CALCUL FORMEL ET PREUVES FORMELLES

ASSIA MAHBOUBI

1. Introduction

Calcul formel et preuve formelle sont des disciplines scientifiques à la croisée des mathématiques et de l'informatique. Elles explorent toutes deux « dans quelle mesure un ordinateur peut « faire des mathématiques » [5], en exploitant une représentation symbolique des objets mathématiques et des algorithmes efficaces pour les manipuler. Dans ces deux domaines, les avancées théoriques vont de pair avec un travail d'implantation sur machine, souvent dans des logiciels spécialisés – systèmes de calcul formel dans un cas, assistants à la preuve dans l'autre – dont la conception et la mise en oeuvre sont eux-mêmes des sujets de recherche.

Chacune de ces disciplines concentre néanmoins ses efforts sur une classe différente d'objets mathématiques. Le calcul formel étudie les objets mathématiques représentables par des expressions finies et exactes : polynômes, entiers, etc. Il s'agit d'expliciter des algorithmes efficaces, si possible optimaux, pour manipuler ces objets, et répondre ainsi à des questions mathématiques possiblement sophistiquées. La preuve formelle étudie les représentations symboliques des énoncés mathématiques et de leur preuves. Le choix de la logique utilisée comme syntaxe pour ces représentations fait partie des sujets d'étude, et de recherche. Cette formalisation permet en particulier de réduire la vérification des preuves à un procédé mécanique, mais aussi d'analyser le discours mathématique et de faire des expérience sur le corpus de théories formalisé.

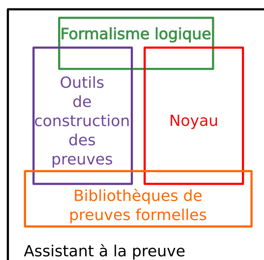
Ce document propose un aperçu illustré du domaine de la preuve formelle, sous l'angle de ses relations avec le calcul formel. Il se veut aussi peu technique que possible, et n'a aucune prétention à l'exhaustivité. La Section 2 décrit brièvement l'anatomie d'un assistant de preuve, et donne quelques exemples de systèmes. La Section 3 propose un choix subjectif de quelques réalisations, récentes ou moins récentes, en preuve formelle de mathématiques calculatoires. La Section 4 décrit l'assistant de preuve de prédilection de l'auteur de ce cours, le logiciel Coq [2] et la Section 4 conclut l'exposition.

2. Qu'est-ce qu'un assistant de preuve ?

Cette section est en partie adaptée d'un article paru sur le site Image des Mathématiques [31].

Le domaine de recherche de la preuve formelle rassemble tous les ingrédients qui visent à permettre la formalisation des mathématiques *en pratique*. Les mathématiques sont comprises ici dans leur acception la plus étendue, et on s'intéressera aussi bien aux théorèmes qui décrivent les propriétés d'un compilateur, qu'aux propriétés des fonctions holomorphes.

2.1. **Principe généraux.** Pour illustrer la palette de ces ingrédients, la figure 2.1 donne le schéma grossier du squelette d'un assistant de preuve :



On peut voir un assistant de preuve comme une machine à vérifier les démonstrations mathématiques. Il est important de noter ici que c'est bien la vérification des preuves qui est automatique, et non leur découverte : l'assistant de preuve attend de l'utilisateur qu'il fournisse une description non seulement de l'énoncé à valider mais aussi de la démonstration candidate — c'est ce qu'on appelle formaliser une démonstration. Tout comme les machines de calcul mécaniques ont précédé les ordinateurs, des vérificateurs de preuves mécaniques existaient bien avant l' ère de l'informatique. Le beau Piano Logique [29], conçu par William Stanley Jevons à la fin du XIXème siècle, est un exemple de telle machine, que l'on peut admirer au musée de l'histoire des sciences d'Oxford.

Pour construire un assistant de preuve, il faut d'abord se donner un langage formel, qui est un formalisme logique. Ce composant codifie l'écriture des énoncés mathématiques et des démonstrations. Il définit en particulier les règles de bonne formation de ces démonstrations, qui gouvernent le procédé mécanique de leur vérification. Cette tâche peut par suite être confiée à un programme unique, appelé le noyau. Celui-ci est vraiment la clef de voûte de l'assistant de preuve : ce sont ces mêmes quelques lignes de programme qui vont valider toutes les démonstrations soumises par l'utilisateur. Ainsi, si l'on est convaincu que ce programme est un vérificateur de preuves fiable, on pourra faire confiance à toutes les démonstrations qu'il valide. Autrement dit, la vérification d'une preuve mathématique donnée est ramenée à la confiance en la correction d'un programme unique qui sait les vérifier toutes. Plus ce vérificateur est simple, plus il est facile de faire confiance au programme qui l'implante. Mais plus il est sophistiqué, moins les preuves que l'utilisateur doit composer seront verbeuses.

Maintenant que l'on a fixé un langage formel pour écrire les énoncés et leurs preuves, il s'agit de décrire concrètement les théories qui nous intéressent dans ce langage et d'utiliser le noyau pour en vérifier les démonstrations. C'est ici que s'exerce la créativité de celui qui écrit les bibliothèques formelles, qui doit trouver la définition adéquate des objets mathématiques à un niveau de détail souvent absent de la littérature. À ce stade on se heurte également au fossé qui sépare le langage avec lequel les mathématiciens communiquent de façon intelligible et la verbosité exigée par une telle vérification exhaustive. En effet l'intelligibilité du discours des mathématiciens requiert un appareil d'abstraction, conventions et notations sans lequel le propos est enseveli sous les détails. Ce contenu implicite doit néanmoins être restauré pour que le vérificateur de preuve, que l'on veut simple, impitoyable, et donc stupide, puisse valider la preuve. Transcrire complètement dans le langage verbeux de la logique des mathématiques contemporaines semblait un impossible grand écart aux mathématiciens du collectif Nicolas Bourbaki, qui repoussent à plusieurs reprises dans leurs écrits la faisabilité, sinon l'intérêt même d'une telle entreprise [6]. Mais il permis de penser que le développement postérieur des preuves formelles *assistées par ordinateur* apporte un éclairage différent à ce jugement.

De fait, pour que l'activité de formalisation reste possible malgré ce grand écart, un assistant de preuve fournit à son utilisateur une panoplie d'outils qui permettent de minimiser l'information que l'on doit fournir pour décrire sa preuve candidate. Ces outils jouent par exemple le rôle de l'entraînement souvent quasi-inconscient du lecteur cultivé d'un texte mathématique, lorsqu'il reconstruit par lui-même ce qui n'est pas écrit. Nul besoin d'avoir une confiance absolue dans les outils qui permettent cette reconstruction : tous les coups sont permis puisque, in fine, chaque étape de la preuve est impitoyablement vérifiée par le noyau. Il est donc possible ici d'utiliser toutes les heuristiques, optimisations, automatisations que l'on souhaite pour faciliter la tâche de l'utilisateur. Dans cette partie de l'assistant de preuve, on trouve de fait des procédures de preuve automatique, qui dans ce contexte ont pour but non pas de se substituer à l'inventivité du mathématicien, mais au contraire, pour le décharger de ce qui lui serait trop pédestre.

Enfin, il ne s'agit pas de réinventer la roue chaque fois qu'une nouvelle démonstration vient à être vérifiée : il est primordial que l'utilisateur puisse s'appuyer sur des bibliothèques de mathématiques déjà formalisées et validées, pour construire de nouveaux développements. En fait, ces bibliothèques sont la véritable finalité de tout ce qui précède. Au delà de la vérification formelle d'un corpus de lemmes et de théorèmes, ces bibliothèques offrent un vaste terrain d'expérimentation avec les théories mathématiques.

2.2. **Logique(s).** La grande diversité des assistants de preuve tient en partie au choix que chacun d'entre eux fait pour la logique qui le sous-tend. Il y a de multiples choix possibles pour ces fondements, qui influent de façon significative l'activité de formalisation. La première qualité que l'on demande

à un tel formalisme logique est sa cohérence : il est indispensable de s'assurer que le cadre de départ dans lequel on va travailler est non contradictoire pour avoir confiance dans les preuves que l'on va formaliser !

En fait, cette preuve de non contradiction ne peut pas être exprimée dans le formalisme lui-même si celui-ci est suffisamment expressif pour nos besoins : c'est l'essence du second théorème de Gödel [17]. Mais on peut tout à fait faire une preuve de non contradiction, en la formalisant dans une logique légèrement plus puissante. Une autre possibilité consiste à traduire le formalisme qui nous intéresse dans un autre, considéré comme mieux compris, comme la théorie des ensembles de Zermelo-Fraenkel. Idéalement cette étude de la cohérence doit être elle-même vérifiée par l'assistant de preuve lui-même. C'est chose faite pour la plupart des assistants de preuve. Il reste ensuite à se convaincre que le vérificateur de preuves que l'on programme pour ce formalisme logique n'a pas de bugs. Le plus souvent, la taille de ce vérificateur est suffisamment petite pour qu'une inspection manuelle soit réaliste. Mais il est possible d'aller plus loin et d'utiliser toute la panoplie des méthodes formelles qui garantissent non seulement la preuve mathématique que les spécifications du programmes sont respectées, mais aussi que son exécution est fiable, en tenant compte d'aspect de plus bas niveau comme la gestion de la mémoire, de la compilation, etc.

À l'autre extrémité de la chaîne, le choix du formalisme sous-jacent aura des conséquences sur la façon dont on exprimera les définitions des objets mathématiques et leur propriétés. Par exemple, dans les formules mathématiques, on utilise traditionnellement des notations concises appelées quantificateurs pour indiquer la portée de la propriété que l'on énonce. Mais pour préciser le formalisme logique avec lequel on veut travailler, il est nécessaire d'expliquer dans la syntaxe formelle sur quelle nature d'objets on aura le droit de faire porter nos quantifications : certains formalismes interdisent ainsi de faire porter les quantificateurs sur les fonctions. Par exemple dans ce cas, pour exprimer que tout polynôme non nul à coefficients dans un certain corps K a au moins une racine, on écrira la chose suivante :

$$\forall p_0, \dots, p_{n+1} \in K, p_{n+1} \neq 0 \Rightarrow \exists x \in K, p_{n+1}x^{n+1} + \cdots + p_0 = 0$$

qui est une famille de formules paramétrée par un entier n. En effet on ne dispose pas dans cette syntaxe, dite du *premier ordre*, de l'expressivité suffisante pour donner un sens aux points de suspension, c'est à dire pour faire varier arbitrairement, mais dans une seule et même phrase, le nombre de quantificateurs $\forall$ qui permettent de parler d'un polynôme arbitraire. Par contre, si on autorise la quantification à porter sur des fonctions, la même notion s'écrira plutôt :

$$\forall P \in K[X], \quad \deg(P) > 0 \Rightarrow \exists x \in K, P(x) = 0$$

qui est une unique formule. On parle alors de logique d'*ordre supérieur*.

Les différentes natures de quantifications ne sont pas les seuls paramètres dans le choix du formalisme, loin s'en faut, et dans toute sa généralité cette

question d'expressivité a une importance toute particulière. Très grossièrement, plus un formalisme logique est rudimentaire, plus ses propriétés (méta)mathématiques comme sa cohérence, son expressivité, etc. sont faciles à étudier. Il est aussi souvent plus facile de concevoir des heuristiques efficaces pour la recherche automatique de preuve : un exemple extrême est celui de la logique propositionnelle, celle des tables de vérité des connecteurs booléens, à l'expressivité très limitée mais qui bénéficier d'heuristiques et de solveurs extrêmement puissants, appelés solveurs SAT.

Mais si une large part des mathématiques est en théorie exprimable dans une logique très élémentaire, se contraindre en pratique à une logique minimale n'est pas forcément pertinent dans notre contexte. L'une des difficultés principales de l'exercice de formalisation est la (re)définition formelle de tous les objets mathématiques qui composent la théorie étudiée. En particulier, il faut se convaincre que l'objet ou l'énoncé avec lequel on travaille dans la preuve formelle est le reflet fidèle de celui qui est décrit dans la littérature de référence. Cette vérification, qui celle-là ne peut être qu'humaine, peut être rendue beaucoup plus difficile lorsqu'un langage trop pauvre obscurcit le propos en imposant trop de codages et d'annotations.

Ainsi, le choix d'un bon formalisme pour fonder un assistant de preuve est essentiellement guidé par deux critères, parfois en tension : d'une part l'aisance avec laquelle on pourra concevoir un vérificateur de confiance pour ce formalisme, et des outils d'automatisation, et d'autre part le confort avec lequel on pourra y exprimer les mathématiques qui nous intéressent. Il n'y a pas de « meilleure logique possible », mais des compromis fondés sur l'usage auquel ses concepteurs destinent l'assistant de preuve a priori, qui peut aller de la vérification des propriétés de circuits logiques à la formalisation de pans d'algèbre abstraite, d'où la variété des outils disponibles.

2.3. **Quelques exemples d'assistants de preuve.** Il ne s'agit pas ici de donner un panorama exhaustif des assistants de preuve. On se limite à la présentation de quelques grandes familles de systèmes, dont relèvent les systèmes mentionnés dans la section 3. Pour un zoo plus fourni, on pourra par exemple se reporter au fascicule [44], qui présente la même preuve mathématique de l'irrationalité de $\sqrt{2}$, traitée par 17 systèmes différents. Édité il y a une décennie, il est bien sûr un peu incomplet ou périmé par endroits, mais il illustre la variété des approches possibles dans la conception d'un assistant de preuve.

2.3.1. *Automath.* Le langage AUTOMATH [36] ne sera pas mentionné dans la section 3, car il n'est plus utilisé aujourd'hui. Développé à la fin des années 60 par le mathématicien Nicolaas Govert de Bruijn, c'est en fait le premier assistant de preuve moderne. Sa conception repose sur la combinaison de plusieurs idées fondamentales qui en font l'ancêtre d'une large famille des assistants de preuve modernes, dont l'assistant de preuve Coq qui fait l'objet de la Section 4. Par un tour de force héroïque, de Bruijn et ses collaborateurs réussissent à formaliser complètement dans ce système un traité classique

d'analyse réelle [43]. Néanmoins ce système, probablement trop novateur, aura peu d'utilisateurs – mais beaucoup de descendants.

2.3.2. *Mizar.* Le Projet Mizar [40] est presque aussi ancien : initié par le mathématicien polonais Andrzej Trybulec au début des années 70, il est toujours utilisé aujourd'hui. Contrairement à la plupart des autres systèmes en usage aujourd'hui, ce langage est construit sur une variation de la théorie des ensembles de Zermelo Fraenkel, appelée théorie des ensembles de Tarski-Grothendieck, et la logique est du premier ordre. La lisibilité des documents de preuve est un souci majeur des concepteurs du système, et un journal en ligne, http ://mizar.org/fm/, décrit le corpus de théories disponible dans la bibliothèque du système. C'est probablement encore à ce jour le système qui possède la plus vaste collection de théories mathématiques formalisées. Par contre, la notion de calcul, et encore moins de programme, n'a pas de place particulière dans ce formalisme, ni dans l'assistant de preuve. Il est par suite difficile de représenter et d'exécuter des calculs non-triviaux en Mizar, et les mécanismes d'automatisation disponibles sont limités.

2.3.3. *Les prouveurs Boyer-Moore.* De l'autre côté du rideau de fer, la famille de systèmes dits « Boyer-Moore » [7], d'après les chercheurs en informatique étasuniens Robert S. Boyer et J Strother Moore, trouve également son origine au début des années 70, avec le système Nqthm [8]. Ils sont basés sur une logique du premier ordre, et de puissantes heuristiques de recherche de preuve automatique. L'interaction avec le système consiste à énoncer des successions de conjectures, assez élémentaires pour pouvoir être prouvées automatiquement, et qui une fois validées viennent grossir la base de connaissance utilisée par ces procédures automatiques. Si Nqthm a été utilisé pour vérifier des résultats d'algèbre ou de logique, sa version moderne, le système ACL2 [30] est surtout utilisé pour des applications industrielles, comme la conception de microprocesseurs et la vérification de circuits [9]. En particulier, il a été utilisé pour vérifier l'algorithme de division flottante du processeur AMD5K86, au lendemain de la découverte de l'illustre bug du processeur Pentium IV. Programmé en Common Lisp, il permet tout à la fois de prouver des théorèmes sur des fonctions d'un sous-ensemble (d'une variante) de Common Lisp, et d'exécuter efficacement ces fonctions.

2.3.4. *Robin Milner et LCF.* Une grande partie des assistants de preuve modernes sont, d'une façon ou d'une autre, des héritiers des travaux du britannique Robin Milner (prix Turing 1991). Celui-ci conçoit, toujours à l'aube des années 70, l'assistant de preuve LCF (pour Logic of Computable Functions) [35]. En fait, il conçoit simultanément LCF et le langage dans lequel celui-ci est programmé, appelé ML (pour Meta-Language). Ce dernier est l'ancêtre de plusieurs langages de programmation fonctionnels comme OCaml ou Standard ML. L'architecture de l'assistant de preuve LCF repose sur la définition d'un type de données abstrait du langage ML, qui représente les théorèmes de la logique. Celui-ci ne peut être habité que par des

programmes qui combinent les quelques constructions disponibles à cet effet, et qui représentent les connecteurs et les axiomes de la logique. Un langage d'interaction avec l'assistant de preuve permet une construction graduelle des démonstrations, en effectuant des pas de déduction élémentaires, ou de plus grand pas, pourvu qu'ils restent bien typés. La succession de ces commandes est donc le compte-rendu du chemin de preuve suivi. Cette architecture permet de concevoir des vérificateurs de preuves facile à vérifier et LCF donne son nom à la famille de systèmes qui l'utilisent.

2.3.5. *La famille HOL.* Certains descendants directs de LCF, comme Isabelle/HOL [38] ou encore HOL-Light [22], font partie des assistants de preuve les plus utilisés aujourd'hui. Ils sont bâtis sur le principe LCF mentionné plus haut, et ils proposent à leurs utilisateurs une logique classique d'ordre supérieure (HOL est l'acronyme de Higher-Order Logic), dans un formalisme introduit par Alonzo Church [11].

Le système HOL-Light tire son nom de sa réimplantation légère de son ancêtre HOL, et en particulier du soin apporté à la concision de son noyau logique. Ce système a été utilisé pour vérifier des algorithmes efficaces en virgule flottante [23], en particulier dans l'entreprise Intel qui employait son concepteur. Il possède également une bibliothèque d'analyse réelle et complexe très étoffée, probablement la plus complète à ce jour indépendemment de l'assistant de preuve considéré. Il est le système principalement utilisé dans la preuve formelle de la conjecture de Kepler (cf. section 3.1).

Le système Isabelle/HOL est lui aussi un successeur de HOL. Entre autres qualités, il propose à ses utilisateurs un langage de commandes avancé pour la description des preuves, qui améliore significativement leur lisibilité, ainsi que des outils de déduction automatique puissants, basés sur une combinaison de techniques de preuve automatique et d'apprentissage [4]. En Isabelle/HOL, il est possible d'écrire des programmes « dans la logique », c'est à dire dans le langage qui sert à écrire les énoncés, puis de traduire ces programmes dans le langage de programmation sous-jacent (un dialecte ML). La description du programme dans le langage de la logique permet de prouver des propriétés sur ce programme. La version traduite du programme, appelé « code extrait », permet de les exécuter de façon raisonnablement efficace. Ce code extrait est considéré comme très fiable, et il est donc possible d'exploiter le résultat de ces calculs dans les preuves : par exemple pour prouver que $2 + 3 = 5$, si l'addition est représenté par un programme, on pourrait extraire ce programme ainsi que les représentation des entiers 2 et 5, on calcule à l'extérieur de la logique et on retraduit le résultat, le nombre 5, dans la logique. Pour cette nature d'identités, ce mécanisme a bien sûr peu d'intérêt, et le calcul pourrait être (et est en pratique) effectué par réécriture, sans sortir de la logique. Mais il est crucial dans la vérification de preuves plus massivement calculatoires, qui justifie d'augmenter ainsi la base de code de confiance en ajoutant le code qui implante l'extraction à celui du noyau.

2.3.6. *Théorie des types dépendants.* Nous concluons ce rapide tour d'horizon avec une famille d'assistants de preuve qui proposent un langage logique encore plus expressif que ceux de la famille HOL. Cette famille comprend les assistants de preuve Coq [2], Agda [10], ou encore le plus récent Lean [15]. Leur formalisme logique sous-jacent trouve ses origines dans les travaux du mathématicien et philosophe suédois Per Martin Löf, qui propose dans les années 80 un formalisme appelé théorie des types intuitionniste [34], comme fondements constructifs des mathématiques. La thèse du mathématicien français Thierry Coquand [14] introduit quant à lui le Calcul des Constructions, descendant de la théorie des types de Martin Löf, mais aussi d'AUTOMATH, comme fondement d'un nouvel assistant de preuve, Coq. Celui-ci permet des quantifications très expressives : par exemple, on pourra formaliser l'énoncé « Tout corps algébriquement clos a la propriété d'élimination des quantificateurs » sans recourir à un schéma d'axiome. Comme dans le cas de la famille HOL, le formalisme est construit à partir du concept de λ-terme (c'est à dire de fonction, ou plus précisément de calcul) plutôt que d'ensemble. Ainsi, le langage logique inclut un langage de programmation typé, qui peut être exécuté dans la logique, ou bien extrait. Par contre, on ne peut pas ré-insérer dans un énoncé le résultat des calculs obtenus par le code extrait sans le justifier. Le parti pris des assistants de preuve Coq et Lean est en fait de doter le noyau du système de mécanismes efficaces d'évaluation des programmes écrits dans la logique [18, 3]. Le système de types de ce formalisme est très expressif : on peut y définir des structures de données pour un type des matrices de taille (n, m), des corps de caractéristique p, etc. pour n, m, p des paramètres symboliques entiers. Il est en fait à ce point expressif que c'est lui qui fournit le langage dans lequel on écrit les énoncés. Au cours de la dernière décennie, des connexions inattendues ont été mises en évidence entre la théorie des types de Martin Löf et la théorie abstraite, synthétique, de l'homotopie. Ces correspondances ont motivé une nouvelle direction de recherche en mathématique appelée théorie des types homotopique [42]. En particulier, elles permettent de « lire » certains résultats classique de théorie de l'homotopie comme des résultats formels en théorie des types. Par suite, les assistants de preuves sont utilisés de façon privilégiée pour formaliser, voire découvrir des résultats de théorie synthétique de l'homotopie.

3. Quelques succès récents

Cette section propose un échantillon, forcément restreint, du traitement dans les assistants de preuve de sujets liés au calcul formel. En fait, il s'agit plutôt d'une petite vitrine de preuves mathématiques par calcul symbolique certifié. Ainsi, on a écarté les formalisations de théorèmes liés au calcul formel, mais qui reposent sur des techniques moins spécifiques, comme par exemple les résultats de complexité de l'algorithmique du "diviser pour régner" [16], la décidabilité des corps réels clos [13] ou la théorie élémentaire

des courbes elliptiques [1]. Enfin, ce choix privilégie des résultats récents, publié au cours des cinq dernières années, et écarte ainsi des sujets pourtant pertinents comme les tests de primalité [19]. Notons que les calculs vérifiés de bases de Gröbner et leur applications en preuve formelle ont fait l'objet du cours de Loïc Pottier aux JNCF 2011.

3.1. **Conjecture de Kepler.** En 2005, un article [21] de Thomas Hales paraît dans le journal Annals of Mathematics, qui prouve le théorème suivant :

Théorème 1. *La densité d'un empilement de sphères congruentes dans* $\mathbb{R}^3$ *est au plus celle de l'empilement en configuration cubique à face centrée, soit* $\frac{\pi}{\sqrt{18}}$.

L'auteur avait annoncé en 1998 la conclusion de cette preuve monumentale, qui conclue elle-même une épopée de plusieurs siècles, ponctuée d'annonces de preuves qui se révéleront erronées, ou incomplètes. Mais cet article a lui même posé un défi aux méthodes usuelles de sélection des publications : il comprend de substantielles contributions mais repose de façon cruciale sur des calculs intensifs, réalisés par des programmes en partie écrits par l'auteur lui-même. Après plusieurs années de délibérations, de réorganisation, et de controverses, celui-ci est donc finalement publié. En 2014, Thomas Hales annonce l'achèvement d'un autre monument : la vérification formelle de la preuve complète, calculs inclus, en utilisant une combinaison de systèmes de preuve formelle (HOL-Light, Isabelle/HOL) [20].

La partie calculatoire de la preuve a trois volets. D'abord, il s'agit de construire une archive, finie mais volumineuse, de contre-exemples potentiels, qui sont en fait des graphes planaires, chacun des graphes étant associé à une configuration possible pour un amas de sphères proches dans l'empilement. Il s'agit de vérifier que l'archive est exhaustive [37], une vérification conduite avec l'assistant de preuve Isabelle/HOL. Puis, pour chacune de ces configurations, il s'agit de vérifier que la densité est en fait bornée comme attendue. Le deuxième volet calculatoire consiste en un problème de programmation linéaire, de grande taille (environ 10^5 problèmes, d'environ 200 variables et 2000 contraintes) même s'il reste bien en deçà des standards des applications industrielles. Enfin, le dernier volet consiste à borner des fonctions mathématiques non-linéaires, voire transcendantes en un petit nombre de variables, sur des pavés. C'est la vérification rigoureuse de cette dernière partie qui utilise des techniques symboliques-numériques, d'arithmétique d'intervalles, et en particulier des modèles de Taylor, en HOL-Light [39]. La vérification complète de cette preuve est un tour de force : la parallélisation du traitement des sous-problèmes, et la combinaison finale des sous-preuves, parfois écrites dans des langages différents, a posé des défis considérables d'ingénierie de preuve formelle.

3.2. **Un solveur d'ODE rigoureux.** En 1999, Warwick Tucker annonce que « L'attracteur de Lorentz existe » [41], apportant ainsi une solution

au 14ème problème de Smale. Cette solution repose en partie sur l'implantation d'un solveur d'équations différentielles ordinaires rigoureux, qui implante dans le langage C++ un algorithme dû à l'auteur et basé sur une arithmétique d'intervalles.

Fabian Immler a implanté dans l'assistant de preuve Isabelle/HOL un solveur d'équations différentielles ordinaires rigoureux *et* certifié, basé sur le travail de Warwick Tucker [28]. Au moyen de ce solveur, il a calculé une approximation de l'attracteur de Lorentz similaire à celle de la preuve originale, qui permet ainsi une analyse de ces calculs, à l'intérieur d'un assistant de preuve. Le calcul est réalisé par un programme extrait en SML et compilé avec MLTon. Les calculs arithmétiques sont exacts, et utilisent la bibliothèque GMP. En 2015, l'auteur mentionne que les calculs, distribué sur les 1024 cœurs d'un cluster avec une limite de temps de 7 heures prennent 7000 heures. Warwick Tucker, quinze ans auparavant, annonce 100 heures de calculs, distribués sur 20 processeurs.

3.3. **Wilf-Zeilberger.** John Harrison a étudié la formalisation dans le système HOL-Light, dont il est l'auteur, de l'algorithme de Wilf-Zeilbeger (WZ) pour les sommes de suites hypergéométriques [25]. Cette méthode présente un intérêt spécial dans le contexte d'une vérification formelle, car elle permet de prouver une identité grâce à un certificat, ici une fraction rationnelle. Peu importe comment ce certificat est calculé, et la difficulté de ce calcul, une fois connu, il réduit la preuve de l'identité initiale à des manipulations algébriques triviales. L'intérêt de ce travail est en particulier de mettre en valeur le statut pernicieux des dénominateurs des fractions rationnelles en jeu. On ne peut cacher à l'assistant de preuve la distinction entre fraction rationnelle (de l'algèbre) et fonction rationnelle (possiblement non définie en certains points) et les présentations trop rapides de l'algorithme WZ ne justifient pas complètement les calculs. John Harrison propose une preuve de correction, basée sur des arguments d'analyse complexe, à notre connaissance nouvelle, qu'il a vérifié formellement.

3.4. **Discussions avec des oracles.** L'exemple de l'algorithme WZ mentionné en Section 3.3 est emblématique d'une classe de problèmes particulièrement avantageux du point de vue de la vérification formelle. Comme les capacités de calcul des assistants de preuve sont limitées par leur rôle de vérificateur, il est préférable de déléguer des phases d'exploration à un programme externe aussi souvent que possible. Dès les années 90, John Harrison et Laurent Théry soulignent les bénéfices de ce mode d'interaction [26]. Depuis plusieurs travaux ont proposé des ponts entre l'un ou l'autre des systèmes de calcul formel, et l'un ou l'autre des assistants de preuve. Malheureusement, ces ponts sont par nature des programmes très fragiles, et la plupart des preuves formelles qui utilisent des oracles externes incluent leur propre canal de communication. Par exemple, la preuve formelle de l'irrationalité de $\zeta(3)$ par calcul formel obtenue par Frédéric Chyzak et ses

co-auteurs [12] repose sur un traducteur ad hoc des résultats produits par la bibliothèque Maple/Algolib comme des énoncés en Coq.

Bien sûr, les oracles intéressants ne sont pas seulement ceux fournis par des algorithmes symboliques. Par exemple, la distribution standard de HOL-Light, et celle de Coq, incluent des procédures de décision pour l'arithmétique non-linéaire reposant sur des décompositions en sommes de carrés calculées par programmation semi-définie [24]. Le récent travail d'Erik Martin-Dorel et Pierre Roux améliore significativement les performances de ces heuristiques de preuve formelle [33].

3.5. Nombre de Schur.

Le dernier exemple de cette liste ne relève ni tout à fait du calcul formel, ni tout à fait de la preuve formelle au sens où nous l'avons entendue jusqu'ici. Il s'agit de la toute récente preuve informatique d'un problème posé il y a un siècle :

Théorème 2. *Le plus grand entier n pour lequel il existe un cinq-coloriage des entiers positifs inférieurs ou égaux à n sans solution monochromatique de l'équation $a + b = c$ vaut* 160*.*

Marijn Heule a obtenu cette valeur [27] en encodant ce problème dans la logique propositionnelle (celle des tables de vérité booléennes), et en utilisant des algorithmes de preuve automatiques de pointe pour cette classe de formules. Ces algorithmes son implantés dans des outils appelés solveurs SAT, et dans le cas présent les calculs sont massivement parallèles. Le solveur SAT produit une réponse, mais il peut aussi produire un certificat, que l'on peut vérifier avec un assistant de preuve. Dans le cas du problème ci-dessus, le certificat produit (un fichier d'environ deux petabytes) a été vérifié avec le système ACL2.

Le même auteur avait déjà montré la puissance et l'intérêt des techniques de résolution SAT sur d'autres problèmes de combinatoire dans la famille de la théorie de Ramsey. En 2016, il a résolu avec ses co-auteurs le problème booléen des triplets pythagoriciens, avec un certain écho médiatique autour de la supposée « plus grosse preuve mathématique du monde ». Une page consacrée à ce résultat en présente la description grand public, et recense les commentaires médiatiques ou scientifique ayant accueilli cette annonce.

4. Un petit tour guidé de Coq et de ses bibliothèques

Il s'agit dans la deuxième partie de ce cours de rendre plus concret l'activité de preuve formelle. On se propose d'illustrer le propos par une démonstration du système, et quelques exercices. Ces derniers sont des miniatures d'ingrédients d'un travail récent sur la vérification d'estimations d'intégrales définies [32]. Le lecteur intéressé se reportera donc à la page web [1] consacrée à cette introduction.

1. `http://specfun.inria.fr/amahboub/Jncf18/cours2.html`

5. Conclusion

Les deux grandes techniques de vérification de programmes qui convoquent des assistants de preuves sont :

— *l'extraction*, qui permet de générer des programmes dans un langage fonctionnel (souvent un dialecte ML comme OCaml, ou SML), en traduisant une représentation idéales du code sur laquelle on a prouvé des propriétés ;

— l'annotation de programmes écrits dans un langage généraliste, possiblement impératif, qui permet à des outils de vérification de programme, de générer des obligations de preuve qui garantissent que les propriétés annotant les entrées du programme suffisent à garantir les propriétés des sorties. La plupart des formules générées comme obligations seront prouvées par des outils automatiques, mais des cas plus rares pourront nécessiter une preuve interactive.

Il nous semble que les algorithmes du calcul symbolique, ou symbolique-numérique, méritent tout à la fois de bénéficier des progrès significatifs des techniques de vérification de programme, et d'une approche plus spécifique. D'une part, vérifier des algorithmes de calcul formel en utilisant un assistant de preuve peut permettre d'enrichir l'arsenal d'outils d'automatisation de ce dernier, ce qui motive dans certains cas leur implantation « dans la logique ». D'autre part, la plupart des systèmes de calcul formel entretiennent un flou assumé sur la sémantique des objets qu'ils manipulent, à la fois expressions purement symboliques et fonctions. Cette ambiguité est une source d'erreurs, par interprétation abusive plutôt que par une forme plus classique de bug. Ces glissements de sens justifient l'emploi d'un langage plus expressif pour spécifier les programmes, langage dans lequel on pourra représenter à la fois les instances des structures algébriques sur lequel porte le calcul formel, mais aussi les objets qu'elles idéalisent ainsi que leur conditions d'existence.

Références

[1] E.-I. Bartzia and P.-Y. Strub. A formal library for elliptic curves in the coq proof assistant. In G. Klein and R. Gamboa, editors, *Interactive Theorem Proving*, pages 77–92, Cham, 2014. Springer International Publishing.

[2] Y. Bertot and P. Castran. *Interactive Theorem Proving and Program Development : Coq'Art The Calculus of Inductive Constructions.* Springer Publishing Company, Incorporated, 1st edition, 2010.

[3] M. Boespflug, M. Dénès, and B. Grégoire. Full reduction at full throttle. In *First International Conference on Certified Programs and Proofs, Tawain, December 7-9*, Lecture Notes in Computer Science. Springer, 2011.

[4] S. Böhme and T. Nipkow. Sledgehammer : Judgement day. In *Proceedings of the 5th International Conference on Automated Reasoning*, IJCAR'10, pages 107–121, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] A. Bostan, F. Chyzak, M. Giusti, R. Lebreton, G. Lecerf, B. Salvy, and É. Schost. *Algorithmes Efficaces en Calcul Formel.* Frédéric Chyzak (auto-édit.), Palaiseau, Sept. 2017. 686 pages. Imprimé par CreateSpace. Aussi disponible en version électronique.

[6] N. Bourbaki. L'architecture des mathématiques. In F. L. Lionnais, editor, *Les grands courants de la pensée mathématique*. Cahiers du Sud, 1948.

[7] R. Boyer, M. Kaufmann, and J. Moore. The boyer-moore theorem prover and its interactive enhancement. *Computers & Mathematics with Applications*, 29(2) :27 – 62, 1995.

[8] R. S. Boyer and J. S. Moore. Proving theorems about lisp functions. *J. ACM*, 22(1) :129–144, Jan. 1975.

[9] B. Brock, M. Kaufmann, and J. S. Moore. Acl2 theorems about commercial microprocessors. In M. Srivas and A. Camilleri, editors, *Formal Methods in Computer-Aided Design*, pages 275–293, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[10] T. C. Catarina Coquand. Structured type theory. In *Workshop on Logical Frameworks and Metalanguages*, 1999.

[11] A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5(2) :56–68, 06 1940.

[12] F. Chyzak, A. Mahboubi, T. Sibut-Pinote, and E. Tassi. A computer-algebra-based formal proof of the irrationality of $\zeta(3)$. In R. G. Gerwin Klein, editor, *Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*. Springer, 2014.

[13] C. Cohen and A. Mahboubi. Formal proof in real algebraic geometry : from ordered fields to quantifier elimination. *Logical Methods in Computer Sciences*, 8(1 :2) :1–40, 2012.

[14] T. Coquand. *Une théorie des constructions*. PhD thesis, Paris 7, 1985. Thèse de doctorat dirigée par Huet, Gérard.

[15] L. M. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The lean theorem prover (system description). In A. P. Felty and A. Middeldorp, editors, *CADE*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[16] M. Eberl. Proving divide and conquer complexities in isabelle/hol. *J. Autom. Reason.*, 58(4) :483–508, Apr. 2017.

[17] K. Gödel. Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme, I. *Monatshefte für Math.u.Physik*, 38 :173–198, 1931.

[18] B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *International Conference on Functional Programming 2002*, pages 235–246. ACM Press, 2002.

[19] B. Grégoire, L. Théry, and B. Werner. A computational approach to pocklington certificates in type theory. In *Functional and Logic Programming, 8th International Symposium, FLOPS 2006, Fuji-Susono, Japan, April 24-26, 2006, Proceedings*, volume 3945 of *Lecture Notes in Computer Science*, pages 97–113. Springer, 2006.

[20] T. HALES, M. ADAMS, G. BAUER, T. D. DANG, J. HARRISON, L. T. HOANG, C. KALISZYK, V. MAGRON, S. MCLAUGHLIN, T. NGUYEN, and et al. A formal proof of the kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[21] T. C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162(3) :1065–1185, 2005.

[22] J. Harrison. HOL light : A tutorial introduction. In M. Srivas and A. Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.

[23] J. Harrison. Floating-point verification using theorem proving, 2006.

[24] J. Harrison. Verifying nonlinear real formulas via sums of squares. In K. Schneider and J. Brandt, editors, *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2007*, volume 4732 of *Lecture Notes in Computer Science*, pages 102–118, Kaiserslautern, Germany, 2007. Springer-Verlag.

[25] J. Harrison. Formal proofs of hypergeometric sums (dedicated to the memory of andrzej trybulec). *Journal of Automated Reasoning*, 55 :223–243, 2015.

[26] J. Harrison and L. Théry. A skeptic's approach to combining HOL and Maple. *J. Autom. Reason.*, 21(3) :279–294, Dec. 1998.

[27] M. J. Heule. Schur number five. `https://arxiv.org/abs/1711.08076`, 2018. Accepted for publication at AAAI 2018.

[28] F. Immler. A verified enclosure for the lorenz attractor (rough diamond). In C. Urban and X. Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, pages 221–226, 2015.

[29] W. S. Jevons. On the mechanical performance of logical inference. In *Philosophical Transactions of the Royal Society of London for the year 1870*, volume 160 II, pages 497–518. Taylor and Francis, 1870.

[30] M. Kaufmann and J. S. Moore. An industrial strength theorem prover for a logic based on common lisp. *IEEE Trans. Softw. Eng.*, 23(4) :203–213, Apr. 1997.

[31] A. Mahboubi. Un ordinateur pour vérifier des preuves mathématiques. Images des Mathématiques, en partenariat avec le séminaire Bourbaki, août 2014. `http://images.math.cnrs.fr/Un-ordinateur-pour-verifier-les.html#nb2`.

[32] A. Mahboubi, G. Melquiond, and T. Sibut-Pinote. Formally Verified Approximations of Definite Integrals. Under review, Feb. 2017.

[33] E. Martin-Dorel and P. Roux. A Reflexive Tactic for Polynomial Positivity using Numerical Solvers and Floating-Point Computations (regular paper). In *ACM SIGPLAN Conference on Certified Programs and Proofs (CPP), Paris, 16/01/2017-17/01/2017*, pages 90–99, http ://www.acm.org/, janvier 2017. ACM.

[34] P. Martin-Löf and G. Sambin. *Intuitionistic type theory*. Studies in proof theory. Bibliopolis, 1984.

[35] R. Milner. Lcf : A way of doing proofs with a machine. In J. Bečvář, editor, *Mathematical Foundations of Computer Science 1979*, pages 146–159, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

[36] R. Nederpelt, H. Geuvers, and R. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1994.

[37] T. Nipkow, G. Bauer, and P. Schultz. Flyspeck I : tame graphs. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 21–35, 2006.

[38] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL : A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[39] A. Solovyev. *Formal Computations and Methods*. PhD thesis, University of Pittsburgh, 2012.

[40] A. Trybulec. The MIZAR-QC/6000 logic information language. *Association for Literary and Linguistic Computing Bulletin*, 6 :136–140, 1978.

[41] W. Tucker. A rigorous ode solver and smale's 14th problem. *Foundations of Computational Mathematics*, 2(1) :53–117, Jan 2002.

[42] T. Univalent Foundations Program. *Homotopy Type Theory : Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[43] L. van Benthem Jutting. *Checking Landau's "Grundlagen" in the automath system*. Mathematical Centre tracts. Mathematisch Centrum, 1979.

[44] F. Wiedijk. *The Seventeen Provers of the World : Foreword by Dana S. Scott (Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.