
Axiom proof_irrelevance : forall (P:Prop) (p1 p2:P), p1 = p2.
```

Ce principe bien connu est consistant avec la logique de Coq sous réserve que P soit dans la sorte `Prop` [Chl13, Chap. 4]. Nous pouvons par conséquent l'utiliser sans risques dans notre développement. Une fois que toutes les propriétés du premier groupe sont prouvées pour $\mathcal{HR}_\omega$ dans Coq, nous déclarons dans Coq que $\mathcal{HR}_\omega$ muni de ses opérations et de la relation d'égalité $=_\omega$ possède une structure d'anneau. Cela fonctionne de manière similaire à ce que nous avons fait pour le type A et cela permet d'utiliser les tactiques `ring` et `ring_simplify` pour faire des simplifications d'expressions algébriques de type $\mathcal{HR}_\omega$. La seule restriction est que les propriétés de morphisme comme

$$\forall x, y, z, t \in \mathcal{HR}_\omega \quad x =_\omega y \Rightarrow z =_\omega t \Rightarrow x \times_\omega z =_\omega y \times_\omega t$$

doivent être prouvées manuellement par l'utilisateur avant de pouvoir faire de la réécriture avec cette égalité.

R2. Propriétés de base de $>_\omega$

Le deuxième groupe d'axiomes expriment les propriétés attendues de la relation d'ordre $>_\omega$.

1. $\forall x, y \in \mathcal{HR}_\omega, \neg (x >_\omega y \wedge y >_\omega x)$
2. $\forall x, y \in \mathcal{HR}_\omega, (x >_\omega y) \Rightarrow \forall z\ (x >_\omega z\ or\ z >_\omega y)$
3. $\forall x, y \in \mathcal{HR}_\omega, \neg(x \neq_\omega y) \Rightarrow x =_\omega y$
4. $\forall x, y \in \mathcal{HR}_\omega, (x >_\omega y) \Rightarrow \forall z\ (x +_\omega z >_\omega y +_\omega z)$
5. $\forall x, y \in \mathcal{HR}_\omega, (x >_\omega 0_\omega \wedge y >_\omega 0_\omega) \Rightarrow x \times_\omega y >_\omega 0_\omega$

Toutes ces propriétés se démontrent aisément dans Coq. Notons que la définition de l'inégalité $\neq_\omega$ est définie en utilisant la relation d'ordre $>_\omega$ sur $\mathcal{HR}_\omega$: $X \neq_\omega Y \equiv (X >_\omega Y \wedge Y >_\omega X)$ plutôt que d'utiliser la négation et l'égalité comme cela est fait habituellement. Cette définition permet de se contenter de l'hypothèse de décidabilité de l'égalité sur A pour prouver toutes les propriétés du groupe R2 sans forcément disposer de la décidabilité de la relation d'ordre $>_\omega$ à l'échelle ω. Cette hypothèse de décidabilité de l'égalité est simplement un axiome supplémentaire dans la version axiomatique de l'arithmétique non-standard que nous considérons jusqu'à présent. Néanmoins, cela sera un problème dans la pratique, quand nous chercherons à implanter concrètement la droite d'Harthong-Reeb. Ces questions seront étudiées en détail dans la section 4.3 où nous présentons un modèle basé sur les entiers de Laugwitz-Schmieden, dans lequel cette inégalité n'est pas décidable. Une alternative, proposée par G. Wallet, serait d'obtenir cette propriété en considérant un autre modèle s'appuyant sur une variante non-standard de la théorie des types de Martin-Löf [ML84, ML90].

4. Pour cela, nous utilisons les définitions et axiomes de la bibliothèque standard de Coq disponibles dans le fichier `ProofIrrelevance.v`.

Liens entre les relations d'ordre dans $\mathcal{HR}_\omega$ et dans A. Nous rappelons que, dans $\mathcal{HR}_\omega$, la relation *strictement supérieur* $>_\omega$ et la relation *supérieur ou égal* $\geqslant_\omega$ sont définies à partir de la relation *inférieur ou égal* sur le type A ($y >_\omega x \equiv \exists^{\lim} n \in A \quad n\,(y - x) \geqslant \omega$ et $y \geqslant_\omega x \equiv \forall^{\lim} n \in A \quad n\,(y - x) \leqslant \omega$). Cependant, nous caractérisons la relation $\geqslant_\omega$ en nous appuyant sur la relation $>_\omega$:

$$\forall a, b \in \mathcal{HR}_\omega, b \geqslant_\omega a \qquad \Longleftrightarrow \qquad (\forall c \in \mathcal{HR}_\omega, a >_\omega c \Rightarrow b >_\omega c).$$

Cela rend les preuves des propriétés de $\geqslant_\omega$ plus faciles à écrire. Enfin, nous établissons également les deux théorèmes suivants pour tout $a, b \in \mathcal{HR}_\omega$:

$$a \geqslant b \text{ implique } a \geqslant_\omega b \quad \text{et} \quad a >_\omega b \text{ implique } a > b$$

Ces propriétés sont fondamentales dans notre développement et se démontrent aisément en Coq. Elles nous permettent de passer facilement, quand c'est pertinent, des propriétés d'ordre sur les éléments de A à celles sur les éléments de $\mathcal{HR}_\omega$, et inversement.

R3. Propriétés spéciales de $>_\omega$

Les deux derniers axiomes du système proposé par Bridges sont les suivantes :

1. **Axiome d'Archimède** : Pour tout $X \in \mathcal{HR}_\omega$, il existe un entier $n \in A$ tel que $X < n$.
2. **Le principe constructif de la borne supérieure** qui est introduit dans la section suivante.

La propriété d'Archimède peut être facilement décrite dans Coq par l'énoncé suivant :

```
Lemma Archimedes : forall X:HRw, exists n:HRw, n >=w X.
```

Démonstration. Sa preuve est immédiate puisque les éléments x de $\mathcal{HR}_\omega$ sont tels qu'il existe un nombre limité $k \in \mathbb{N}$, tel que $|x| < k\omega$. La propriété peut donc être prouvée en utilisant l'entier $k\omega$ comme témoin pour n. □

4.2.3 Propriété de la borne supérieure

Nous commençons par quelques définitions. Un sous-ensemble S de $\mathcal{HR}_\omega$ est un ensemble d'éléments de $\mathcal{HR}_\omega$ qui satisfait une certaine propriété définie dans le système formel. Un tel sous-ensemble S est *borné pour la relation* $\geqslant_\omega$ s'il existe $b \in \mathcal{HR}_\omega$ tel que $b \geqslant_\omega s$ pour tout $s \in S$; l'élément b s'appelle un *majorant* de S. Une *borne supérieure* pour S est un élément $b \in \mathcal{HR}_\omega$ tel que :

— $\forall s \in S \quad b \geqslant_\omega s$ (b est un majorant de S) ;

— $\forall b'\ (b >_\omega b') \Rightarrow (\exists s \in S \quad s >_\omega b')$.

Une borne supérieure est unique : si b et c sont deux bornes supérieures de S, alors on a les propriétés suivantes : $\neg(b >_\omega c)$ et $\neg(c >_\omega b)$; par conséquent, en vertu des propriétés des relations $>_\omega$, $\geqslant_\omega$ et $=\omega$, on en déduit que $c \geqslant_\omega b$ et $b \geqslant_\omega c$ et par conséquent que $b =_\omega c$.

Le principe constructif de la borne supérieure : Soit S un sous-ensemble non-vide de $\mathcal{HR}_\omega$, qui est borné pour la relation $\geqslant_\omega$, tel que pour tous $\alpha, \beta \in \mathcal{HR}_\omega$ avec $\beta >_\omega \alpha$, ou bien β est un majorant de S ou bien il existe $s \in S$ avec $s >_\omega \alpha$; alors S admet une borne supérieure.

Démonstration. La formalisation et la preuve de cette propriété dans Coq suit la preuve mathématique informelle proposée par Agathe Chollet dans [CWF+09], qui s'appuie elle-même sur les idées proposées par Bridges dans [BR97]. Cette preuve est structurée en deux parties : nous construisons un candidat b pour la borne supérieure, puis nous vérifions que ce candidat b est bien la borne supérieure.

Définitions en Coq Un sous-ensemble est décrit par sa propriété d'appartenance : $S\ x$ signifie que x appartient à l'ensemble S. Nous définissons ensuite les notions de *majorant* (`upper_bound` et de *borne supérieure* (`least_upper_bound`).

```
Definition subset := HRw->Prop.

Definition least_upper_bound (S:subset) (b:HRw) : Prop :=
 (forall s:HRw, (S s -> b >=w s)) /\
 (forall b':HRw, (b >w b') -> exists o:HRw, S o/\ o >w b').

Definition upper_bound (X:subset) (m:HRw) : Prop :=
 forall x:HRw, X x -> m >w x.
```

Deux suites (s_n, b_n) Dans la version initiale de la preuve à la main, quatre suites $(s_n, b_n, \alpha_n, \beta_n)$ mutuellement récursives sont définies afin de construire un candidat b pour la borne supérieure. La formalisation en Coq nous a permis de constater qu'il suffisait de calculer récursivement les deux suites s_n et b_n. En effet, pour chaque rang n, les suites α_n et β_n ne dépendent que des valeurs de s_n et de b_n. Il n'est donc pas utile de les inclure dans le processus de calcul récursif des suites (s_n) et (b_n). A chaque étape, les valeurs α_n et β_n peuvent être recalculées de la manière suivante :

$$\alpha_n = \frac{2}{3} \times s_n + \frac{1}{3} \times b_n \qquad \beta_n = \frac{1}{3} \times s_n + \frac{2}{3} \times b_n.$$

Calculer les termes suivants s_{n+1} et b_{n+1} des suites ne dépend que les deux termes précédents s_n et b_n. A partir de ces éléments, nous pouvons constuire α_n et β_n ainsi que la preuve de la propriété $\beta_n >_\omega \alpha_n$. Cependant, nous devons propager à chaque étape n les trois propriétés principales de s_n et b_n :

— s_n appartient à S

— b_n est un majorant de S

— $b_n -_\omega s_n =_\omega (\frac{2}{3})^n \times_\omega (b_0 -_\omega s_0)$.

Nous réalisons cela en spécifiant aussi précisément que possible la fonction lors de sa définition formelle en Coq. Cela se traduit par un nombre élevé de postconditions pour la fonction `def_s_b`. Ces propriétés caractérisent le résultat (s_n, b_n) de la fonction appelée sur les paramètres n et Hn (qui est une preuve que `lim` $n \wedge 0 \leqslant n$).

```
Definition def_s_b :
    forall n:A, forall Hn:(lim n /\ 0 <= n),
{sn:HRw &
  {bn:HRw &
      S sn /\
      upper_bound S bn /\
      bn +w (-w sn) =w ((power two_third n Hn)*w (b0 +w (-w s0)))
  }
}.
```

La notation $\{x : A\ \&\ Q\ x\}$ correspond à la quantification existentielle (calculatoire) dans Coq. Elle exprime qu'il existe un élément x de type A tel que l'on a $Q\ x$, avec Q une fonction à valeurs dans la sorte `Type`. Une telle quantification peut être éliminée (contrairement à la quantification existentielle usuelle avec la sorte `Prop`) pour utiliser les valeurs de `sn` et `bn` dans la suite des calculs.

Initialement, nous commençons avec (s_0, b_0) où s_0 est un élément arbitraire de S, b_0 un majorant de S, $\alpha_0 = \frac{2}{3}s_0 + \frac{1}{3}b_0$ et $\beta_0 = \frac{1}{3}s_0 + \frac{2}{3}b_0$.

Cela nécessite une hypothèse en Coq qui exprime que l'on peut toujours choisir un élément arbitraire s de S. Cela se traduit par une forme d'axiome du choix, qui s'exprime de la manière suivante :

```
Definition non_empty (S:subset) := {x:HRw|S x}.
```

Grâce à la construction $\{x : \mathcal{HR}_\omega \ \&\ X\ x\}$, la définition précédente nous donne un principe d'élimination permettant de prendre (quand on en a besoin) un élément x de $\mathcal{HR}_\omega$ dans X.

Supposons que pour un n donné, nous disposions des termes (s_n, b_n). Nous pouvons construire les termes (α_n, β_n) ainsi qu'une preuve de la propriété $\alpha_n <_\omega \beta_n$. Par hypothèse, deux cas distincts peuvent se produire :

— **Premier cas** β_n est un majorant de S. Nous choisissons alors s_{n+1} et b_{n+1} tels que $s_{n+1} = s_n$ et $b_{n+1} = \beta_n$.

— **Deuxième cas** il existe s tel que $(S\ s)$ et $\alpha_n <_\omega s$. Nous choisissons alors s_{n+1} et b_{n+1} tels que $s_{n+1} = s$ et $b_{n+1} = b_n + s - \alpha_n$.

Propriétés clés. Plusieurs propriétés clés des éléments de la suite sont déjà décrits dans le type de la fonction `def_s_b`. Ces propriétés sont vérifiées par construction (c'est-à-dire qu'elles sont établies en même temps que les suites sont effectivement calculées). Parmi elles, nous savons que, pour chaque n, limité et positif, la propriété $S(s_n) \wedge$ `upper_bound` $S\ b_n$ est vérifiée. Nous en déduisons que pour tout k et pour tout n, la propriété suivante est vérifiée : $b_k >_\omega s_n$. De plus, b_n et s_n sont reliés par la relation suivante :

$$b_n -_\omega s_n =_\omega (\frac{2}{3})^n \times_\omega (b_0 -_\omega s_0).$$

En plus des propriétés déjà spécifiées dans le type de `def_s_b`, nous avons besoin de montrer que la suite (s_n) est croissante. Bien que cela soit immédiat à partir de la définition mathématique, c'est assez technique à démontrer en Coq. En effet, l'application du principe d'induction spécifique `nat_like_induction` (qui simule le principe d'induction des enters naturels `nat_ind` pour les éléments standards de A) et ses règles de réduction associées est rendue plus difficile par la présence de Σ-types existentiels et de constructions *let-in* dans la définition `def_s_b`. De plus, replier les définitions expansées (notamment l'appel récursif à `nat_like_induction`) doit être fait à la main [5].

Grâce à l'axiome ANS5, les suites (s_n) et (b_n) peuvent être étendues à tous les entiers, y compris les non-limités. De plus, le principe de permanence nous permet d'établir l'existence d'un nombre infiniment grand ν tel que la propriété suivante est vérifiée :

$$\min_{0 \leqslant k \leqslant \nu} b_k \geqslant s_\nu \geqslant \ldots \geqslant s_1 \geqslant s_0.$$

Un candidat pour la borne supérieure de S : $b := \min_{0 \leqslant k \leqslant \nu} b_k$

L'étape suivante, qui consiste à vérifier que b est bien la borne supérieure, peut se résumer à la démonstration des deux propriétés suivantes :

— d'une part, b est un majorant de S,

5. Cela nécessite de *copier-coller* le but courant et d'appliquer la tactique `change` pour remplacer explicitement ce terme par un autre, mieux adapté à la suite de la preuve. Le système peut établir que les deux termes sont convertibles, mais il ne peut pas inventer le terme que l'on souhaiterait avoir dans le but.

— d'autre part, b est effectivement une borne supérieure, c'est-à-dire que pour tout $b' <_\omega b$, il existe un élément $s \in S$ tel que $s >_\omega b'$.

□

Le lecteur intéressé pourra consulter les détails de ces preuves dans les fichiers `LUB*.v` du développement Coq (`http://dpt-info.u-strasbg.fr/~magaud/Harthong-Reeb/`).

A ce stade, nous avons démontré formellement que la droite d'Harthong-Reeb, construite à partir d'un ensemble minimal d'axiomes pour l'arithmétique non-standard comme celui présenté dans la section 4.1, vérifie bien la description de Bridges d'une droite constructive réelle.

4.3 Une représentation concrète de la droite d'Harthong-Reeb

Afin de donner une dimension calculatoire à la droite d'Harthong-Reeb, nous cherchons à en construire une implantation informatique. Cette implantation s'appuie sur les entiers de Laugwitz-Schmieden (qui sont des suites à valeurs entières) pour capturer l'arithmétique non-standard. Nous étudions dans quelle mesure cette construction peut être un modèle de la droite d'Harthong-Reeb. Nous montrons que c'est un modèle (d'une forme alternative) du continu mais que, en revanche, ce n'est pas un modèle de la droite d'Harthong-Reeb. Nous proposons alors deux solutions radicalement différentes pour avoir un modèle de la droite d'Harthong-Reeb : un point de vue théorique consistant à modifier légèrement les axiomes de la droite d'Harthong-Reeb pour qu'ils soient prouvables dans cette représentation ou alors une restriction des objets considérés pour qu'ils se comportent conformément aux axiomes.

4.3.1 Les entiers de Laugwitz-Schmieden

Les Ω-nombres de Laugwitz et Schmieden permettent d'étendre un système numérique classique dans le but d'obtenir un système non-standard correspondant. Ici, nous présentons cette extension pour les entiers. Cette extension peut être vue comme un modèle de la définition axiomatique de l'arithmétique non-standard présenté dans la section 4.2. Dans [Lau83, Lau92, LS58], Laugwitz et Schmieden étendent les nombres rationnels et montrent que leur système est équivalent aux nombres réels classiques. Dans cette section, nous présentons uniquement les notations essentielles pour construire une représentation concrète de la droite d'Harthong-Reeb. Dans le but d'étendre la théorie des nombres entiers, Laugwitz et Schmieden introduisent un nouveau symbole Ω en plus des entiers usuels et de leurs opérations $(0, 1, 2, 3, \ldots, +, /, \ldots)$. La seule propriété que ce nombre Ω doit vérifier est la suivante, improprement appelée *Basic Definition* en anglais par Laugwitz et Schmieden [Lau83, Lau92]).

Propriété 3. *Soit $S(n)$ un énoncé sur $\mathbb{N}$ dépendant de $n \in \mathbb{N}$. Si $S(n)$ est vraie pour presque tout $n \in \mathbb{N}$, alors $S(\Omega)$ est vraie.*

Dans cette propriété, l'expression "*presque tout* $n \in \mathbb{N}$" signifie "pour tout $n \in \mathbb{N}$ à partir d'un certain rang N", c'est-à-dire "$(\exists N \in \mathbb{N})$ tel que $(\forall n \in \mathbb{N})$ avec $n > N$". Dans ce contexte, nous allons représenter chaque élément a de cette théorie par une suite $(a_n)_{n\in\mathbb{N}}$. Nous pouvons immédiatement vérifier que le nombre Ω est infiniment grand, c'est-à-dire plus grand que chaque élément de $\mathbb{N}$. En effet, pour $p \in \mathbb{N}$, nous appliquons la propriété appelée *Basic Definition* (BD) à l'énoncé $p < n$, qui est vérifiée pour presque tout $n \in \mathbb{N}$; par conséquent $p < \Omega$ pour tout $p \in \mathbb{N}$. Un candidat naturel pour Ω est la suite $(n)_{n\in\mathbb{N}}$.

D'un point de vue plus technique, les Ω-entiers sont définis en Coq comme des suites d'entiers relatifs (`Z`) indicées par des entiers naturels (`nat`). La fonction `Z_of_nat`, utilisée ci-après, est simplement l'injection canonique des entiers naturels vers les entiers relatifs.

```
Definition A := nat->Z.

Definition a0 : A := fun (n:nat) => 0%Z.
Definition a1: A := fun (n:nat) => 1%Z.

Definition w :A := fun (n:nat) => (Z_of_nat n).
```

Pour construire la droite d'Harthong-Reeb, il faut disposer d'un élément infiniment grand ω. Nous choisissons d'utiliser la suite $w_n = n$. Cependant, notre développement formel est paramétré pour pouvoir être utilisé avec trois ω différents, définis par $w_n = n$, $w_n = n^2$ ou bien $w_n = 2^n$. La seule contrainte est que la suite (w_n) doit être strictement croissante. La suite $w_n = 2^n$ est la plus pratique parce que tous ses termes sont non-nuls, ce qui simplifie la définition de l'opération de division dans l'implantation (voir section 4.4 pour plus de détails).

Pour comparer des Ω-nombres, nous déclarons une nouvelle relation d'équivalence :

Définition 12. *Soient $a = (a_n)_{n\in\mathbb{N}}$ et $b = (b_n)_{n\in\mathbb{N}}$ deux Ω-nombres, a et b sont égaux s'il existe $N \in \mathbb{N}$ tel que pour tout $n > N$, $a_n = b_n$.*

Cette définition se traduit naturellement en Coq de la manière suivante :

```
Definition equalA (u v:A) :=
 exists N:nat, forall n:nat, n>N -> u n=v n.
```

Nous définissons deux classes d'éléments dans cette théorie non-standard :

- la classe des éléments *limités* : ce sont les éléments $\alpha = (\alpha_n)_{n\in\mathbb{N}}$ qui vérifient la propriété $\exists p \in \mathbb{Z}$ tel que $\exists N \in \mathbb{N},\ \forall n > N, \alpha_n < p$ (c'est par exemple le cas de $(2)_{n\in\mathbb{N}}$).
- la classe des éléments *infiniment grands* (c'est-à-dire les éléments positifs non-limités), qui sont des suites de la forme $\alpha = (\alpha_n)_{n\in\mathbb{N}}$ avec $lim_{n\to+\infty}\alpha_n = +\infty$

L'axiome ANS3 garantit l'existence de nombres non-limités, qui, rappelons-le, correspondent aux nombres infiniment grands.

Les opérations et les relations sur A (l'ensemble des Ω-nombres) sont définies de la manière suivante et peuvent être immédiatement formalisées dans Coq.

Définition 13. *Soient $a = (a_n)_{n\in\mathbb{N}}$ et $b = (b_n)_{n\in\mathbb{N}}$ deux Ω-nombres,*

— $a + b =_{def} (a_n + b_n)_{n\in\mathbb{N}}$ *et* $-a =_{def} (-a_n)$ *et* $a \times b =_{def} (a_n \times b_n)_{n\in\mathbb{N}}$;

```
Definition plusA (u v:A) := fun (n:nat) => Zplus (u n) (v n).
Definition multA (u v:A) := fun (n:nat) => Zmult (u n) (v n).
Definition oppA (u:A) := fun (n:nat) => Zopp (u n).
```

— $a > b =_{def} [(\exists N \forall n > N)\ a_n > b_n]$ *et* $a \geqslant b =_{def} [(\exists N \forall n > N)\ a_n \geqslant b_n]$ *;*

```
Definition leA (u v:A) :=
   exists N:nat, forall n:nat, n>N -> Zle (u n) (v n).

Definition ltA (u v:A) :=
   exists N:nat, forall n:nat, n>N -> Zlt (u n) (v n).
```

— $|a| =_{def} (|a_n|)$.

```
Definition absA (u:A) := fun (n:nat) => Zabs (u n).
```

Spécificités de la théorie. En ce qui concerne la relation d'ordre, les propriétés usuelles sur $\mathbb{Z}$ ne sont pas toujours vérifiées sur $\mathbb{Z}_\Omega$ (qui correspond aux suites à valeurs entières de type $\mathbb{N} \to \mathbb{Z}$ et est noté `A` dans notre développement formel). Par exemple, ce n'est pas un ordre total et

$$(\forall a, b \in \mathbb{Z}_\Omega) \quad (a \geqslant b) \vee (b \geqslant a) \tag{4.1}$$

n'est donc pas une proposition valide comme le montre l'exemple des deux Ω-entiers $a = ((-1)^n)_{n\in\mathbb{N}}$ et $b = ((-1)^{n+1})_{n\in\mathbb{N}}$. Cependant, étant donnés deux Ω-nombres arbitraires $a = (a_n)$ et $b = (b_n)$, nous pouvons prouver que

$$(\forall n \in \mathbb{N}) \quad (a_n \geqslant b_n) \vee (b_n \geqslant a_n) \tag{4.2}$$

grâce à la propriété de décidabilité de l'ordre sur $\mathbb{Z}$. En utilisant la propriété *Basic Definition* (BD), nous obtenons $(a_\Omega \geqslant b_\Omega) \vee (b_\Omega \geqslant a_\Omega)$ et par conséquent (4.1) puisque $a_\Omega = a$ et $b_\Omega = b$. Cela mène à une contradiction. Pour éviter cela, nous devons accepter le fait que l'utilisation de la *Basic Definition* (BD) conduise à une notion de vérité plus faible que la définition habituelle.

Au bout du compte, cela montre que les Ω-nombres ne sont pas un modèle exact de la théorie présentée dans la section 4.2. Le problème principal est la propriété de décidabilité `A0_dec` de la relation d'ordre qui n'est évidemment pas démontrable dans le cadre des Ω-nombres. Dans la section suivante, nous montrons comment modifier légèrement la théorie de la section 4.2 pour que les Ω-nombres en sont effectivement un modèle.

Axiomes non-standards Nous définissons deux prédicats `std` et `lim` : (`std` n) exprime qu'un Ω-nombre n est standard (c'est-à-dire dénoté par une suite constante à partir d'un certain rang) et (`lim` n) exprime que n est limité (c'est-à-dire que sa valeur absolue est bornée par un nombre standard positif).

```
Definition std (u:A) :=
 exists N:nat, forall n m, n>N -> m>N -> (u n)=(u m).

Definition lim (a:A) :=
 exists p, std p /\ leA a0 p /\ ltA (absA a) p.
```

Ces définitions nous permettent de démontrer les axiomes ANS1 à ANS4 présentés dans la section 4.2. A ce stade, nous n'avons pas étudié comment définir ANS5 de manière constructive et démontrer formellement en Coq cet énoncé dans le contexte des Ω-nombres.

4.3.2 La droite d'Harthong-Reeb avec les entiers de Laugwitz-Schmieden

Instantiation du système d'axiomes minimal pour l'arithmétique non-standard

Nous étudions maintenant comment prouver les propriétés de notre système d'axiomes minimal pour l'arithmétique non-standard (présenté dans la section 4.1) dans le contexte des Ω-nombres de Laugwitz-Schmieden. Cela nécessite la conception et l'implantation de quelques tactiques avancées avec Ltac, notamment pour pouvoir travailler facilement avec les énoncés contenant des quantifications existentielles.

Considérons l'exemple suivant où A (instancié par `nat`$\to$`Z`) est le type des suites à valeurs entières.

$$\texttt{lt\_plus} : \forall x\, y\, z\, t : A, x < y \Rightarrow z < t \Rightarrow (x + z) < (y + t).$$

Prouver cette propriété, revient, une fois que toutes les définitions ont été dépliées, à prouver l'énoncé suivant :

$$\begin{array}{l}\forall x\, y\, z\, t : \texttt{nat} \to \texttt{Z},\\ \forall H : \exists P : nat, \forall p : \texttt{nat}, p > P \to (x\ p < y\ p),\\ \forall H0 : \exists Q : \texttt{nat}, \forall q : \texttt{nat}, n > Q \to (z\ q < t\ q),\\ \quad \exists N : \texttt{nat}, \forall n : \texttt{nat}, n > N \to (x\ n + z\ n < y\ n + t\ n).\end{array}$$

Cela nécessite d'extraire deux témoins d'existence P et Q, puis de construire un nouveau témoin $N := P+Q+1$. On généralise ensuite toutes les hypothèses provenant de la destruction de H (resp. $H0$) avec la variable quantifiée universellement n et une preuve que n est plus grand que P (resp. Q), ce qui est bien sûr le cas puisque n est plus grand que $P+Q+1$ par hypothèse. Nous obtenons alors une inégalité sur `Z`, qui peut être prouvée en utilisant la tactique automatique `omega` ou un lemme adéquat de la bibliothèque `ZArith` de Coq. Les tactiques `solve_ls` et `solve_ls_w_l` (qui prend en argument supplémentaire un lemme de la bibliothèque `ZArith`) réalisent tout ce travail en une seule étape.

Limitations des entiers de Laugwitz-Schmieden

Les entiers de Laugwitz-Schmieden ne peuvent pas être un modèle de la droite d'Harthong-Reeb telle qu'elle est définie à la section 4.2. Dans [CWF$^+$12], Chollet et al. identifient parmi toutes les propriétés requises pour être un corps de Bridges-Heyting celles qui ne peuvent pas être vérifiées dans l'instantiation de la droite d'Harthong-Reeb avec les Ω-nombres.

Le théorème 7 extrait de [CWF$^+$12] et présenté dans la figure 4.3 résume cet état de fait. Il affirme que trois des axiomes de Bridges, à savoir : R.2.2, R2.3 et R3.2 (le principe de la borne supérieure) ne sont pas vérifiées quand les entiers non-standards sont représentés par des entiers de Laugwitz-Schmieden. Cependant, ces trois énoncés peuvent être légèrement modifiés pour être prouvables dans le cas des entiers de Laugwitz-Schmieden. Le nouveau système d'axiomes conduit alors à une forme alternative du continu (qui est donc différente de $\mathbb{R}$ que nous cherchons à représenter).

Théorème 7. *Le système $\mathcal{HR}_\omega$ a les propriétés suivantes :*
(a) *Tous les axiomes de corps ordonné de Bridges-Heyting sont vérifiés, hormis les axiomes* R2.2, R2.3 *et* R3.2.
(b) R2.2' *Si $x, y \in \mathcal{HR}_\omega$ sont tels que $x <_\omega y$, alors pour tout $z \in \mathcal{HR}_\omega$, il existe $q \in \mathbb{N}$ tel que $(q(y-z) \geqslant \omega) \vee (q(z-x) \geqslant \omega)$ est faiblement vrai.*
(c) R2.3' *Si $x, y \in \mathcal{HR}_\omega$ sont tels que $x \triangle y$ (congruents) et $\neg(x \neq_\omega y)$, alors $x =_\omega y$.*
(d) R3.2' *Si S est un sous-ensemble non vide de $\mathcal{HR}_\omega$ qui est majoré pour la relation $\geqslant_\omega$ et tel que pour tous $(\alpha, \beta) \in \mathcal{HR}_\omega^2$ avec β un majorant de S, $\alpha \in S$ et pour tous $(a, b) \in \mathcal{HR}_\omega^2$ tels que $\alpha \leqslant_\omega a \leqslant_\omega b \leqslant_\omega \beta$, ou bien b un majorant de S ou bien il existe $s \in S$ avec $s >_\omega a$. Alors il existe un élément $\tau \in \mathcal{HR}_\omega$ qui est la borne supérieure de S dans le sens suivant :*
(I') $\forall \mu <_\omega \tau$, $\exists s \in S$ *tel que* $\mu <_\omega s$ *(même propriété que celle de la borne supérieure usuelle)*
(II') *($\forall \nu \in \mathcal{HR}_\omega$ tel que $\tau <_\omega \nu$)($\exists b$ un majorant de S) $\tau \leqslant_\omega b <_\omega \nu$*

FIGURE 4.3 – Théorème issu de [CWF$^+$12], dont on extrait un système d'axiomes alternatif pour représenter le continu : $\mathcal{HR}_\omega$ dénote le système numérique $(\mathcal{HR}_\omega, >_\omega, +_\omega, \times_\omega, 0_\omega, 1_\omega, \mathrm{Op}_\omega, \mathrm{Inv}_\omega)$ s'appuyant sur les Ω-entiers de Laugwitz-Schmieden.

Notre formalisation en Coq confirme qu'il n'est pas possible de prouver les axiomes R2.2, R2.3 et R3.2 dans le modèle s'appuyant sur les entiers de Laugwitz-Schmieden. Le problème

vient de l'impossibilité de prouver l'axiome `A0_dec` (où A est instancié par `nat→Z`) dans le contexte des Ω-entiers de Laugwitz-Schmieden :

```
Axiom A0_dec : forall x: A, {x < 0}+{x = 0}+{0 < x}.
```

En effet, comme nous l'avons vu dans la section précédente, nous ne disposons pas de propriétés de décidabilité sur l'ordre des suites a et b en général. Nous pouvons simplement établir une propriété de décidabilité pour chacun des élements des suites (a_n) et (b_n). De plus, des propriétés simples telles que la généralisation de la propriété de croissance d'une suite, qui s'appuyent sur des propriétés de décidabilité ne sont plus démontrables dans ce contexte. C'est le cas notamment de l'énoncé suivant :

$$\forall n, s_n \leq s_{n+1} \Rightarrow (\forall p\, q, p \leq q \Rightarrow s_p \leq s_q).$$

Malgré le constat que nous n'obtiendrons pas exactement le modèle espéré, nous avons néanmoins mené le travail de formalisation à son terme et décrit dans Coq les axiomes alternatifs proposés par Agathe Chollet dans sa thèse [Cho10]. Une partie de la description formelle de ces axiomes est présentée à la figure 4.4.

Le système formel construit en s'appuyant sur les entiers de Laugwitz-Schmieden ne vérifie pas tous les axiomes de Bridges-Heyting. Néanmoins toutes les propriétés algébriques (le groupe d'axiomes R1) de cette axiomatique sont vérifiées. Nous pouvons donc utiliser cette représentation du continu pour faire des calculs. Nous explorons cette possibilité dans le contexte de la géométrie discrète. Dans la section suivante, nous décrivons un procédé d'arithmétisation bien adapté à la droite d'Harthong-Reeb pour modéliser des objets continus par des courbes discrètes.

4.4 Application en géométrie discrète : Arithmétisation d'Euler et courbes discrètes

Dans cette section, nous montrons comment utiliser notre description formelle de la droite d'Harthong-Reeb comme un modèle du continu pour des applications en géométrie discrète. En effet, ce modèle est bien adapté pour modéliser certains objets continus comme des droites ou des cercles, en utilisant uniquement des ensembles de points à coordonnées entières. Il est possible, en utilisant un procédé d'arithmétisation, de représenter de tels objets continus par des courbes discrètes.

Arithmétisation d'Euler et courbes discrètes

Un avantage de disposer d'une représentation concrète des entiers non-standards est que l'on peut faire des calculs avec. Un exemple d'application pratique est de calculer des fonctions continues uniquement avec des suites d'entiers. C'est ce que nous faisons dans cette section avec les Ω-entiers.

Nous utilisons pour cela le schéma d'Ω-arithmétisation présenté dans [CWF+12]. Un schéma d'arithmétisation est un processus qui permet de construire un équivalent discret à une fonction continue. Nous partons du schéma d'arithmétisation d'Euler :

$$\begin{cases} T_0 = A; X_0 = B \\ T_{k+1} = T_k + \frac{1}{h} \\ X_{k+1} = X_k + \frac{1}{h} \times F(T_k, X_k) \end{cases}$$

Ce schéma calcule une approximation numérique $(T_k, X_k)_{k\geqslant 0}$ d'une fonction continue $X : T \mapsto X(T)$ définie sur un intervalle de $\mathbb{R}$ à valeurs dans $\mathbb{R}$, qui est la solution du problème de Cauchy

```
(* A2_2' *)
Definition weakly_holds4
  (P :Z -> Z -> Z -> Z -> Prop) (a:A) (b:A) (c:A) (d:A) :=
  exists M, forall m, (m >M)%nat -> P (a m) (b m) (c m) (d m).

Lemma R2_2' : forall x y,  y [>]x -> forall z:HRw,
  let xx := proj1_sig x in
  let yy := proj1_sig y in
  let zz := proj1_sig z in
  weakly_holds4
    (fun x y z w => exists q, (q*(y-z)>= w)%Z \/ (q *(z-x)>= w))%Z xx yy zz w.

(* A2_3' *)
Lemma R2_3' : forall x y, HRw_congruent x y -> ~HRwdiff x y ->
HRwequal x y.

(* A3_2' *)
Definition least_upper_bound_weak (S:subset) (tau:HRw) : Prop :=
  (forall v:HRw,  v [>] tau ->
     exists b, upper_bound S b /\ v [>] b /\  b [>=] tau) /\
  (forall mu:HRw, tau [>] mu -> exists o:HRw, S o/\ o [>] mu).

Definition has_least_upper_bound_weak (s:subset) : Prop :=
  exists tau, least_upper_bound_weak s tau.

Lemma least_upper_bound_principle :
  forall S:subset, non_empty S -> bound_above S ->
    (forall alpha beta a b:HRw, upper_bound S beta /\ S alpha /\
       (beta [>=] b /\ b [>=] a /\ a [>=] alpha) ->
       {s:HRw| S s /\ s [>] a}+{upper_bound S b}) ->
          has_least_upper_bound_weak S.
```

FIGURE 4.4 – Enoncés en Coq des trois axiomes alternatifs

suivant : $X^{'} = F(X,T),\ X(A) = B$. L'erreur d'approximation $|X(T_k) - X_k|$ tend vers 0 quand $h \to +\infty$.

En s'inspirant de l'approche de Reeb et Reveillès [RR96], l'idée est de remplacer le nombre h par un Ω-nombre infiniment grand et de traduire toutes les autres quantités en utilisant la fonction $\Psi_\omega : \mathbb{R} \to \mathbb{Z}_\Omega$ telle que $\Psi_\omega(U) = (\lfloor \omega_m \times U \rfloor)_{m\in\mathbb{N}}$ et où ω est un Ω-entier infiniment grand.

L'Ω-arithmétisation du schéma d'Euler selon l'échelle ω est la suivante avec les variables $x_k, t_k \in \mathbb{Z}_\Omega$ (qui sont des suites d'éléments de $\mathbb{Z}$) :

$$\begin{cases} t_0 = a; x_0 = b \\ t_{k+1} = t_k + \beta \\ x_{k+1} = x_k + f(t_k, x_k) \div \beta \end{cases}$$

où β est un Ω-entier positif infiniment grand tel que $\beta^2 = \omega$ et $a = \Psi_\omega(A)$, $b = \Psi_\omega(B)$ et f est une traduction de la fonction F,

$$f(t,x) = (\lfloor \omega_m \times F(t_m/\omega_m, x_m/\omega_m) \rfloor)_{m\in\mathbb{N}}.$$

Ce schéma calcule une suite de points (t_k, x_k) qui correspond au graphe de la fonction discrète $t \mapsto x(t)$, que l'on appelle l'Ω-arithmétisation à l'échelle globale ω de la fonction $T \mapsto X(T)$. Le problème avec cette traduction directe est que le domaine de la fonction $t \mapsto x(t)$ n'est pas connecté puisque $t_{k+1} - t_k = \beta$ qui est infiniment grand. Pour corriger cela, nous utilisons une opération arithmétique de mise à l'échelle $x \mapsto x \div \beta$. Cette fonction envoie β sur 1 et les points (t_k, x_k) sont alors observés à l'échelle intermédiaire β. Afin de calculer l'arithmétisation directement à l'échelle β, nous utilisons le schéma suivant :

$$\begin{cases} \widetilde{t}_0 = a \div \beta, \widetilde{x}_0 = b \div \beta \text{ et } \widehat{x}_0 = b \bmod \beta \\ \widetilde{t}_{k+1} = \widetilde{t}_k + 1 \\ \widetilde{x}_{k+1} = \widetilde{x}_k + (\widehat{x}_k + \widetilde{f}_k) \div \beta \\ \widehat{x}_{k+1} = (\widehat{x}_k + \widetilde{f}_k) \bmod \beta \end{cases}$$

où $\widetilde{x} = x \div \beta$, $\widehat{x} = x \bmod \beta$ et $\widetilde{f}_k = f(\widetilde{t}_k \times \beta + a \bmod \beta, \widetilde{x}_k \times \beta + \widehat{x}_k) \div \beta$. Ce schéma calcule une suite de points $(\widetilde{t}_k, \widetilde{x}_k)$ qui correspond au graphe de la fonction discrète $\widetilde{t} \mapsto \widetilde{x}(\widetilde{t})$ définie sur un domaine connexe puisque $\widetilde{t}_{k+1} - \widetilde{t}_k = 1$. Cette fonction discrète est l'Ω-arithmétisation de la fonction $T \mapsto X(T)$ à l'échelle intermédiaire β.

Les propriétés de l'Ω-arithmétisation sont étudiées en détails dans [CWF+12]. La propriété la plus remarquable est que l'Ω-arithmétisation est une représentation fidèle de la fonction continue $T \mapsto X(T)$, ce qui signifie que la représentation discrète contient autant d'informations que la représentation continue.

Ces résultats nous incitent à expérimenter dans Coq des méthodes de calcul utilisant la description formelle en Coq de la droite d'Harthong-Reeb. Le schéma précédent est implanté en Coq de manière récursive :

```
Fixpoint LS_Euler_stepA
  beta a b (f:A *  A -> A) m : list (A * A * A) :=
  match m with
  | O => (a / beta, b / beta, b mod% beta)::nil
  | (S k) =>
    let r := (LS_Euler_stepA beta a b f k) in
    let '(ttk, xtk, xhk) := hd (a0, a0, a0)  r in
```

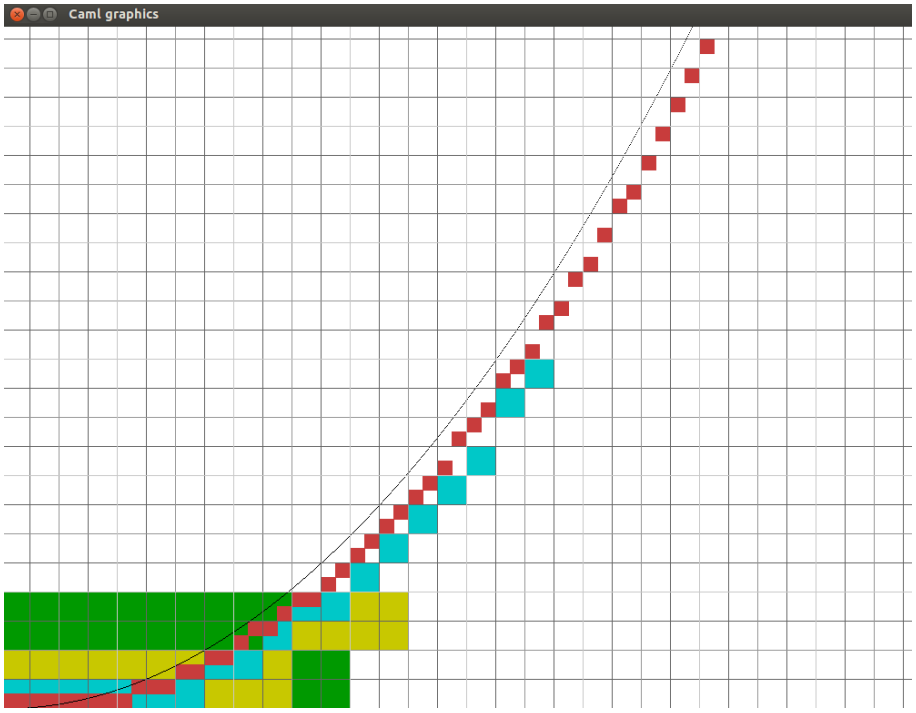


FIGURE 4.5 – L'arithmétisation de la fonction $t \mapsto \frac{t^2}{6}$: les graphes de la fonction $\widetilde{t} \mapsto \widetilde{x}(\widetilde{t})$ sont dessinés seulement pour certains rangs des suites infinies d'entiers.

```
    let ftk :=
      (f ((ttk * beta) + (a mod% beta) , (xtk * beta) + xhk)) / beta
    in
    ( ttk + a1,
     xtk + (xhk + ftk) / beta,
     (xhk + ftk) mod% beta ) :: r
end.
```

Notons que la fonction `hd` qui retourne la tête d'une liste est appelée avec un argument supplémentaire qui sera retournée dans le cas où la fonction est appelée sur la liste vide. Cela permet de s'assurer que la fonction est bien totale comme l'exige le système Coq.

Le contenu calculatoire de la formalisation des Ω-entiers de Laugwitz-Schmieden, y compris la fonction d'arithmétisation, peut être automatiquement extrait vers un programme fonctionnel en OCaml. Les principales commandes d'extraction sont présentées dans la figure 4.6. Le code extrait peut ensuite être connecté à un outil de visualisation qui permet d'observer la fonction continue que nous calculons avec le schéma d'arithmétisation.

L'extraction se contente de supprimer tous les termes sans contenu calculatoire et transforme tous les types dépendants en types non-dépendants qui seront acceptés par le vérificateur de type d'OCaml. Les commandes `Extract Inductive` et `Extract Constant` fournissent un moyen de paramétrer le système en indiquant comment les définitions inductives faites en Coq ainsi que les constantes doivent être implantées concrètement en OCaml. Il n'y a pas de vérification formelle de la pertinence de ces traductions. Il est de la responsabilité de l'utilisateur de s'assurer qu'aucune erreur n'est commise dans cette étape. Une amélioration possible serait de faire les calculs directement dans Coq et de ne conserver que l'interface graphique comme un élément extérieur à Coq. Si l'outil de visualisation pouvait être intégré dans Coq, nous pourrions même éviter complètement d'utiliser l'outil d'extraction.

```
Extraction Language Ocaml.

Extract Inductive prod => "(*)"  [ "(,)" ].
Extract Inductive bool => "bool" ["true" "false"].
Extract Inductive sumbool => "bool" ["true" "false"].

Extract Inductive Z => "int" ["0" "(fun x -> x)" "(fun x -> -x)"].
Extract Inductive positive => "int" ["(fun x -> 2*x+1)" "(fun x -> 2*x)" "1"].
Extract Inductive list => "list" ["([])" "(::)"].
Extract Constant Z_of_nat =>
  "(fun x -> let rec n2b x = match x with O -> 0 | S n -> (n2b n)+1 in n2b x)".

Extract Constant Zcompare =>
  "(fun x y -> if (x=y) then Eq else if (x<y) then Lt else Gt)".
Extract Constant Psucc => "(fun x -> x+1)".
Extract Constant Pplus => "(fun x y -> x+y)".
Extract Constant Pmult => "(fun x y -> x*y)".
Extract Constant Pminus => "(fun x y -> x-y)".
Extract Constant Pdouble_minus_one => "(fun x -> 2*x+1)".
Extract Constant Zdiv_eucl_POS => "(fun x y -> (x/y, x mod y))".
Extract Constant Zdiv_eucl => "(fun x y -> (x/y, x mod y))".

Extract Constant Zplus => "(fun x y -> x+y)".
Extract Constant Zmult => "(fun x y -> x*y)".
Extract Constant Zopp => "(fun x -> -x)".

Extract Constant Zge_bool => "(fun x y -> x >= y)".
Extract Constant Zgt_bool => "(fun x y -> x > y)".

Extract Constant A0_dec_weak_gen => "(fun x n ->
if (x n<0) then (Inleft true) else
if (x n=0) then (Inleft false) else (Inright))".

Extract Constant Zabs => "(fun x -> if x < 0 then -x else x)".
```

FIGURE 4.6 – Version simplifiée des commandes d'extraction

Nous présentons dans la figure 4.5 le résultat de l'arithmétisation de la fonction $t \mapsto \frac{t^2}{6}$. Dans cet exemple, la fonction correspondant à F dans l'algorithme ci-dessus est $F : (T, X) \mapsto \frac{T}{3}$. Comme les Ω-entiers sont des suites infinies d'entiers, ils ne peuvent pas être visualisés directement. Nous devons choisir certains rangs dans les suites considérées pour observer les graphes de la fonction discrète $\widetilde{t} \mapsto \widetilde{x}(\widetilde{t})$ correspondant à ces rangs.

Un travail similaire de description de la méthode d'arithmétisation d'Euler, en s'appuyant sur la droite d'Harthong-Reeb, a été effectué par Jacques Fleuriot dans le contexte de l'assistant de preuve Isabelle/HOL [NPW02]. Ce travail s'appuie sur une bibliothèque d'analyse non-standard déja existante en Isabelle/HOL [FP98]. Une différence importante avec nos travaux est que cette bibliothèque d'analyse non-standard n'est pas constructive. A l'aide des outils à disposition dans cette bibliothèque d'analyse non-standard, Fleuriot formalise la méthode d'arithmétisation d'Eu-

ler et prouve que l'approximation algorithmique produite peut être aussi proche que l'on veut de son homologue continue [Fle11]. Bien que ces travaux proposent uniquement une description mathématique abstraite du problème, là où nous cherchons à construire un formalisme dans lequel on peut effectivement calculer, ils pourraient nous servir de guide pour démontrer formellement, dans notre contexte, que le procédé d'Ω-arithmétisation conduit bien à une représentation fidèle de la fonction continue arithmétisée.

4.5 Une approche alternative avec les nombres B-approximables

En revenant sur les raisons pour lesquelles les entiers de Laugwitz-Schmieden ne peuvent pas être utilisés pour décrire un modèle de la droite d'Harthong-Reeb vérifiant les axiomes de Bridges (et notamment les propriétés R2.2, R2.3 et R3.2), nous constatons que le problème des Ω-entiers est qu'il s'agit d'une structure trop riche où de nombreux Ω-entiers n'ont pas d'interprétation comme des entiers naïfs (ou standards). Certains éléments comme $(-1)^n$ ont des oscillations trop importantes et n'ont pas d'interprétation possible en termes d'entiers, quelque soit le rang à partir duquel on les observe.

Une fois interprétés dans la théorie de la droite d'Harthong-Reeb, ces Ω-entiers font apparaître des éléments qui peuvent être à la fois non nuls et non inversibles. De plus, le modèle s'appuie uniquement sur des propriétés de convergence et on ne contrôle pas la précision des calculs à une échelle (ou un rang) donné(e). Les nombres B-approximables, décrits par Valérie Ménissier-Morain dans sa thèse [MM94] produisent, à une échelle donnée, de meilleures approximations, qui peuvent d'ailleurs être quantifiées.

4.5.1 Définitions et propriétés des nombres B-approximables

Soit $x \in \mathbb{R}$ et $B \geq 2$ un entier. Nous allons écrire x dans la base $(B^{-n})_{n\in\mathbb{Z}}$ afin d'obtenir une suite d'entiers $(x_n)_{n\in\mathbb{Z}}$:

$$\begin{array}{c}(\ldots, x_{-n}, ..., x_{-2}, x_{-1}, x_0, x_1, x_2, ..., x_n, \ldots)\\ \ldots, B^n, ..., B^2,\ B^1, B^0, \frac{1}{B}, \frac{1}{B^2}, ..., \frac{1}{B^n}, \ldots\end{array}$$

Dans cette base, le nombre x s'écrit $\sum_{i\in\mathbb{Z}} x_i B^{-i}$. On s'intéresse alors aux propriétés de la suite $(x_n)_{n\in\mathbb{Z}}$. Tout d'abord, il faut que le nombre réel que l'on veut représenter puisse s'écrire dans la base $(B^{-n})_{n\in\mathbb{Z}}$. C'est pourquoi nous introduisons les nombres B-adiques :

Définition 14. *Soit B un nombre entier ≥ 2. Un nombre rationnel r est dit B-adique s'il existe p, $q > 0$ tels que :*

$$r = \frac{p}{B^q}.$$

Exemple 1. *(a) Soit $B = 4$ et $x = \frac{1}{2}$. Alors x s'écrit $\frac{2}{4^1}$ et est 4-adique avec $p = 2$ et $q = 1$.*

(b) Soit $B = 4$ et $x = \frac{1}{3}$. Pour que x soit 4-adique, il faudrait que l'on ait $B^q = p \times 3$ ce qui est impossible (décomposition en facteurs premiers).

Les nombres B-adiques sont dénombrables et ne représentent qu'une petite partie de $\mathbb{R}$ (qui est l'ensemble que l'on souhaite représenter). Valérie Ménissier-Morain introduit alors la notion de nombres B-approximables :

Définition 15. *Soit $x \in \mathbb{R}$. On dit que x est B-approximable s'il existe une fonction récursive g telle que $\forall N \in \mathbb{Z}$, $g(N)$ est un nombre B-adique fini et :*

$$|x - g(N)| < \frac{1}{B^N}.$$

Par définition des nombres B-approximables, on peut s'en approcher aussi près que l'on souhaite à l'aide d'une fonction récursive (donc implantable). Les nombres B-approximables sont bien plus nombreux que les nombres réels B-adiques et nous allons chercher à montrer que ces nombres B-approximables représentent bien tous les éléments de $\mathbb{R}$. Les nombres B-approximables sont représentés par des suites d'entiers (x_n) croissantes en valeur absolue telles que :

$$\lim_{n->+\infty} \frac{x_n}{B^n} = x.$$

De plus on souhaite que la suite représentant un réel B-approximable x s'en approche de plus en plus au fur et à mesure que l'on augmente dans les indices de la suite. C'est ce qu'exprime la propriété suivante :

Définition 16 (Propriété d'encadrement)**.** *Soit x un réel B-approximable. Si pour tout $n \in \mathbb{Z}$ on a : $(x_n - 1)B^{-n} < x < (x_n + 1)B^{-n}$, alors (x_n) représente x.*

Si une suite vérifie la propriété précédente, alors il est clair que l'on va pouvoir se rapprocher du réel représenté aussi près que l'on souhaite. De plus, si on veut avoir une précision donnée, alors on sait quel élément de la suite on va devoir regarder. Si un entier x_n vérifie la propriété précédente pour un réel x, on dira que x_n vérifie la propriété d'encadrement de x à l'ordre n.

Exemple 2. *Prenons $x = \sqrt{2} \simeq 1,41421$ et $B = 4$. La suite (x_n) dont les premiers termes sont donnés ci-après représente x :*
$x_{-2} = 1$; $x_{-1} = 1$; $x_0 = 1$; $x_1 = 5$; $x_2 = 22$; $x_3 = 90$; $x_4 = 362$; $x_5 = 1448$; $x_6 = 5792$; $x_7 = 23170$
En effet, on a bien les encadrements suivants :
$\frac{x_2-1}{4^2} \simeq 1,31 < \sqrt{2} < 1,44 \simeq \frac{x_1+1}{4^2}$
$\frac{x_3-1}{4^3} \simeq 1,39 < \sqrt{2} < 1,42 \simeq \frac{x_2+1}{4^3}$
$\frac{x_4-1}{4^4} \simeq 1,410 < \sqrt{2} < 1,418 \simeq \frac{x_3+1}{4^4}$
$\frac{x_5-1}{4^5} \simeq 1,4130 < \sqrt{2} < 1,4150 \simeq \frac{x_4+1}{4^5}$
$\frac{x_6-1}{4^6} \simeq 1,4138 < \sqrt{2} < 1,4143 \simeq \frac{x_6+1}{4^6}$
$\frac{x_7-1}{4^7} \simeq 1,41412 < \sqrt{2} < 1,41424 \simeq \frac{x_7+1}{4^7}$

4.5.2 Formalisation en Coq

Les nombres réels calculables définis par Valérie Ménissier-Morain dans sa thèse ont été formalisés initialement en Coq par Jérôme Créci au début des années 2000 [Cré02]. Son développement est d'ailleurs distribué dans les contributions au système Coq [6].

Jérôme Créci a défini formellement les nombres B-approximables et s'est intéressé aux preuves de stabilité des opérations sur ces nombres. Ces propriétés de stabilité ont été établies pour les opérations d'addition, multiplication, opposé et valeur absolue. Notre étude a permis, avec le concours d'Yves Bertot, de mettre en évidence que la formalisation proposée présentait quelques imprécisions. De plus la bibliothèque n'était que partiellement développée. Nous avons terminé les preuves de correction pour l'inverse et la racine carrée. Ce travail est en cours d'intégration dans `coq-community`, projet visant à maintenir et pérénniser les contributions des utilisateurs de Coq.

Concrètement, pour montrer la stabilité de l'ensemble par une opération $\oplus$, il suffit de prendre deux nombres réels B-approximables x et y quelconques dont on suppose qu'ils sont

6. `https://github.com/coq-contribs/exact-real-arithmetic`

représentés respectivement par deux suites $(x_n)_{n\in\mathbb{Z}}$ et $(y_n)_{n\in\mathbb{Z}}$ (c'est à dire des suites qui vérifient la propriété d'encadrement), et de montrer que le nombre $x \oplus y$ peut être représenté par une suite que l'on explicite et qui vérifie elle aussi la propriété d'encadrement (propriété 16).
Formellement en Coq, le type `Reelc` représente les suites d'entiers, `R` l'ensemble des nombres réels tel qu'il est axiomatisé dans Coq, `IZR` correspond à l'injection d'un entier relatif dans `R` et `B_powerRZ` est la fonction de calcul des puissances entières de `B`. Une fois toutes nos notions disponibles, la propriété d'encadrement est formalisée de la façon suivante :

```
Definition encadrement (xc : Reelc) (x : R) : Prop :=
forall n : Z, (IZR (xc n) - 1 < x * B_powerRZ n < IZR (xc n) + 1)%R.
```

Dans ce contexte, les propriétés de stabilité s'énoncent dans la façon suivante :

```
Lemma addition_correct :
 forall (x y : R) (xc yc : Reelc),
 encadrement xc x ->
 encadrement yc y -> encadrement (addition_reelc xc yc) (x + y).

Lemma inverse_correct :
forall (x : R) (xc : Reelc),
x <> 0 -> encadrement xc x -> encadrement (inverse_reelc2 xc) (1 * / x).
```

La correction de la contribution et la formalisation des propriétés d'encadrement pour l'inverse et la racine carrée représente environ 350 lignes de spécifications et 3 500 lignes de preuve. Cela correspond à un doublement de la taille du développement formel par rapport aux travaux initiaux de Jérôme Créci. Les preuves formelles en Coq sont longues, mais elles suivent assez précisement les preuves mathématiques. De nombreuses propriétés, considérées comme triviales en mathématiques sont prouvées automatiquement à l'aide de la tactique `lra` (qui est une procédure de décision pour l'arithmétique linéaire réelle et rationnelle : voir Section 7.6.3 de [Coq19] pour plus de détails).

4.6 Conclusions et perspectives

Dans ce chapitre, nous avons étudié une représentation originale du continu : la droite d'Harthong-Reeb. Nous avons formalisé en Coq cette représentation en utilisant une description axiomatique des entiers non-standards. Nous avons montré que cette représentation vérifie bien tous les axiomes de Bridges et peut donc être considérée comme un corps ordonné de Bridges-Heyting. Nous avons ensuite cherché à construire un modèle calculatoire, s'appuyant sur les entiers de Laugwitz-Schmieden. Dans ce cas, presque tous les axiomes de Bridges peuvent être vérifiés. Néanmoins trois d'entre eux nécessitent un traitement spécial. Ils peuvent être modifiés pour obtenir une représentation alternative du continu, comme le propose Agathe Chollet dans sa thèse. Nous avons également formalisé en Coq une représentation alternative des réels, en utilisant les nombres B-approximables.

Cette formalisation originale du continu nous a montré que les outils mis à disposition dans Coq pour faire des preuves manquent parfois de généricité. C'est le cas notamment des tactiques de décision sur l'arithmétique entière comme la tactique `omega` qui permet de montrer automatiquement des inégalités arithmétiques sur les entiers, qui ne fonctionnent que sur `Z`, et en s'y prenant bien également sur `nat`. Dès lors que l'on utilise un type abstrait (comme le type A des entiers non-standards), il est possible (et facile) de déclarer une structure d'anneau, puis d'utiliser la tactique `ring`. En revanche, il n'existe pas d'équivalent permettant de

tirer avantage d'une tactique comme `omega`. Dans ce cas précis, nous avons réalisé une première étude [Mag17] afin de rendre utilisable une procédure de décision comme celle sous-jacente à `omega` sur d'autres représentations des entiers. Nous avons utilisé les théorèmes de transfert établis par Théo Zimmermann et Hugo Herbelin dans [ZH15] pour rendre la tactique `omega` et d'autes tactiques similaires utilisables sur les entiers `ssrint` de la bibliothèque *Mathematical Components*. D'autres outils facilitant les démonstrations pourraient être envisagés pour simplifier et rendre plus lisibles les preuves d'inégalités. On pourrait imaginer un mécanisme, similaire à l'environnement *calc* présent dans l'assistant de preuve Lean [dMKA+15] pour travailler avec les inégalités de la forme $e_1 < e_n$, qui se démontrent généralement par étapes successives $e_1 < e_2 <= e_3 < \ldots < e_{n-1} < e_n$. Actuellement en Coq, il faut copier-coller les expressions et faire apparaître des expressions arithmétiques parfois complexes dans le script de preuve sous forme d'assertions $e_i < e_{i+1}$.

Du point de vue du développement formel, il nous reste à relier entre eux les deux représentations des réels : celle s'appuyant sur la droite d'Harthong-Reeb et les Ω-entiers de Laugwitz-Schmieden, et celle utilisant les nombres B-approximables. Un axe de recherche supplémentaire serait de relier notre formalisation à la formalisation du calcul réel exact développée par Robbert Krebbers et Bas Spitters [KS11a] à partir de la bibliothèque CoRN *Constructive Real Numbers* [CGW04]. Nous pourrions également montrer que la droite d'Harthong-Reeb vérifie bien les axiomes des réels constructifs tels qu'ils sont décrits dans [KS11b].

Enfin, d'autres applications de la droite d'Harthong-Reeb en géométrie discrète sont envisagées. Nous avons mené quelques travaux préliminaires utilisant la théorie des Ω-entiers pour représenter les applications affines (rotations, homothéties, translations, etc.) de $\mathbb{R}^n$ [ACF+14, Maz19]. De telles applications sont habituellement approchées par des applications discrétisées (à une échelle donnée), appelées applications quasi-affines. En nous appuyant sur la théorie des Ω-entiers et la définition initiale des applications quasi-affines, nous définissons des Ω-applications quasi-affines qui sont les discrétisées *multi-échelle* des applications affines. La description mathématique de ces suites, mettant en jeu une double discrétisation, ainsi que les démonstrations des propriétés de ces nouveaux objets sont techniques et très calculatoires. Les formaliser en Coq permettraient de s'assurer qu'aucune erreur de calcul ne s'est glissée dans la modélisation.

Chapitre 5

Conclusions et Perspectives

Dans ce document, nous avons présenté les principaux travaux que nous avons réalisé ces dernières années. Tous sont en lien avec la géométrie et consistent à formaliser dans Coq des concepts mathématiques et à prouver certaines de leurs propriétés. Dans ce chapitre, nous synthétisons les principaux résultats obtenus. Nous présentons ensuite les perspectives ouvertes par les travaux réalisés. Ces perspectives prennent la forme de nouveaux développements formels envisagés à court ou moyen terme. Nous présentons également notre vision des outils nécessaires pour faciliter l'utilisation d'un système d'aide à la preuve comme Coq, notamment à travers la question de la robustesse et de la maintenance des (scripts de) preuves déjà construit(e)s.

5.1 Conclusions

Nous avons traité de questions de géométrie algorithmique, de questions de modélisation et d'automatisation des preuves en géométrie projective ainsi que de questions en lien avec la représentation exacte des réels en informatique. Dans chaque cas, la modélisation du problème mathématique considéré a évoluée vers la mise en œuvre de solutions informatiques : un calcul d'enveloppe convexe effectif, un outil de preuve par saturation du contexte dans le cadre de l'approche combinatoire de la géométrie projective, ou encore la construction d'un modèle concret sur lequel il est possible de calculer dans le cas de la droite d'Harthong-Reeb.

Nous avons démontré formellement la correction de deux variantes de l'algorithme incrémental de calcul d'enveloppe convexe d'un ensemble de points du plan. Suite à ce travail et aux difficultés rencontrées pour faire des preuves lors de ce premier contact avec la géométrie, nous avons formalisé la correspondance entre deux approches axiomatiques complémentaires pour décrire un fragment très simple de la géométrie : la géométrie projective. Nous avons produit un prouveur automatique s'appuyant sur une approche combinatoire et avons démontré avec celui-ci plusieurs théorèmes emblématiques de la géométrie projective. Enfin, nous avons abordé la question inévitable en géométrie de la représentation des nombres réels. Nous nous sommes intéressés à une description bien adaptée à la géométrie discrète : la droite d'Harthong-Reeb, dont nous avons prouvé que la version axiomatique est bien une représentation discrète du continu. Nous avons également construit un modèle calculatoire utilisant les entiers de Laugwitz-Schmieden. Nous travaillons actuellement pour relier effectivement cette implantation du continu avec d'autres modèles existants.

Ces formalisations ont mis en évidence tous les cas particuliers ou dégénérés à prendre en compte. Elles ont également rendu plus précises certaines descriptions mathématiques, notamment celle de la droite d'Harthong-Reeb ou même la description axiomatique de la géométrie projective en termes de rangs.

Ces travaux ne sont pas une fin en soi. Ils ouvrent des perspectives sur de nouvelles preuves à développer en s'appuyant sur les résultats déjà obtenus. Ils confirment également, si cela était encore à établir, que le système Coq est un système bien adapté pour modéliser des problèmes issus de domaines variés à la frontière de l'informatique et des mathématiques.

5.2 Perspectives sur les preuves à venir

Modélisation géométrique à base topologique

L'approche topologique que nous avons utilisée dans nos travaux sur l'algorithme incrémental de calcul de l'enveloppe convexe dans le plan prend tout son sens dès que l'on passe en 3D ou plus. Les approches naïves utilisables dans le plan (liste de points par exemple) ne sont plus valables et il devient indispensable d'utiliser des structures de données de haut niveau comme les cartes combinatoires pour décrire les objets géométriques et les opérations étudiées. Un premier exemple, dans la lignée des travaux que nous avons réalisés, pourrait être de décrire formellement l'algorithme incrémental de calcul de l'enveloppe convexe d'un ensemble de points en 3D [dBCvKO08, Chap.11] et d'en prouver la correction.

Géométrie projective

Nos travaux sur l'automatisation des preuves en géométrie projective vont se poursuivre, notamment par l'étude de la preuve de la réciproque du théorème de Dandelin-Gallucci et son automatisation. L'automatisation des preuves d'autres théorèmes de géométrie comme le théorème de Pascal ou le théorème de Brianchon pourrait également être étudiée.

Sur les modèles finis de la géométrie projective et notamment les espaces projectifs pg(3,q), nous travaillons à démontrer formellement en Coq des propriétés de décomposition en spreads et packings[1] de ces espaces. Il s'agit, pour commencer, de déterminer tous les spreads puis tous les packings d'un espace projectif fini comme pg(3,2) ou pg(3,3). L'étape suivante est de classifier ces ensembles de spreads et de packings en identifiant ceux qui sont isomorphes. La difficulté principale est l'explosion combinatoire quand le q de pg(3,q) augmente de 1 : il y a 240 packings dans pg(3,2), répartis en 2 classes, alors qu'il y a 73 343 classes de packings dans pg(3,3). L'objectif est d'arriver à décrire formellement les résultats présentés dans [Bet16] sur les packings de pg(3,3) par exemple. Pour cela, il faut être capable de résoudre des problèmes où la taille des données ne permet plus de construire interactivement certains objets et encore moins de faire des preuves interactives avec. Dans ce cas, la spécification du problème ainsi que les scripts de preuve doivent être générés automatiquement et Coq n'est plus utilisé que comme un simple vérificateur de démonstrations, qui accepte ou non le script de preuve fourni. Du point de vue de l'ingénierie de la preuve, nous souhaitons développer des outils accélérant les démonstrations basées sur du raisonnement par cas (introduction d'une notion d'ordre sur les points et les droites, recherche de symétries pour réduire le nombre de cas à traiter). Ces optimisations de haut niveau, combinées à des optimisations de plus bas niveau sur les tactiques de résolution de buts simples devraient nous permettre de gagner un ordre de grandeur supplémentaire dans la classe des problèmes traités, l'idée étant de se rapprocher de ce qui se fait de nos jours par calcul intensif pour vérifier, à terme, des propriétés sur les espaces projectifs finis en dimension supérieure, par exemple pg(3,4) ou pg(3,5) [TZ17, TZ19].

1. Un *spread* est un ensemble de q^2+1 droites disjointes 2 à 2 et qui forment donc une partition de l'ensemble des points. Un packing est un ensemble de q^2+q+1 spreads disjoints 2 à 2 et qui forment donc une partition de l'ensemble des droites.

Enfin le champ d'application de nos techniques pourrait être étendu à des domaines des mathématiques autres que la géométrie, notamment pour ce qui concerne la prise en compte des aspects combinatoires.

Calcul réel exact

Autour du calcul réel exact, nous allons poursuivre nos travaux pour établir une correspondance entre les nombres B-approximables de Valérie Ménissier-Morain et un sous-ensemble des éléments de la droite d'Harthong-Reeb (les éléments réguliers). De manière plus générale, nous souhaitons comparer notre modélisation des réels via la droite d'Harthong-Reeb avec les autres formalisations existantes dans les différents systèmes de preuve (une synthèse de toutes ces formalisations est présentée dans [BLM16]). Nous nous intéresserons tout particulièrement à l'implantation des réels dans le système ACL2 [KM08]. En effet, cette implantation s'appuie également sur l'analyse non-standard [GK01].

5.3 Perspectives sur les outils d'aide à la preuve

Les outils interactifs de preuve formelle permettent de modéliser et ensuite de prouver formellement des propriétés difficiles. En effet, dans le cas des outils interactifs, c'est l'utilisateur qui fait le travail de preuve, le rôle du système est simplement de vérifier que les étapes de raisonnement proposées par l'utilisateur sont correctes. C'est pourquoi il est indispensable d'aider le développeur à construire des preuves formelles. Il s'agit de développer des approches et outils d'ingénierie de la preuve (*proof engineering*) du même calibre que les approches et outils à disposition en génie logiciel (*software engineering*). Cette thématique a été remise au goût du jour récemment par les travaux de Talia Ringer [RYLG18, RSGL20, RPS+19]. Nous présentons ici quelques perspectives sur lesquelles nous allons travailler pour produire de nouveaux outils d'aide à la preuve.

Intégration d'outils externes dans Coq

Dans le cas de notre prouveur automatique en géométrie projective, la génération des scripts de preuve se fait actuellement à l'extérieur du système de preuve Coq et le script est ensuite fourni à Coq qui le valide. Une piste d'amélioration serait de générer les objets et les preuves directement dans le *plug-in* pour Coq écrit en OCaml. Dans ce cas, notre prouveur automatique pourrait être utilisé directement dans le système interactif, permettant à l'utilisateur de faire certaines parties de sa preuve automatiquement tout en gardant la possibilité (dans les cas les plus complexes ou quand le prouveur automatique échoue) de faire certaines parties à la main.

Automatisation des démonstrations

En reprenant des idées issues de nos travaux de thèse sur les changements de représentation des données [MB02, Mag03], nous avons récemment travaillé sur un outil [Mag17] permettant d'utiliser automatiquement les tactiques implantant des procédures de décision sur l'arithmétique (par exemple `omega` ou `lia`) sur la structure de données décrivant les entiers relatifs dans la bibliothèque *Mathematical Components* [MT16]. Les entiers relatifs sont décrits dans cette bibliothèque comme le type des entiers naturels et leur miroir (pour les nombres négatifs). Cette représentation est isomorphe à la représentation habituelle des entiers relatifs avec le type inductif `Z` de Coq, néanmoins les tactiques fournies dans Coq ne sont pas suffisamment génériques pour être utilisées avec cette structure de données spécifique. Nous avons utilisé l'outil de transfert de théorèmes proposé par Zimmermann et Herbelin [ZH15] pour nous ramener à des énoncés

sur les élements du type inductif `Z`. Dans la suite de ce travail, nous souhaitons pouvoir rendre utilisables les procédures de décision sur des sous-ensembles des entiers de Laugwitz-Schemieden par exemple.

Maintenance et pérénnité des développements formels

Tous les travaux que nous avons présenté dans ce document contiennent de longs et parfois délicats scripts de preuves en Coq. Une fois terminés, ces scripts sont archivés localement ou dans les contributions du système Coq. Ces développements formels ont alors tendance à être oubliés, puis un jour, on souhaite les réutiliser de nouveau ou du moins en prendre connaissance avec la dernière version publique de Coq. Souvent le développement ne *compile* plus. Les raisons sont souvent bénignes (léger changement de sémantique d'une tactique, modification d'un énoncé de la bibiliothèque standard, etc.), mais il est généralement très coûteux de remettre le développement au goût du jour. Des outils permettant de mémoriser le déroulement des preuves avec notamment la visualisation des étapes de raisonnement ont été proposés au début des années 2010, par exemple l'infrastructure Proviola [TGMW10]. L'équipe de développement de Coq maintient également à disposition un grand nombre de versions historiques de Coq, permettant de recompiler aisément le développement. Cependant, si l'on souhaite utiliser deux développements créés dans deux versions différentes de Coq et que l'on veut de plus utiliser les outils mis à disposition par les versions les plus récentes de Coq, la première tâche à faire est de porter ces développements vers la dernière version de Coq. Ce travail laborieux pourrait être simplifié si l'on disposait d'un système de *post-processing* des scripts de preuve, visant à rendre les preuves plus robustes. Une fois construit les scripts pourraient être modifiés pour intégrer de la structure en utilisant les accolades ({ et }), les variables introduites automatiquement pourraient être nommées explicitement quand elles sont utilisées ensuite dans le raisonnement. Afin de conserver une certaine cohérence dans les démonstrations, des morceaux de script comme `apply T; try omega.` pourraient être réécrits en rendant explicite le fait que la tactique `omega` ne démontre que les buts 1 et 3 par exemple. Le code précédent serait alors remplacé par `apply T; [omega | idtac | omega | idtac]` . En plus d'édicter des règles de bonne pratique pour le développement et la rédaction des scripts de preuve, il est en effet indispensable de proposer des outils permettant d'imposer ces bonnes pratiques, dans le but de pérénniser les développements formels et d'en permettre plus facilement la réutilisation.

Bibliographie

[ABC+16] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti : a logical framework based on the $\lambda - \Pi$-calculus modulo theory. Manuscript `http://www.lsv.fr/~dowek/Publi/expressing.pdf`, 2016.

[ACF+14] Éric Andres, Marie-Andrée Da Col, Laurent Fuchs, Gaëlle Largeteau-Skapin, Nicolas Magaud, Loïc Mazo, and Rita Zrour. Les Ω-AQA : Représentation discrète des applications affines. In Nicolas Passat, editor, *Neuvièmes journées du Groupe de Travail de Géométrie Discrète (GéoDis 2014)*, Nov. 2014. Présenté à ReimsImage'2014 `http://reimsimage2014.univ-reims.fr/geodis-2014/sessions/`.

[AFG+11] Mickaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. 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.

[Bat97] Lynn Margaret Batten. *Combinatorics of Finite Geometries*. Cambridge Univ. Press, 1997.

[BB12] Mathieu Boespflug and Guillaume Burel. CoqInE : Translating the calculus of inductive constructions into the λΠ-calculus modulo. In *Second International Workshop on Proof Exchange for Theorem Proving (PXTP 2012)*, Manchester, United Kingdom, June 2012.

[BBB+18] Michael Beeson, Pierre Boutry, Gabriel Braun, Charly Gries, and Julien Narboux. GeoCoq, June 2018.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art : The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin/Heidelberg, May 2004. 469 pages.

[BDM12a] Christophe Brun, Jean-François Dufourd, and Nicolas Magaud. Designing and Proving Correct a Convex Hull Algorithm with Hypermaps in Coq. *Computational Geometry, Theory and Applications*, 45(8) :436–457, 2012.

[BDM12b] Christophe Brun, Jean-François Dufourd, and Nicolas Magaud. Formal Proof in Coq and Derivation of a Program in C++ to Compute Convex Hulls. In Tetsuo Ida and Jacques D. Fleuriot, editors, *Automated Deduction in Geometry (ADG'2012) - Revised Selected Papers*, volume 7993 of *LNCS*, pages 71–88. Springer, 2012. ISBN 978-3-642-40671-3.

[Bet16] Anton Betten. The packings of pg(3, 3). *Designs, Codes and Cryptography*, 79(3) :583–595, 2016.

[BLM16] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis : a survey of proof assistants and libraries. *Math. Struct. Comput. Sci.*, 26(7) :1196–1233, 2016.

[BM15] David Braun and Nicolas Magaud. Des preuves formelles en Coq du théorème de Thalès pour les cercles. In David Baelde and Jade Alglave, editors, *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA2015)*, Jan. 2015.

[BMS18] David Braun, Nicolas Magaud, and Pascal Schreck. Formalizing Some "Small" Finite Models of Projective Geometry in Coq. In Jacques Fleuriot, Dongming Wang, and Jacques Calmet, editors, *Proceedings of Artificial Intelligence and Symbolic Computation 2018 (AISC'2018)*, number 11110 in LNAI, pages 54–69, Sept. 2018.

[BMS19] David Braun, Nicolas Magaud, and Pascal Schreck. Two Cryptomorphic Formalizations of Projective Incidence Geometry. *Annals of Mathematics and Artificial Intelligence*, 85(2–4) :193–212, 2019.

[BMS20] David Braun, Nicolas Magaud, and Pascal Schreck. A Generic Automatic Prover for Projective Incidence Geometry and its Interaction with the Coq Proof Assistant. *En préparation*, 2020.

[BOB92] Michael Blair, Sally Obenski, and Paula Bridickas. Patriot missile defense : Software problem led to system failure at dhahran, saudi arabia. Technical Report GAO/IMTEC-92-26, GAO/IMTEC, February 1992.

[BP15] John Bamberg and Tim Penttila. Completing Segre's Proof of Wedderburn's Little Theorem. *Bulletin of the London Mathematical Society*, 47(3) :483–492, 2015.

[BR97] Douglas Bridges and Steve Reeves. Constructive mathematics, in theory and programming practice. Technical Report CDMTCS-068, Centre for Discrete Mathematics and Theorical Computer Science, November 1997.

[Bra19] David Braun. *Approche combinatoire pour l'automatisation en Coq des preuves formelles en géométrie d'incidence projective*. PhD thesis, Université de Strasbourg, sept. 2019.

[Bri99] Douglas Bridges. Constructive mathematics : A foundation for computable analysis. *Theor. Comput. Sci.*, 219(1-2) :95–109, 1999.

[Bue95a] Francis Buekenhout. Chapter 1 - An Introduction to Incidence Geometry. In F. Buekenhout, editor, *Handbook of Incidence Geometry*, pages 1–25. North-Holland, Amsterdam, 1995.

[Bue95b] Francis Buekenhout, editor. *Handbook of Incidence Geometry*. North Holland, 1995.

[Bue95c] Buekenhout, Francis. *Handbook of Incidence Geometry : buildings and foundations*. Elsevier, 1995.

[BY98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998. 544 pages. Translated by Hervé Brönnimann.

[CGW04] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-corn, the constructive coq repository at nijmegen. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *Mathematical Knowledge Management, Third International Conference, MKM 2004, Bialowieza, Poland, September 19-21, 2004, Proceedings*, volume 3119 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2004.

[Chl13] Adam Chlipala. *Certified Programming with Dependent Types : A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press, 2013.

[Cho10] Agathe Chollet. *Formalismes non classiques pour le traitement informatique de la topologie et de la géométrie discrète*. PhD thesis, Université de la Rochelle, 2010.

[CKK+12] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c - A software analysis perspective. In George Eleftherakis, Mike Hinchey, and Mike Holcombe, editors, *Software Engineering and Formal Methods - 10th International Conference, SEFM 2012, Thessaloniki, Greece, October 1-5, 2012. Proceedings*, volume 7504 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2012.

[Coq19] Coq development team. *The Coq Proof Assistant Reference Manual, Version 8.9.* INRIA, 2019.

[Cox03] Harold Scott Macdonald Coxeter. *Projective Geometry.* Springer Science & Business Media, 2003.

[Cré02] Jérôme Créci. Certification d'algorithmes d'arithmétique réelle exacte dans le système Coq. Rapport de DEA, Université Paris-Sud, Orsay, France, September 2002.

[CWF+09] Agathe Chollet, Guy Wallet, Laurent Fuchs, Gaëlle Largeteau-Skapin, and Eric Andres. Insight in discrete geometry and computational content of a discrete model of the continuum. *Pattern recognition*, 42 :2220–2228, 2009.

[CWF+12] Agathe Chollet, Guy Wallet, Laurent Fuchs, Eric Andres, and Gaëlle Largeteau-Skapin. Foundational Aspects of Multiscale Digitization. *Theor. Comput. Sci.*, 466 :2–19, 2012.

[DB10] Jean-François Dufourd and Yves Bertot. Formal Study of Plane Delaunay Triangulation. In *Interactive Theorem Proving*, 2010. LNCS, 6172 :211–226, Springer, 2010.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry, Algorithms and Applications (Third Edition).* Springer-Verlag, Berlin/Heidelberg, March 2008. 386 pages.

[DD04a] Christophe Dehlinger and Jean-François Dufourd. Formalizing the generalized maps in Coq. *Theoretical Computer Science*, 323(1-3) :351–397, September 2004.

[DD04b] Christophe Dehlinger and Jean-François Dufourd. Formalizing the Trading Theorem in Coq. *Theoretical Computer Science*, 323(1-3) :399–442, September 2004.

[DFKT14] Jan Duracz, Amin Farjudian, Michal Konecný, and Walid Taha. Function interval arithmetic. In Hoon Hong and Chee Yap, editors, *Mathematical Software - ICMS 2014 - 4th International Congress, Seoul, South Korea, August 5-9, 2014. Proceedings*, volume 8592 of *Lecture Notes in Computer Science*, pages 677–684. Springer, 2014.

[Die92] Marc Diener. Application du calcul de Harthong-Reeb aux routines graphiques. In J.-M. Salanskis and H. Sinaceurs, editors, *Le Labyrinthe du Continu*, pages 424–435. Springer, 1992.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3 : An Efficient SMT Solver. In *Proceedings of TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[dMKA+15] Leonardo Mendonça 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 *LNCS*, pages 378–388. Springer, 2015.

[DR89] Francine Diener and Georges Reeb. *Analyse Non Standard.* Hermann, Paris, 1989.

[dt04] CGAL development team. *The Computational Geometry Algorithms Library (CGAL) Manual 3.1*, 2004. Available from http ://www.cgal.org.

[Duf09] Jean-François Dufourd. An Intuitionistic Proof of a Discrete Form of the Jordan Curve Theorem Formalized in Coq with Combinatorial Hypermaps. *Journal of Automated Reasoning*, 43(1) :19–51, 2009.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, New York, NY, USA, 1987.

[Edm71] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1) :127–136, 1971.

[Fle11] Jacques Fleuriot. Exploring the foundations of discrete analytical geometry in Isabelle/HOL. In Pascal Schreck, Jürgen Richter-Gebert, and Julien Narboux, editors, *Proceedings of Automated Deduction in Geometry 2010*, volume 6877 of *LNAI*. Springer, 2011.

[For86] Steven Fortune. A sweepline algorithm for voronoi diagrams. In Alok Aggarwal, editor, *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry, Yorktown Heights, NY, USA, June 2-4, 1986*, pages 313–322. ACM, 1986.

[FP98] Jacques D. Fleuriot and Lawrence C. Paulson. A combination of nonstandard analysis and geometry theorem proving, with application to newton's principia. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction - CADE-15, 15th International Conference on Automated Deduction, Lindau, Germany, July 5-10, 1998, Proceedings*, volume 1421 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 1998.

[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, Christine Paulin, and David Pichardie, editors, *ITP 2013, 4th Conference on Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179, Rennes, France, July 2013. Springer.

[GK01] Ruben Gamboa and Matt Kaufmann. Nonstandard analysis in ACL2. *J. Autom. Reasoning*, 27(4) :323–351, 2001.

[GN00] Herman Geuvers and Milad Niqui. Constructive reals in coq : Axioms and categoricity. In Paul Callaghan, Zhaohui Luo, James McKinna, and Robert Pollack, editors, *Types for Proofs and Programs, International Workshop, TYPES 2000, Durham, UK, December 8-12, 2000, Selected Papers*, volume 2277 of *Lecture Notes in Computer Science*, pages 79–95. Springer, 2000.

[Gon08] Georges Gonthier. Formal Proof - The Four-Colour Theorem. *Notices of the AMS*, 55(11) :1382–1393, 2008.

[Gra72] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4) :132–133, 1972.

[HAB+15] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason M. Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture. *CoRR*, abs/1501.02155, 2015.

[Hal43] Marshall Hall. Projective planes. *Transactions of the American Mathematical Society*, 54(2) :229–277, 1943.

[Hen25] Henry Frederick Baker. *Principles of Geometry*, volume 1. Cambridge University Press, 1925.

[Hor17] Ákos G. Horváth. Gallucci's axiom revisited, 2017.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science*. Cambridge University Press, 2nd edition edition, 2004.

[IEE19] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[IGG] IGG Team. CGoGN : Combinatorial and Geometric mOdeling with Generic N-dimensional Maps. `https://iggservis.u-strasbg.fr/CGoGN/`.

[Jac70] Alain M. Jacques. Constellations et Graphes topologiques. In *Combinatorial Theory Symposium*, pages 653–673, 1970.

[Jar73] Ray A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inf. Process. Lett.*, 2(1) :18–21, 1973.

[JLB+15] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. A formally-verified C static analyzer. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 247–259. ACM, 2015.

[JR17] Claude-Pierre Jeannerod and Nathalie Revol. Analyser et encadrer les erreurs dues à l'arithmétique flottante. In *Informatique mathématique : une photographie en 2017*, pages 115–144. CNRS Editions, 2017. ISSN 978-2-271-11523-2.

[KAE+10] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4 : formal verification of an operating-system kernel. *Commun. ACM*, 53(6) :107–115, 2010.

[KM08] Matt Kaufmann and J Strother Moore. An ACL2 tutorial. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 17–21. Springer, 2008.

[KMP+04] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom Examples of Robustness Problems in Geometric Computations. In *European Symposium on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 702–713, Bergen, Norway, September 2004. Springer.

[Knu92] Donald Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin/Heidelberg, 1992. 109 pages.

[KO09] Cezary Kaliszyk and Russell O'Connor. Computing with classical real numbers. *J. Formalized Reasoning*, 2(1) :27–39, 2009.

[KS11a] Robbert Krebbers and Bas Spitters. Computer certified efficient exact reals in coq. In James H. Davenport, William M. Farmer, Josef Urban, and Florian Rabe, editors, *Intelligent Computer Mathematics - 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings*, volume 6824 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 2011.

[KS11b] Robbert Krebbers and Bas Spitters. Type classes for efficient exact real arithmetic in coq. *Logical Methods in Computer Science*, 9(1), 2011.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *International Conference on Computer Aided Verification*, pages 1–35. Springer, 2013.

[Lal90] R. Lalement. *Logique, réduction, résolution.* Études et recherches en informatique. Masson, Paris, 1990.

[Lau83] Detlef Laugwitz. Ω-calculus as a generalization of field extension an alternative approach to nonstandard analysis. In A. Hurd, editor, *Nonstandard Analysis - Recent developments*, volume 983 of *Lecture Notes in Mathematics*, pages 120–133. Springer, 1983.

[Lau92] Detlef Laugwitz. Leibniz' principle and Ω-calculus. In J.M. Salanskis and H. Sinacoeur, editors, *Le Labyrinthe du Continu*, pages 144–155. Springer France, 1992.

[Ler09a] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7) :107–115, 2009.

[Ler09b] Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4) :363–446, 2009.

[LP18] Leonidas Lampropoulos and Benjamin C. Pierce. *QuickChick : Property-Based Testing in Coq.* Software Foundations series, volume 4. Electronic textbook, 2018. Version 1.0. `http://www.cis.upenn.edu/~bcpierce/sf`.

[LS58] Detlef Laugwitz and Curt Schmieden. Eine erweiterung der infinitesimalrechnung. *Mathematische Zeitschrift*, 89 :1–39, 1958.

[Lut92] Robert Lutz. La force modélisatrice des théories infinitésimales faibles. In J-M. Salanskis and H. Sinaceur, editors, *Le Labyrinthe du Continu*, pages 414–423. Springer-Verlag, 1992.

[Mag03] Nicolas Magaud. Changing Data Representation within the Coq System. In *TPHOLs'2003*, volume 2758 of *LNCS*. Springer-Verlag, 2003.

[Mag17] Nicolas Magaud. Transferring Arithmetic Decision Procedures (on Z) to Alternative Representations. In Emilio Jesús Gallego Arias and Sandrine Blazy, editors, *Proceedings of the CoqPL workshop 2017*, 2017.

[May01] Micaela Mayero. *Formalisation et automatisation de preuves en analyses réelle et numérique.* PhD thesis, Université Paris VI, décembre 2001.

[Maz19] Loïc Mazo. Multi-scale arithmetization of linear transformations. *Journal of Mathematical Imaging and Vision*, 61(4) :432–442, 2019.

[MB02] Nicolas Magaud and Yves Bertot. Changing data structures in type theory :a study of natural numbers. In Paul Callaghan, Zhaohui Luo, James McKinna, and Randy Pollack, editors, *Post-proceedings of TYPES'2000*, volume 2277 of *LNCS*. Springer-Verlag, 2002.

[MCF14] Nicolas Magaud, Agathe Chollet, and Laurent Fuchs. Formalizing a discrete model of the continuum in coq from a discrete geometry perspective. *Annals of Mathematics and Artificial Intelligence*, 74(3-4) :309–332, 2014.

[MF06] Laura Meikle and Jacques Fleuriot. Mechanical Theorem Proving in Computational Geometry. In Hoon Hong and Dongming Wang, editors, *Automated Deduction in Geometry*, volume 3763 of *Lecture Notes in Computer Science*, pages 1–18, Berlin/Heidelberg, 2006. Springer. 5th International Workshop, ADG 2004, Gainesville, FL, USA, September 16-18, 2004.

[ML84] Per Martin-Löf. *Intuitionnistic Type Theory.* Bibliopolis, Napoli, 1984.

[ML90] Per Martin-Löf. Mathematics of infinity. In *COLOG-88 Computer Logic*, Lecture Notes in Computer Science, pages 146–197. Springer-Verlag Berlin, 1990.

[MM94] Valérie Ménissier-Morain. *Arithmétique exacte - Conception, algorithmique et performances d'une implémentation informatique en précision arbitraire*. PhD thesis, Université Paris VII, 1994.

[MM05] Valérie Ménissier-Morain. Arbitrary Precision Real Arithmetic : Design and Algorithms. *J. Log. Algebr. Program.*, 64(1) :13–39, 2005.

[MNS08] Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing Projective Plane Geometry in Coq. In Thomas Sturm, editor, *Post-proceedings of ADG'2008*, volume 6301 of *LNAI*. Springer, 2008.

[MNS09] Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing Desargues' Theorem in Coq using Ranks. In *Proceedings of the ACM Symposium on Applied Computing SAC 2009*, pages 1110–1115. ACM, ACM Press, March 2009.

[MNS12] Nicolas Magaud, Julien Narboux, and Pascal Schreck. A Case Study in Formalizing Projective Geometry in Coq : Desargues Theorem. *Computational Geometry : Theory and Applications*, 45(8) :406–424, 2012.

[Moo62] Ramon E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. PhD thesis, Department of Computer Science, Stanford University, 1962.

[Mou02] Forest Ray Moulton. A Simple Non-Desarguesian Plane Geometry. *Transactions of the American Mathematical Society*, 3(2) :192–195, 1902.

[MS06] Dominique Michelucci and Pascal Schreck. Incidence Constraints : a Combinatorial Approach. *Int. Journal of Computational Geometry and Applications*, 16(5-6) :443–460, 2006.

[MT16] Assia Mahboubi and Enrico Tassi. *Mathematical Components*. Draft, 2016.

[Mül00] Norbert Th. Müller. The irram : Exact arithmetic in C++. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis, 4th International Workshop, CCA 2000, Swansea, UK, September 17-19, 2000, Selected Papers*, volume 2064 of *Lecture Notes in Computer Science*, pages 222–252. Springer, 2000.

[Nel77] Edward Nelson. Internal set theory : A new approach to nonstandard analysis. *Bulletin of the American Mathematical Society*, 83(6) :1165–1198, November 1977.

[Nel87] Edward Nelson. *Radically Elementary Theory*. Annals of Mathematics Studies. Princeton University Press, 1987.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Oxl06] James G Oxley. *Matroid Theory*, volume 3. Oxford University Press, USA, 2006.

[PB01] David Pichardie and Yves Bertot. Formalizing Convex Hull Algorithms. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics*, volume 2152 of *Lecture Notes in Computer Science*, pages 346–361, Berlin/Heidelberg, 2001. Springer. 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001.

[PD00] François Puitg and Jean-François Dufourd. Formalizing mathematics in higher-order logic : A case study in geometric modelling. *Theor. Comput. Sci.*, 234(1-2) :1–57, 2000.

[PNVM12] Bruno Cuervo Parrino, Julien Narboux, Eric Violard, and Nicolas Magaud. Dealing with arithmetic overflows in the polyhedral model. In Uday Bondhugula and Vincent Loechner, editors, *2nd International Workshop on Polyhedral Compilation Techniques (IMPACT'2012)*, Jan. 2012. Accepted for presentation at the workshop.

[PS93] Franco P. Preparata and Michael I. Shamos. *Computational Geometry : An Introduction (5th printing)*. Monographs in Computer Science. Springer-Verlag, New York, 1993. 398 pages.

[Rev91] Jean-Pierre Reveillès. *Géométrie discrète, calcul en nombres entiers et algorithmique*. Université Louis Pasteur, Strasbourg, 1991.

[RPS+19] Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric, and Zachary Tatlock. Qed at large : A survey of engineering of formally verified software. *Foundations and Trends in Programming Languages*, 5(2-3) :102–281, 2019.

[RR96] Jean-Pierre Reveillès and Denis Richard. Back and forth between continuous and discrete for the working computer scientist. *Annals of Mathematics and Artificial Intelligence, Mathematics and Informatic*, 16(1-4) :89–152, 1996.

[RR06] Eugenio Roanes-Macías and Eugenio Roanes-Lozano. A maple package for automatic theorem proving and discovery in 3d-geometry. In Francisco Botana and Tomás Recio, editors, *Automated Deduction in Geometry, 6th International Workshop, ADG 2006, Pontevedra, Spain, August 31-September 2, 2006. Revised Papers*, volume 4869 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2006.

[RSGL20] Talia Ringer, Alex Sanchez-Stern, Dan Grossman, and Sorin Lerner. Replica : REPL instrumentation for coq analysis. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 99–113. ACM, 2020.

[RYLG18] Talia Ringer, Nathaniel Yazdani, John Leo, and Dan Grossman. Adapting proof automation to adapt proofs. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 115–129. ACM, 2018.

[Sut09] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure. *Journal of Automated Reasoning*, 43(4) :337, 2009.

[TG15] Tobias Tebbi and Jason Gross. A Profiler for Ltac. In *Coq PL Workshop 2015*, 2015.

[TGMW10] Carst Tankink, Herman Geuvers, James McKinna, and Freek Wiedijk. Proviola : A tool for proof re-animation. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *Intelligent Computer Mathematics, 10th International Conference, AISC 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, MKM 2010, Paris, France, July 5-10, 2010. Proceedings*, volume 6167 of *Lecture Notes in Computer Science*, pages 440–454. Springer, 2010.

[Tut59] William Thomas Tutte. Matroids and graphs. *Transactions of the American Mathematical Society*, 90(3) :527–552, 1959.

[TZ17] Svetlana Topalova and Stela Zhelezova. New parallelisms of pg(3, 4). *Electronic Notes in Discrete Mathematics*, 57 :193–198, 2017.

[TZ19] Svetlana Topalova and Stela Zhelezova. Types of spreads and duality of the parallelisms of pg(3, 5) with automorphisms of order 13. *Des. Codes Cryptogr.*, 87(2-3) :495–507, 2019.

[VY18] Oswald Veblen and John Wesley Young. *Projective geometry*, volume 2. Boston Ginn, 1918.

[YYD+10] Jihun Yu, Chee Yap, Zilin Du, Sylvain Pion, and Hervé Brönnimann. The design of core 2 : A library for exact numeric computation in geometry and algebra. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010, Third International Congress on Mathematical Software, Kobe, Japan, September 13-17, 2010. Proceedings*, volume 6327 of *Lecture Notes in Computer Science*, pages 121–141. Springer, 2010.

[ZG15] Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal planar lambda terms. *Logical Methods in Computer Science*, 11(3), 2015.

[ZH15] Théo Zimmermann and Hugo Herbelin. Automatic and transparent transfer of theorems along isomorphisms in the coq proof assistant. Conference on Intelligent Computer Mathematics (CICM 2015), May 2015.

[Zim10] Paul Zimmermann. Reliable computing with GNU MPFR. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010, Third International Congress on Mathematical Software, Kobe, Japan, September 13-17, 2010. Proceedings*, volume 6327 of *Lecture Notes in Computer Science*, pages 42–45. Springer, 2010.

Annexe A

Liste des développements réalisés et disponibles en ligne

Nous donnons ici les liens vers les développements formels faits en Coq des différents travaux présentés dans ce mémoire.

- Algorithme incrémental de calcul de l'enveloppe convexe d'un ensemble de points du plan
  - V1 : calcul par récursion structurelle sur la structure inductive des cartes
    `https://github.com/magaud/ConvexHullV1`
  - V2 : calcul par récursion générale en suivant la forme de l'enveloppe convexe
    `https://github.com/magaud/ConvexHullV2`
- Formalisation de la géométrie projective
  - Preuve du théorème de Desargues en utilisant l'approche combinatoire
    `https://github.com/ProjectiveGeometry/ProjectiveGeometry/tree/master/Dev`
  - Equivalence des approches synthétique et combinatoire
    `https://github.com/ProjectiveGeometry/ProjectiveGeometry/tree/amai2018`
  - Etude sur les plans et espaces projectifs finis
    `https://github.com/ProjectiveGeometry/ProjectiveGeometry/tree/aisc2018`
  - Implantation du prouveur automatique basé sur les rangs et cas d'études
    `https://github.com/ProjectiveGeometry/ProjectiveGeometry/tree/master/Prog/Theorem_rk_solver`
- Représentation de nombres réels
  - La droite d'Harthong-Reeb, du point de vue axiomatique
    `https://dpt-info.u-strasbg.fr/~magaud/Harthong-Reeb/index.html`
  - La droite d'Harthong-Reeb, du point de vue calculatoire
    `https://github.com/magaud/HR`
  - Formalisation des nombres réels exacts [MM94] (en cours de mise à jour)
    `https://github.com/coq-community/exact-real-arithmetic`
  - Etude de la correspondance entre ces deux représentations
    `https://github.com/magaud/HRBnVMM`
- Autres travaux
  - Outil de transfert des propriétés sur `Z` vers les propriétés homologues sur `ssrint`
    `https://dpt-info.di.unistra.fr/~magaud/SSR/`
  - Des preuves du théorème de Thalès pour les cercles
    `http://ddbraun.free.fr/Thales/`

Annexe B

Liste des publications

Cette annexe regroupe les travaux les plus significatifs publiés au cours de ma carrière d'enseignant-chercheur.

Revues internationales

[2019] David Braun, Nicolas Magaud, and Pascal Schreck. Two Cryptomorphic Formalizations of Projective Incidence Geometry. *Annals of Mathematics and Artificial Intelligence*, 85(2-4) : 193–212, 2019.

[2015a] Nicolas Magaud, Agathe Chollet, and Laurent Fuchs. Formalizing a Discrete Model of the Continuum in Coq from a Discrete Geometry Perspective. *Annals of Mathematics and Artificial Intelligence*, 74(3–4) :309–332, 2015.

[2012a] Nicolas Magaud, Julien Narboux, and Pascal Schreck. A Case Study in Formalizing Projective Geometry in Coq : Desargues Theorem. *Computational Geometry : Theory and Applications*, 45(8) :406–424, 2012.

[2012b] Christophe Brun, Jean-François Dufourd, and Nicolas Magaud. Designing and Proving Correct a Convex Hull Algorithm with Hypermaps in Coq. *Computational Geometry : Theory and Applications*, 45(8) :436–457, 2012.

[2002] Yves Bertot, Nicolas Magaud, and Paul Zimmermann. A Proof of GMP Square Root. *Journal of Automated Reasoning*, 29(3-4) :225–252, 2002. Special Issue on Automating and Mechanising Mathematics : In honour of N.G. de Bruijn.

Conférences et workshops internationaux

[2018a] David Braun, Nicolas Magaud, and Pascal Schreck. Formalizing Some "Small" Finite Models of Projective Geometry in Coq. In Jacques D. Fleuriot and Dongming Wang and Jacques Calmet, editors, *Artificial Intelligence and Symbolic Computation - 13th International Conference, AISC 2018, Proceedings*, pages 54–69, LNCS 11110, Springer, Sept. 2018.

[2017] Nicolas Magaud. Transferring Arithmetic Decision Procedures (on Z) to Alternative Representations. In S. Blazy, and Emilio Jesús Gallego Arias, editors, *CoqPL 2017 : The Third International Workshop on Coq for Programming Languages informal proceedings*, Jan. 2017.

[2012c] Bruno Cuervo Parrino, Julien Narboux, Eric Violard, and Nicolas Magaud. Dealing with Arithmetic Overflows in the Polyhedral Model. In U. Bondhugula and V. Loechner, editors,

2nd International Workshop on Polyhedral Compilation Techniques (IMPACT'2012), Jan. 2012.

[2012d] Christophe Brun, Jean-François Dufourd, and Nicolas Magaud. Formal Proof in Coq and Derivation of an Imperative Program to Compute Convex Hulls. In Tetsuo Ida and Jacques D. Fleuriot, editors, *Automated Deduction in Geometry - 9th International Workshop, ADG 2012. Revised Selected Papers*, pages 71–88, LNCS 7993, Springer, Sept. 2012.

[2009] Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing Desargues' Theorem in Coq using Ranks. In *Proceedings of the ACM Symposium on Applied Computing SAC 2009*, pages 1110–1115. ACM, ACM Press, March 2009.

[2008] Nicolas Magaud, Julien Narboux, and Pascal Schreck. Formalizing Projective Plane Geometry in Coq. In T. Sturm, editor, *Post-proceedings of Automated Deduction in Geometry (ADG'2008)*, volume 6301 of *LNAI*. Springer, 2008.

[2003] Nicolas Magaud. Changing Data Representation within the Coq System. In *Theorem Proving in Higher Order Logics (TPHOLs'2003)*, volume 2758 of *LNCS*. Springer-Verlag, 2003.

[2000] Nicolas Magaud and Yves Bertot. Changing data structures in type theory :a study of natural numbers. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Post-proceedings of TYPES'2000*, volume 2277 of *LNCS*. Springer-Verlag, 2002.

[1999] Jean-Christophe Filliâtre and Nicolas Magaud. Certification of Sorting Algorithms in the Coq System. In *Theorem Proving in Higher Order Logics : Emerging Trends*, 1999.

Conférences francophones et groupes de travail

[2019] Nicolas Magaud and Zaynah Dargaye. Journées Francophones des Langages Applicatifs 2019. Edition des actes des JFLA 2019. Jan 2019, Les Rousses, France.

[2018b] Sylvie Boldo and Nicolas Magaud. Journées Francophones des Langages Applicatifs 2018. Edition des actes des JFLA 2018. Jan 2018, Banyuls-sur-Mer, France.

[2015b] David Braun and Nicolas Magaud. Des preuves formelles en Coq du théorème de Thalès pour les cercles. In D. Baelde and J. Alglave, editors, *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA2015)*, Jan. 2015.

[2014] Eric Andrès, Marie-Andrée Da Col, Laurent Fuchs, Gaëlle Largeteau-Skapin, Nicolas Magaud, Loïc Mazo, and Rita Zrour. Les Ω-AQA : Représentation discrète des applications affines. In N. Passat, editor, *Neuvièmes journées du Groupe de Travail de Géométrie Discrète (GéoDis 2014)*, Nov. 2014.

[2001] Nicolas Magaud and Yves Bertot. Changements de Représentation des Structures de Données en Coq : le cas des Entiers Naturels. In *Actes des Journées Francophones des Langages Applications JFLA'2001*, 2001.