Quel est l'intérêt des ordinateurs ? Une question pour les mathématiciens purs

Kevin Buzzard

Imperial College London
Département de mathématiques

Résumé : Nous discutons de l'idée que les ordinateurs pourraient bientôt aider les mathématiciens à prouver des théorèmes dans des domaines dans lesquels ils n'ont pas été utiles jusque-là. De plus, nous arguons que ces mêmes outils informatiques nous aideront également dans la communication et l'enseignement des mathématiques.

Introduction

En 2021 les ordinateurs sont phénoménaux. Ils peuvent faire des milliards de calculs en une seconde. Ils sont extrêmement doués pour obéir avec précision à des instructions spécifiques. Les mathématiques sont un jeu avec des règles précises. On peut donc se demander de quelle manière les ordinateurs peuvent être utilisés pour nous aider, nous mathématiciens[1], à faire notre travail.

Bien sûr, les ordinateurs ont été utilisés pour aider certains mathématiciens dans leur tâche depuis qu'ils existent. Birch et Swinnerton-Dyer ont utilisé un ancien ordinateur (qui avait la taille d'une grande pièce et qui avait 20 kilo-octets de mémoire) pour calculer de nombreux exemples de solutions d'équations cubiques à deux variables modulo des nombres premiers [BSD65]. La représentation graphique adaptée des données en sortie a conduit à de nouvelles perspectives dans la théorie des courbes elliptiques qui ont finalement amené à la conjecture de Birch et Swinnerton-Dyer, l'un des problèmes Clay du millénaire. Au moment de l'écriture de cet article, cette conjecture est encore ouverte, bien que des percées régulières (plus récemment dans la théorie non commutative d'Iwasawa) aient permis des progrès progressifs.

Cet article ne concerne pas l'utilisation des ordinateurs de cette façon. Cet article est une tentative d'expliquer à *tous* les chercheurs en mathématiques que, grâce aux percées de l'informatique, les ordinateurs peuvent désormais être utilisés pour nous aider non seulement dans les calculs, mais aussi dans le *raisonnement*. En d'autres termes, il s'agit de la possibilité que les ordinateurs nous aident bientôt à *prouver des théorèmes*, qu'ils concernent des objets "calculables" tels que des courbes elliptiques, ou bien des objets plus difficiles à traiter tels que des espaces de Banach, des schémas, des catégories abéliennes ou des espaces perfectoïdes, des choses qui ne peuvent pas être répertoriées ou classées ou en général stockées dans un logiciel d'algèbre informatique traditionnel de manière significative. En particulier, il s'agit de la possibilité que des assistants informatiques de preuve puissent aider le mathématicien qui jusqu'alors n'avait pas besoin de calcul dans ses recherches et pouvait donc en déduire à tort que les ordinateurs n'avaient rienà lui offrir du tout. Je dois également souligner que les personnes concernées ne se limitent pas aux personnes intéressées par des sujets fondamentaux tels que la théorie des ensembles ou la théorie des types ; je réfléchis à des applications en géométrie, topologie, combinatoire, théorie des nombres, algèbre, analyse, ...

Je termine cette introduction par un résumé de ce à quoi s'attendre et ce à quoi ne pas s'attendre de ce domaine à croissance rapide au cours de la prochaine décennie. La première chose à souligner est que les ordinateurs ne nous mettront pas au chômage. Les assistants de preuves peuvent maintenant comprendre l'*énoncé* de l'hypothèse de Riemann, mais je mangerai mon chapeau si un ordinateur, tout seul, apporte une *preuve* de l'hypothèse de Riemann (ou en effet, une preuve de n'importe quel problème intéressant pour la plupart des mathématiciens purs) dans les 10 prochaines années[2].

Voici ce qui va arriver, je pense, dans les 10 prochaines années : des outils seront créés qui *aideront* les mathématiciens à prouver des théorèmes. Des bases de données numérisées et consultables sémantiquement

email : k.buzzard@imperial.ac.uk

Référence : https://arxiv.org/abs/2203.16344.
Traduction : Denise Vella-Chemla, mai 2022.

[1]Tout au long de cet article, par "nous", je fais référence à la communauté de personnes qui, comme moi, s'identifient à des mathématiciens purs.

[2]Les conjectures qui s'étendent au-delà d'une période de 10 ans sont, je pense, très imprudentes ; comme les mathématiques, parfois l'informatique évolue très vite.

sur les mathématiques apparaissent. Les ordinateurs vont commencer à faire des chasses aux diagrammes pour nous, remplir les preuves des lemmes, indiquer des contre-exemples à nos idées et suggérer des résultats qui pourraient nous être utiles. La technologie pour fabriquer de tels outils arrive déjà ; elle est viable. De plus, les bases de données d'énoncés de théorèmes et de preuves qui apparaissent n'auront pas seulement des applications dans la recherche ; nous pourrons les utiliser pour enseigner et pour communiquer les mathématiques de nouvelles façons. Les étudiants de premier cycle pourront obtenir des commentaires instantanés sur leur travail. Les doctorants pourront rechercher des théorèmes et des contre-exemples dans des bases de données. Les chercheurs pourront rédiger des documents de nouvelle génération sans erreur où les détails peuvent être pliés et dépliés par l'utilisateur. Patrick Massot a écrit un article profond [Masb] expliquant ces idées et d'autres plus en détail. Les ordinateurs seront capables de comprendre *votre domaine* des mathématiques et même de le suivre au fur et à mesure de son développement. Mais il y a un hic. Qui va créer la base de données des résultats importants en théorie non commutative d'Iwasawa, ou quel que soit le domaine qui vous intéresse, qui alimentera ces outils ? Ce ne seront pas les informaticiens, car la plupart d'entre eux ne connaissent rien à la théorie non commutative d'Iwasawa. *Ça doit être nous.*

Si vous souhaitez voir des progrès dans la preuve automatique de théorèmes dans votre propre domaine des mathématiques, je vous *exhorte* à prendre du temps pour travailler sur des tutoriels et pour apprendre l'un de ces langages d'assistant de preuve informatique. Ce n'est pas difficile à faire - j'enseigne un cours qui a du succès aux étudiants de dernière année en mathématiques où nous apprenons à faire des mathématiques de premier cycle (topologie, analyse, théorie des groupes, etc.) en utilisant le démonstrateur de théorème Lean[3]. S'engager dans des mathématiques plus complexes n'est pas du tout difficile *une fois qu'on connaît le langage* [pca]. Si vous voulez apprendre le langage Lean, un bon point de départ est le site Web de la communauté des démonstrateurs Lean [pca]. Coq et Isabelle/HOL sont deux autres démonstrateurs de théorèmes bien établis avec de grandes bibliothèques mathématiques, et il y en a plein d'autres. Si vous pouvez arriver au point où vous êtes capable d'expliquer les *instructions* de vos propres théorèmes à un assistant de preuve informatique, alors ces déclarations peuvent être ajoutées aux bases de données, et en plus vous avez appris une nouvelle compétence. Si toutefois vous pouvez arriver au point où vous pouvez expliquer les *preuves*, alors les gens de l'IA seront extrêmement intéressés, tout comme les gens qui construisent d'énormes bibliothèques mathématiques formalisées qui représentent le Bourbaki du 21ème siècle. De plus, vous vous amuserez : la formalisation des preuves, ce sont les mathématiques réinterprétées comme un intéressant jeu de réflexion sur ordinateur. Si vous n'avez pas le temps, trouvez un étudiant qui en a. Au lieu du traditionnel "faire un projet consistant à lire un article puis à écrire un article montrant que vous avez compris l'article", pourquoi ne pas obtenir des étudiants qu'ils écrivent un programme qui prouve qu'ils ont compris l'article ? Ils peuvent apprendre pour eux-mêmes le langage du démonstrateur, puis vous l'enseigner comme vous leur enseignez les mathématiques.

Les fichiers que les assistants de preuve informatique peuvent lire et écrire représentent une manière de numériser des idées mathématiques. Numériser quelque chose change *complètement* (en fait, cela augmente considérablement) les façons dont cette chose peut être utilisée. Considérons par exemple la numérisation de la musique, avec le CD et le fichier mp3. Cela a révolutionné la façon dont la musique est consommée et livrée. Ma collection de musique se compose de centaines de disques vinyles, de cassettes et de CD dans mon bureau et mon loft. La collection de mes enfants est dans le cloud, a une masse et un volume essentiellement nuls et est accessible partout. Non seulement cela, mais les plateformes musicales basées dans le cloud ont également fondamentalement changé la façon dont le musicien moderne communique avec ses fans, en contournant complètement le processus traditionnel. L'industrie de la musique a été complètement mise sens dessus-dessous par la digitalisation.

Les mathématiques ont été faites de la même manière "papier et crayon" pendant des millénaires, mais il existe maintenant une véritable opportunité de repenser et d'améliorer cette approche. Je n'ose pas rêver à ce que seront les conséquences ultimes de la numérisation des mathématiques, mais je crois fermement que cela rendra les mathématiques plus accessibles - et plus faciles à faire, à communiquer et plus ludiques. La balle est dans notre camp.

[3]Si vous avez installé Lean, alors vous pouvez suivre le cours vous-même ; les ressources sont ici [Buza].

Aperçu de l'article

Cet article décrit une "nouvelle" façon dont les ordinateurs peuvent être utilisés par les mathématiciens. En tant que mathématiciens, notre expérience typique avec les ordinateurs est que nous pouvons utiliser des langages de programmation traditionnels comme python ou des packages informatiques d'algèbre traditionnelle comme `sage` pour faire des choses comme calculer la somme des 100 premiers nombres premiers. Nous savons également que ces outils traditionnels, même s'ils peuvent calculer autant de nombres premiers que l'on veut (dans la limite du raisonnable), ne sont pas capables de *prouver* qu'il existe une infinité de nombres premiers ; l'infini est notre domaine, pas le domaine de l'ordinateur.

Cependant, ce n'est plus le cas. Les assistants informatiques de preuve sont des programmes qui connaissent les axiomes des mathématiques. Une conséquence de cela est qu'ils peuvent faire à la fois du calcul au sens traditionnel et du *raisonnement*. En pratique, cela signifie que l'on peut écrire du code informatique dans un assistant de preuve qui correspond à la preuve qu'il existe une infinité de nombres premiers lien, ou même à une preuve [DHL19] du résultat principal dans un article récent des Annales [EG17].

J'ai écrit "nouveau" entre guillemets ci-dessus car ce n'est pas du tout nouveau ; les informaticiens créent des outils comme celui-ci depuis des décennies. En effet, les premiers assistants informatiques de preuve sont apparus dans les années 1960. Cependant, plus récemment, trois choses se sont produites. Premièrement, la technologie a maintenant atteint le point où les résultats au niveau de la recherche dans tous les domaines traditionnels des mathématiques pures sont désormais simultanément accessibles à ces systèmes, du moins en théorie et, de plus en plus, en pratique. Deuxièmement, les systèmes sont beaucoup plus autonomes qu'ils ne l'étaient. Les stratégies sont des commandes qui peuvent être conçues par les utilisateurs et qui sont capables d'assembler des centaines, voire des milliers d'étapes axiomatiques fastidieuses, permettant aux mathématiciens de communiquer avec ces machines dans un langage de haut niveau, similaire à la manière dont elles communiquent entre elles. Enfin, et surtout, les mathématiciens au niveau de la recherche commencent enfin à s'impliquer ; nous voyons du matériel au niveau mastére et au-delà commencer à être formalisé, par des mathématiciens, dans de nombreux domaines des mathématiques maintenant. Ces développements signifient que l'enseignement d'un matériau à un niveau recherche à un ordinateur dans tous les domaines des mathématiques est en train de devenir une possibilité faisable - en effet, cela arrive en ce moment-même, et aucun signe ne montre que cela va s'arrêter.

Cet article est constitué de 4 sections, qui sont indépendantes les unes des autres, et peuvent être lues dans un ordre indifférent.

La première section est historique ; elle consiste en des descriptions des systèmes qui sont, ou ont été, utilisés pour formaliser les mathématiques, et en des discussions à propos des résultats qui ont été enseignés par les humains aux ordinateurs dans les 20 dernières années. Elle pointe également des avancées techniques historiques diverses.

La seconde section fournit un aperçu d'une des plus grandes bibliothèques monolithiques mathématiques existant, notamment la bibliothèque `mathlib` de Lean. Lean [dMKA+15] est un assistant de preuves libre et open source écrit initialement par Leonardo de Moura au département recherche de Microsoft. La bibliothèque de mathématiques `mathlib` [mathlibc20] de Lean est une bibliothèque libre et open source pour Lean, développée par une communauté d'utilisateurs à travers le monde, allant du niveau d'étudiants universitaires à celui de mathématiciens professionnels. `mathlib` est la bibliothèque qui a rendu possible quelques-uns des résultats les plus récents dans le domaine.

La troisième section consiste en une introduction à la théorie des types comme fondements des mathématiques ; elle explique comment les structures mathématiques, les théorèmes et les preuves peuvent être codés selon ces fondements. Notons que de nombreux systèmes de preuves par ordinateurs qui contiennent des mathématiques non-fondamentales (Lean, Coq, Isabelle/HOL) utilisent la théorie des types plutôt que la théorie des ensembles ; pourtant la théorie des types prouve les mêmes théorèmes que la théorie des ensembles. De plus, les mathématiciens qui peuvent prouver des théorèmes mais qui ne connaissent pas les axiomes de la théorie des ensembles ZFC peuvent joyeusement écrire du code dans un système de preuves basé sur la théorie des types,

code qui correspond à leurs théorèmes sans connaître les axiomes de la théorie des types non plus.

Finalement, une section spéculative décrit plus en détail des idées personnelles de l'auteur et d'autres idées à propos des sortes de choses pour lesquelles on peut utiliser un tel logiciel, et comment cela pourrait nous aider à faire notre travail.

Remerciements : Je remercie la communauté du démonstrateur Lean pour m'avoir accueilli, moi le mathématicien ayant peu d'expérience de la programmation, dans leur communauté en 2017, et également pour avoir lu et fourni de larges commentaires d'une version préliminaire de cet article. Patrick Massot en particulier m'a envoyé de nombreux commentaires utiles sur un premier brouillon. Je remercie Assia Mahboubi et Manuel Eberl pour m'avoir donné des conseils à propos du code en Coq et Isabelle/HOL de cet article, ainsi qu'à eux et à Jeremy Avigad pour leur aide à propos des commentaires historiques. Finalement, je voudrais remercier avec effusion Leonardo de Moura pour avoir écrit mon jeu d'ordinateur favori, et Mario Carneiro pour m'avoir appris à y jouer.

1. Une brève histoire des théorèmes vérifiés formellement

Dans cette section, je parlerai des succès précédents des assistants informatiques de preuves – des programmes informatiques qui vérifient des preuves humaines – en mathématiques. Il y a de nombreux autres projets que j'aurais pu mentionner, et je présente mes excuses à ceux qui ont entrepris des projets de formalisation mathématique majeurs et que je n'ai pas cités. Des exemples d'assistants informatiques de preuves dans lesquels une partie substantielle des mathématiques a été formalisée incluent Lean [dMKA+15], Coq [Tea21], Isabelle/HOL [NPW02], HOL Light [Har09b], Metamath [MW19] et Mizar [NK09].

Pour qu'un ordinateur vérifie formellement une preuve, il a besoin d'être capable de déduire un théorème des axiomes du système de fondements (typiquement, la théorie des ensembles ou la théorie des types) que l'assistant de preuves a été programmé à utiliser. J'utiliserai la discussion ci-dessous à propos des résultats historiques pour introduire quelques percées conceptuelles qui ont permis que la formalisation des mathématiques devienne faisable.

Cette section ne peut rendre justice à tout le travail qui a été réalisé dans le domaine ; je recommande vivement la lecture de l'article de Hales "Mathematics in the age of the Turing machine" [Hal14] pour davantage de support et d'exemples, même si beaucoup de choses se sont produites depuis cet article écrit en 2014.

1.1. Le 20ème siècle. Considérons le problème de démontrer à partir des premiers principes que si x et y sont des nombres réels, alors $(x+y)(x+2y)(x+3y) = x^3+6x^2y+11xy^2+6y^3$. Nous savons tous que les nombres réels forment un anneau commutatif, donc supposons ce fait. La question devient alors de savoir comment utiliser les axiomes d'un anneau commutatif pour prouver l'égalité que nous voulons. Combien de lignes une telle preuve à partir des premiers principes pourrait-elle contenir ? Sûrement pas trop ! On applique la distributivité quelques fois pour développer les crochets à gauche, et alors bien sûr, cela devient une façon de ranger et de rendre égaux des termes. En tant qu'humains, nous ne pensons pas trop au processus de rangement, pourtant si vous essayez de le prouver dans un assistant de preuves, alors vous découvrirez que c'est un vrai cauchemar combinatoire. Par exemple, il y a une étape dans la démonstration où l'on a besoin de démontrer quelque chose de la forme

$$((A+B)+(C+E))+((D+F)+(G+H)) = ((((((A+B)+C)+D)+E)+F)+G)+H$$

en utilisant seulement les lois de la commutativité et de l'associativité de l'addition. Les humains appliquent un *principe* pour justifier cette étape, non pas un axiome, et en effet, prouver une telle évidence en utilisant seulement les axiomes d'un anneau est de façon inattendue très délicat à réaliser. Il y a également le problème de transformer des choses comme $x((2y)x)$ en $(2(x^2))y$ et etc.

Les tout premiers assistants de preuves avaient une capacité très limitée à appliquer les principes, ce qui signifie que prouver des résultats tels que $(x+y)(x+2y)(x+3y) = x^3+6x^2y+11xy^2+6y^3$ devait être fait à la main, ce qui correspond à une preuve d'environ 30 lignes. Si une telle évidence cache 30 lignes de mathématiques axiomatisées, imaginez ce qui se cache derrière des énoncés de la forme "La fonction f est trivialement en $O(x^{-2})$ pour x grand" ? C'est une chose que d'écrire un assistant informatique de preuves –

c'en est une autre d'en écrire un qui réussit à faire les sortes de choses que nous humains faisons intuitivement. Pour cela et pour d'autres raisons, la plupart des résultats initiaux de formalisations du 20ème siècle étaient mathématiquement triviaux. En particulier, il y a eu de nombreuses preuves de l'irrationnalité de $\sqrt{2}$ et de l'infinité de l'ensemble des nombres premiers, mais elles ont été utilisées pour réaliser des études de performance des systèmes.

Dans les deux dernières décennies du 20ème siècle, les assistants informatiques de preuves ont commencé à apparaître, qui avaient de nouvelles fonctionnalités. Dans ces systèmes ultérieurs, les utilisateurs pouvaient écrire des "tactiques". Les tactiques sont du code informatique qui rassemblent les applications des axiomes et les principes ensemble. Par exemple dans un assistant moderne de preuves comme Lean, $(x+y)(x+2y)(x+3y) = x^3 + 6x^2y + 11xy^2 + 6y^3$ peut maintenant être démontré en une ligne en faisant appel à la tactique `ring` [4]. Les stratégies permettent aux mathématiques formalisées de ressembler davantage à la pratique des mathématiques ordinaires en rendant automatiques les choses "évidentes".

1.2. Le théorème des nombres premiers. En 2004, une équipe composée de Jeremy Avigad, Kevin Donnelly, David Gray et Paul Raff a démontré formellement le théorème des nombres premiers dans le système Isabelle/HOL. La preuve qu'ils ont formalisée était la "preuve élémentaire" de Erdös–Selberg. Le travail utilisait des entrées à la fois d'arithmétique et d'analyse réelle basique. Bien sûr, les calculs faisaient intervenir la croissance de fonctions qui semble facile sur papier mais qui a nécessité du temps et des efforts pour être formalisée. La manipulation d'inégalités qui semblent faciles à des humains nécessitent d'être effectuées soit à la main soit via une tactique d'élimination de Fourier–Motzkin dans un assistant informatique de preuve. La raison pour laquelle la preuve d'Erdös–Selberg était préférée à la preuve traditionnelle en analyse complexe était qu'à ce moment-là Isabelle/HOL n'avait pas de bibliothèque d'analyse complexe.

Ce que nous en concluons, c'est que vers 2004, un matériau plus sérieux de niveau universitaire était dès lors accessible en théorie à ces systèmes, au moins dans quelques domaines des mathématiques. On voit également que nous sommes à une étape où les bibliothèques de preuves dans les différents domaines des mathématiques sont capables d'interagir les unes avec les autres.

En 2009 John Harrison a formalisé la preuve en analyse complexe du théorème des nombres premiers dans l'assistant de preuves HOL Light [Har09a], motivé en partie par le fait qu'HOL Light contenait déjà des éléments de théorie analytique complexe incluant la formule de l'intégrale de Cauchy. En 2016 Mario Carneiro a formalisé la preuve élémentaire d'Erdös–Selberg en Metamath, un assistant basé sur la théorie des ensembles qui n'a essentiellement pas de tactique ; comme vous pouvez l'imaginer, cela a été un effort héroïque.

Ainsi, le théorème des nombres premiers est devenu une sorte de publicité pour la formalisation. On peut comprendre pourquoi - c'était un théorème célèbre en mathématiques, sa preuve n'est pas du tout triviale, et toute formalisation dans un démonstrateur de théorèmes démontre que le démonstrateur est capable de raisonner simultanément sur le discret et le continu.

Cependant, comme cela peut devenir évident pour le lecteur, l'une des raisons pour lesquelles le résultat était formalisé indépendamment dans plusieurs démonstrateurs de théorèmes était qu'il est extrêmement difficile de traduire une preuve écrite dans l'un des systèmes en une preuve dans un autre système. L'un des problèmes est que différents systèmes peuvent avoir des fondements différents ; par exemple HOL Light et Isabelle/HOL sont des systèmes de théorie des types, et Metamath est un système de théorie des ensembles. Un autre problème est que même si deux systèmes de preuve ont des fondements très similaires, ils peuvent avoir des *idiomes* différents ; différentes bibliothèques dans différents systèmes pourraient être configurées pour faire la même chose de manières très différentes. Sans devenir trop technique, pour que ces systèmes de preuve informatiques fonctionnent, il faut disposer d'une sorte de méthode pour se déplacer entre les structures "dans les coulisses" - par exemple, les réels forment un corps, et donc ils forment un groupe additif (et un monoïde multiplicatif), et en particulier on veut que tous les théorèmes sur les groupes additifs tels que $0 + a = a$ s'appliquent instantanément à des corps tels que les réels sans prise de tête. Les humains n'ont bien sûr aucun problème avec cela, mais dans un système de preuve informatique, il faut une sorte d'infrastructure qui rend cela automatique, et si différents systèmes le font de différentes manières, cela rend bien sûr la traduction

[4] Voir [GM05] pour une description des sortes de problèmes qui adviennent quand on écrit de telles tactiques.

automatique de la preuve beaucoup plus difficile.

Ainsi, j'ai été très surpris lorsqu'en 2020, Mario Carneiro a annoncé qu'il avait utilisé son projet Metamath Zero [Car21] pour porter la preuve Metamath du théorème des nombres premiers vers Lean. Les deux systèmes sont à peu près aussi éloignés qu'il est possible de l'être - Metamath utilise la théorie des ensembles comme base et Lean utilise la théorie des types, par exemple. Les preuves Metamath sont généralement de beaucoup plus bas niveau, avec une automatisation limitée disponible, alors que les preuves Lean typiques sont très lourdes en tactiques. Cependant le système fonctionnait, et produisait du code qui se compilait ; la preuve était bien sûr aussi illisible. C'était des dizaines de milliers de lignes de code primitif complètement non motivé définissant des variables et appliquant des principes de base de la logique, sans aucun commentaire. En fait, c'était un merveilleux exemple de quelque chose qui satisfaisait une définition formelle d'"être une preuve", tout en ne donnant en quelque sorte aucune information au lecteur humain autre que le fait que le théorème était vrai.

Bien sûr, si les ordinateurs commencent à écrire des preuves par eux-mêmes, ils pourraient tous ressembler à ça, au moins au début.

1.3. Le théorème des 4 couleurs. Le théorème des quatre couleurs (anciennement la conjecture des quatre couleurs) était un problème notoire en théorie des graphes soulevé dans les années 1850 et qui est resté non résolu pendant plus de 100 ans. Une formulation de celui-ci est l'affirmation que les sommets de chaque graphe planaire peuvent être colorés avec quatre couleurs de telle manière que aucuns deux sommets adjacents ne partagent une couleur. L'énoncé est un problème combinatoire élégant, et certains membres de la communauté mathématique ont été choqués par le fait que la preuve, annoncée par Appel et Haken en 1976, utilisait un ordinateur de manière essentielle. Appel et Haken ont construit une collection de 1834 graphes avec la propriété qu'un contre-exemple minimal doit contenir un de ces graphes en tant que sous-graphe, mais qu'à l'inverse aucun graphe contenant un de ces 1834 graphes en tant que sous-graphe ne peut être un contre-exemple minimal. La vérification de ces assertions se faisait à l'aide d'un programme informatique sur mesure qui, à l'époque, nécessitait un mois pour s'exécuter. La preuve Appel-Haken avait une valeur aberrante car si le principe de la preuve était possible à comprendre, les détails étaient trop difficiles à suivre pour un humain dans la pratique ; un milliard de cas à distinguer (c'est à quoi ressemble la partie informatique de la preuve), ce n'est pas quelque chose que les humains peuvent faire manuellement et avec précision dans un délai raisonnable. La preuve repose essentiellement sur un calcul informatique et donc sur l'exactitude du code informatique. Les petits bogues dans le code informatique sont bien sûr monnaie courante, même si l'on pourrait bien sûr ajouter que les petits bogues dans les preuves écrites par l'homme sont également monnaie courante. Cependant, la communauté mathématique est bien équipée pour découvrir et corriger de petits bogues dans les preuves écrites humaines, et elle était peut-être moins bien équipée pour vérifier l'exactitude du code informatique, en particulier en 1976.

En 2004, Georges Gonthier a terminé une vérification formelle du résultat d'Appel–Haken – plus précisément il a formalisé la variante Robertson–Sanders–Seymour–Thomas de 1997 de l'argument [Gon07]. Le travail comprenait 60 000 lignes de code écrites dans l'assistant de preuve Coq. En particulier, ce projet éclipse complètement le projet des nombres premiers discuté dans la section précédente. Il contient une formalisation complète de la partie théorique du travail – preuves formelles de résultats en topologie (pour réduire l'énoncé sur les graphes planaires arbitraires à un énoncé de nature combinatoire discrète) et en théorie des graphes – tout en vérifiant formellement le calcul informatique nécessaire pour finir la preuve. Notez en particulier que (comme dans la preuve du théorème des nombres premiers) une grande partie du travail consistait à écrire du matériau de base plutôt qu'à formaliser la preuve elle-même.

Il est intéressant de noter que le processus de formalisation a conduit à des simplifications dans l'argumentation. Par exemple, Gonthier a développé une théorie de ce qu'il a appelé les hypercartes combinatoires, ce qui a considérablement réduit la quantité de topologie nécessaire dans la preuve, et en particulier a supprimé la dépendance de l'argument sur le théorème de la courbe de Jordan. Gonthier a développé des mathématiques originales dans le cadre de ses travaux - par exemple, il a isolé un critère combinatoire pour ses hypercartes qui équivalait à la planarité.

Naïvement, il semble que dans ce cas, nous remplaçons une “preuve par ordinateur” par une autre, mais en pensant cela, on rate le point essentiel. Premièrement, la formalisation Coq couvre non seulement le code informatique d'Appel-Haken, mais aussi tout le reste de l'argument d'Appel-Haken. Deuxièmement, on peut considérer la vérification formelle comme un contrôle indépendant de la preuve. Enfin, au lieu de devoir faire confiance au code écrit par Appel et Haken et que peu de gens ont lu, on doit plutôt faire confiance au code écrit par les auteurs de Coq. Coq existe depuis longtemps (la première version a été écrite en 1984), a un petit noyau et le système compte de nombreux utilisateurs. Un bogue qui signifiait que Coq pouvait affirmer à tort qu'un théorème non prouvé était vrai ne se manifesterait probablement pas dans un seul projet et serait bien plus susceptible d'être finalement découvert. En revanche, le code d'Appel-Haken est un morceau de code sur mesure avec peu d'utilisateurs, donc les bogues sont sans doute plus probables.

Gonthier a écrit un article très instructif sur son travail [Gon08] pour les Notices of the American Mathematical Society (incluant une présentation de la théorie des hypercartes), dans le cadre du numéro de novembre 2008 ; ce numéro était consacré à la vérification formelle des mathématiques dans un système de preuve informatique et il fournit une excellente vue d'ensemble du domaine tel qu'il se présentait alors.

1.4. Le théorème de l'ordre impair. Le théorème de l'ordre impair. Le théorème d'ordre impair est le théorème selon lequel tout groupe fini d'ordre impair est résoluble. En 2013 une équipe de 15 personnes dirigée par Gonthier a vérifié formellement une preuve de ce théorème en Coq [GAA+13]. Cet ouvrage est remarquable pour plusieurs raisons. Premièrement, la preuve est très longue ; un argumentaire complet (modulo les bases en théorie des groupes et théorie des représentations) est présenté dans les deux tomes [BG94] et [Pet00]. Deuxièmement, nous sommes allés bien au-delà des mathématiques de niveau mastère ici - ce travail était l'une des raisons pour lesquelles Thompson a reçu la médaille Fields en 1970. La preuve est un argument très délicat dans la théorie des groupes finis, dont une grande partie implique l'analyse de la structure d'un contre-exemple minimal et montrant finalement qu'il ne peut pas exister. La preuve Coq impliquait de formaliser les deux livres mentionnés ci-dessus, plus bien sûr tout le matériel de base en théorie des groupes, théorie des représentations, théorie de Galois et théorie des nombres ; en effet, la formalisation du matériau de base a pris une grande partie des six années que les auteurs ont passées sur la preuve. Comprendre comment gérer un projet de formalisation d'une telle envergure était également une tâche non triviale.

Cela vaut peut-être la peine de prendre du recul et de se demander comment un travail comme celui-ci contribue à la compréhension humaine. La réponse naïve à cela est “cela garantit que la preuve humaine est correcte”. Cependant, à mon avis, ce n'est pas la principale contribution. Les humains étaient bien conscients, même dans les années 1960, que la preuve était correcte - s'il y avait eu le moindre doute, Thompson n'aurait pas obtenu la médaille Fields. Ce que les travaux de formalisation nous montrent, c'est que les démonstrateurs de théorèmes sont maintenant devenus capables d'opérer à ce genre d'échelle. Des livres entiers de mathématiques peuvent désormais être formalisés dans un seul système sans que le système ne manque de mémoire ou ne s'arrête. En moyenne, une ligne de mathématiques dans [BG94] ou [Pet00] correspondait à cinq lignes de code informatique, nous apprenons donc qu'en 2013, le soi-disant “facteur de de Bruijn” pour ce type de mathématiques est d'environ 5. Cependant, ce rapport ne doit pas être pris trop au sérieux : dans certaines parties de l'argumentation, le rapport est essentiellement de un, et dans d'autres parties, il est beaucoup plus grand. Notez également que ce facteur peut varier considérablement d'un assistant de preuves à un autre.

Nous apprenons également que de grands projets de formalisation comme celui-ci sont un moyen très efficace de motiver le développement de bibliothèques mathématiques fondamentales. Une des conséquences de ce projet de formalisation a été que Coq a développé une bibliothèque très solide d'algèbre de premier cycle qui peut bien sûr être utilisée (et est utilisée) pour d'autres projets.

Le rapport de travail [GAA+13] sur l'ordre impair est une lecture intéressante. Certaines sections se concentrent sur les mathématiques ou l'histoire, mais il y a aussi une discussion sur les mathématiques constructives, quelque chose qui, à mon avis, n'aurait rien à voir avec le travail, et aussi sur les problèmes de mise en œuvre, autre chose que les mathématiciens n'ont généralement jamais. réfléchir au sujet de. Par exemple, une observation faite dans la section 3 était que de nombreux théorèmes impliquant deux groupes finis ou plus seraient généralement formalisés en supposant que ces groupes étaient tous deux des sous-groupes d'un groupe fini ambiant X plus grand. Cela peut être fait sans aucune perte de généralité bien sûr, car étant donnés

deux groupes G et H, ils sont tous les deux des sous-groupes de $G \times H$. Pourquoi cette observation est-elle importante ? Il s'agit d'un problème d'implémentation, du domaine de l'informaticien. Travailler avec des sous-groupes plutôt qu'avec des groupes peut être plus facile ou plus agréable lorsqu'il s'agit d'implémenter certains théorèmes dans le démonstrateur de théorèmes. Il est à noter cependant qu'une telle astuce ne marche plus généralement pas : par exemple en géométrie algébrique on utilise la catégorie des anneaux commutatifs avec 1, et les morphismes envoient par définition 1 vers 1. Si R et S sont des anneaux commutatifs généraux avec 1 alors il n'y a en général pas de morphisme d'anneaux de R vers $R \times S$ envoyant 1 sur 1, on est donc obligé d'implémenter la théorie des anneaux commutatifs d'une manière plus "traditionnelle". Voir [Wie21] pour savoir comment cela a été fait dans Lean.

En ce qui concerne le constructivisme - les auteurs de l'ouvrage ont fait beaucoup d'efforts pour garder leur preuve "constructive", par exemple en évitant toute utilisation des nombres complexes lors de la mise en place des bases de la théorie des représentations. Les nombres complexes n'ont pas d'égalité décidable, ce qui signifie qu'il n'y a en général aucun algorithme pour prouver que deux nombres complexes définis de manière constructive sont égaux (par exemple, on peut évaluer numériquement une intégrale définie et observer qu'elle semble être de 0 pour 1000 décimales, mais il n'y a pas d'algorithme générique que l'on puisse appliquer à une intégrale arbitraire pour décider si elle est égale à 0 ou pas). Cela signifie qu'en mathématiques constructives, où la loi du tiers exclu ne peut pas être supposée, on ne peut pas faire un cas divisé selon que $z = 0$ ou non, si z est un nombre complexe, et plus généralement beaucoup de constructions deviennent non calculables et il est donc beaucoup plus difficile de raisonner de manière constructive. Ces choix de conception augmentent donc la quantité de travail nécessaire pour faire fonctionner la théorie des représentations. J'avais pensé que les constructivistes s'étaient éteints au début du 20$^{\text{ème}}$ siècle. Il s'avère qu'ils sont bien vivants et qu'ils travaillent généralement de nos jours dans des départements d'informatique. Une des raisons à cela est que le constructivisme joue un rôle important dans la théorie des langages de programmation. La réticence à utiliser la loi du tiers exclu est dans une certaine mesure une décision culturelle. Mais il existe aussi des situations où le travail constructif permet à un système de preuve informatique de prouver "automatiquement" certains résultats (par exemple par un calcul explicite). Bien que travailler de manière constructive ait pu être faisable pour un projet sur les groupes finis, la loi du tiers exclu est utilisée dans la plupart des mathématiques modernes au niveau de la recherche et il n'est pas vraiment possible de travailler de manière constructive lorsqu'on fait le type de mathématiques qui se font de nos jours dans les départements de mathématiques. Cependant, il convient également de souligner que la plupart des assistants de preuve modernes n'ont aucun problème avec la loi du tiers exclu, l'axiome du choix et d'autres axiomes non constructifs - ils sont disponibles, si vous le souhaitez. Certaines tactiques de preuve sont exclues si l'on choisit de travailler de manière non constructive, mais on peut contrer cela en écrivant de nouvelles tactiques spécifiquement conçues pour effectuer des calculs dans des domaines tels que les nombres réels et complexes. Les axiomes non constructifs sont largement utilisés dans la bibliothèque mathématique de Lean, `mathlib`, par exemple.

1.5. La conjecture de Kepler. La conjecture de Kepler stipule que l'emballage cubique à faces centrées est le moyen le plus dense d'emballer des sphères congruentes dans l'espace 3. Hales et Ferguson ont prouvé la conjecture en 1998 ; elle était alors ouvert depuis plus de 350 ans (elle avait été stipulé par Kepler avant que Fermat ne propose son dernier théorème). Une partie de la preuve de Hales–Ferguson impliquait la vérification de plus de 23 000 inégalités non linéaires sur un ordinateur ; une autre partie impliquait une classification informatique de tous les graphes recensés. D'autres calculs informatiques ont également été impliqués. À cet égard, la preuve est similaire à la preuve d'Appel–Haken du théorème des quatre couleurs ; des calculs doivent être effectués, qui sont tout simplement beaucoup trop longs pour que les humains puissent les faire dans un délai raisonnable.

Parce que le résultat était considéré comme important, les referees se sont sentis obligés d'essayer de vérifier la partie informatique de la preuve d'une manière ou d'une autre ; cependant, ils ont finalement abandonné, et dans [HAB+17] Hales déclare que l'article a été publié (dans les Annales) sans certification complète des referees. En 2003, Hales a annoncé un projet visant à vérifier formellement la preuve à l'aide de systèmes de preuve informatiques. Hales a utilisé un combinaison de HOL Light et d'Isabelle/HOL, et le projet s'est transformé en une collaboration internationale, avec 22 auteurs répertoriés sur l'article final. Le projet de formalisation a duré environ 12 ans et il comprenait plus d'un demi-million de lignes de code. Tout comme pour les autres projets de cette section, l'un des principaux avantages du travail pour la communauté de la

preuve formelle est que la bibliothèque standard de HOL Light s'est développée pour inclure des théorèmes tels que le théorème du point fixe de Brouwer, le théorème de Krein–Milman et le Théorème de Stone–Weierstrass.

En 2017, Hales a donné une conférence [Hal] au Newton Institute dans laquelle il a raconté l'histoire de la preuve de Kepler et il a expliqué sa vision de l'avenir des mathématiques formalisées. Cette conférence a été, pour moi, le tournant, et a été l'une des principales motivations derrière le travail décrit dans la sous-section suivante.

1.6. Les espaces perfectoïdes. Les résultats formalisés précédents ont tous quelque chose en commun. Alors que certaines d'entre elles représentent des mathématiques vraiment profondes, toutes les preuves formalisées impliquent un raisonnement sur des objets qui sont en quelque sorte *élémentaires* (graphes planaires, nombres premiers, groupes finis, sphères). De plus, la plupart (mais pas la totalité) de la formalisation effectuée avant 2017 était effectuée par des informaticiens. Dans le discours de Hales lié à ce qui précède, il a soutenu de manière cohérente que pour de nouveaux progrès dans ce domaine, cet état de choses devait changer. À cette époque, je venais tout juste de commencer à m'intéresser aux assistants informatiques de preuve et mes plans initiaux étaient de tenter de les intégrer à mon enseignement de premier cycle. Cependant, les arguments de Hales ont résonné en moi et, en quelques mois, je me suis retrouvé à travailler avec des étudiants de premier cycle à l'Imperial College, formalisant la définition et les propriétés de base des schémas dans le démonstrateur de théorèmes Lean. Ce projet consistait à développer des théories fondamentales de localisation des anneaux et des faisceaux sur les espaces topologiques ; cependant c'était relativement simple (modulo de mauvaises décisions de conception ; le lecteur intéressé par plus de détails peut les trouver dans [Hal]). J'ai donc été choqué de découvrir par la suite que les schémas – une notion si fondamentale en géométrie algébrique – n'avaient été auparavant formalisés dans aucun autre système de preuve informatique ! De plus, le projet m'a fait comprendre que formaliser des objets mathématiques beaucoup plus lourds devrait être possible.

Fin 2017, Patrick Massot (un topologue) et moi-même avons eu indépendamment l'idée de formaliser les espaces perfectoïdes ; le sujet était dans l'air car c'était à l'époque un secret de polichinelle que Scholze allait recevoir une médaille Fields pour son invention / découverte du concept et ses applications à la géométrie arithmétique. Je connaissais la définition mathématique, ayant moi-même touché au domaine, et lorsque Johan Commelin, un autre géomètre arithmétique, est apparu dans le forum de discussion Lean Zulip en 2018, nous avons tous les trois décidé de nous lancer. Environ 16 000 lignes de code et huit mois plus tard, nous avions une définition formalisée ; on pourrait résumer le travail comme une formalisation informatique de la seule ligne de mathématiques "Soit X un espace perfectoïde"[5].

Le travail était bien sûr en partie conçu comme un coup de pub ; les informaticiens étaient bien conscients de l'existence de démonstrateurs de théorèmes informatiques, mais les mathématiciens semblaient ne pas l'être, et c'était une tentative pour les leur faire remarquer. Le plan a été un succès - le projet a semblé rehausser le profil des démonstrateurs de théorèmes informatiques au sein de la communauté des mathématiques. Notez cependant que nous n'avons pas construit d'exemples d'espaces perfectoïdes autres que l'espace perfectoïde vide et tous les trois nous étions bien au courant des problèmes nous empêchant de formaliser aucun des théorèmes sérieux de Scholze à propos des espaces perfectoïdes à ce moment-là ; il nous manquait tant de prérequis. Comme avec les projets précédents, un gain tangible à partir de ce travail était l'agrandissement de la bibliothèque mathématique du système en question. La plupart des résultats de la topologie générale de Bourbaki a fini par faire partie de la bibliothèque mathématique `mathlib` comme résultat de ce projet, ainsi que de nombreux autres résultats d'algèbre topologique, et cela a également motivé les débuts d'une théorie des valuations et des corps de valuation discrète.

On peut considérer que le travail sur les espaces perfectoïdes est orthogonal dans un certain sens à celui qui est effectué dans un assistant de preuve. Beaucoup des résultats précédents soulignés dans cette section sont des preuves de théorèmes longs et complexes à propos d'objets relativement simples. La preuve qu'on peut donner à l'ensemble vide la structure d'espace perfectoïde est un théorème très simple à propos d'un concept

[5]Dans la formalisation de l'ordre impair, le facteur de de Bruijn (ratio de lignes de code rapporté au nombre de lignes de texte écrit humain) était d'environ 5. Ici, on peut arguer qu'il est de 16 000. Pourtant, on pourrait également arguer que cela pourrait également prendre des milliers de lignes de texte humain pour définir complètement un espace perfectoïde.

beaucoup plus compliqué. Bien sûr la question naturelle est de savoir si les systèmes informatiques de preuves peuvent démontrer des théorèmes complexes à propos d'objets complexes. Un an après le projet des espaces perfectoïdes, on a commencé à le découvrir.

1.7. Mathématiques condensées. Clausen et Scholze ont développé une théorie des mathématiques condensées. Un ensemble condensé est une variante d'un espace topologique. L'idée principale est que les objets condensés peuvent avoir de meilleures propriétés homologiques que les objets topologiques (par exemple, la catégorie des groupes abéliens condensés est une catégorie abélienne, alors que la catégorie des groupes topologiques abéliens n'en est pas une). Ils espèrent que ces idées permettront à des techniques d'algèbre homologique de s'appliquer à de nouveaux domaines de la géométrie analytique. À la fin des années 2020, Scholze m'a approché et m'a demandé si nous avions un groupe de travail à l'Imperial college ; je répondis que nous en avions un. Scholze m'a alors demandé si nous avions regardé tous les détails de la preuve du théorème 9.1 de [Schb] ; je répondis que nous ne l'avions pas fait. Scholze fit alors la remarque qu'il avait eu la même réponse d'autres mathématiciens, et il a émis la possibilité que peut-être, personne d'autre que lui et Clausen n'ait jamais lu la preuve avec attention. De plus, il suggéra que peut-être cela pourrait rester vrai même après le processus de contrôle par les referee. La raison qui le motivait était que, pour Scholze, c'était le théorème sur lequel la théorie complète était appuyée. La preuve était très technique ; elle était construite sur un résultat intermédiaire plus "élémentaire" mais peu maniable, le théorème 9.4 de [Schb]. Scholze fut d'accord pour défier la communauté de la formalisation informatique pour prouver son théorème 9.1 dans un post de blog [Schc], publié ultérieurement ici [Sch21]. Bien que le défi était un défi à la communauté de la formalisation en général, il semble que seule la communauté Lean ait répondu ; cela n'est peut-être pas surprenant, puisque (peut-être seulement pour des raisons sociologiques), il s'est avéré que les mathématiciens intéressés par "les genres de mathématiques qui font gagner la médaille Fields" et qui sont également intéressés par les assistants informatiques de preuves ont tendance à graviter autour de Lean.

Johan Commelin devint de facto le meneur du processus de formalisation, avec Patrick Massot l'aidant à écrire un blueprint [CM] de la stratégie (c'est-à-dire, une feuille de route précise) et une équipe de théoriciens algébristes des nombres, de géomètres arithmétiques, et d'autres mathématiciens (Riccardo Brasca, Damiano Testa, Filippo Nuccio, Adam Topaz, moi-même, Patrick Massot, Bhavik Mehta...) a alors commencé à travailler sur le projet, avec l'aide occasionnelle d'autres personnes ayant un bagage informatique comme Mario Carneiro. En six mois, l'équipe avait grossi de plus d'une dizaine de personnes et nous avions formalisé une preuve complète du théorème 9.4 (voir [Cas]). Au moment où nous écrivons, nous n'avons pas déduit le théorème 9.1 [Schb], mais ce n'est qu'une question de temps. Un deuxième article de blog [Scha] de Scholze indique ses réflexions sur la question ; en particulier, nous voyons qu'il est maintenant beaucoup moins préoccupé par la situation concernant l'exactitude des résultats. De plus, Scholze a indiqué (communication personnelle) que le processus lui a permis de mieux comprendre ce qui alimente la preuve, et Commelin a non seulement appris les mathématiques dans le cadre du processus, mais a également simplifié l'argument à plusieurs endroits, notamment dans le suppression de la dépendance de l'argument aux travaux antérieurs de Breen et Deligne.

Pour moi, cela montre une évidence substantielle que maintenant *toutes les mathématiques pures* peuvent être formalisées dans des assistants informatiques de théorèmes – à la fois en théorie, et en pratique. Cela prend du temps, mais c'est possible. La formalisation du travail a amené à la fois à une meilleure compréhension de celui-ci, et à des simplifications de l'argumentation. Il est également à noter que, comme dans beaucoup d'autres projets de formalisation, une portion substantielle du temps a été dépensée à formaliser le matériau de base (par exemple la théorie des groupes normés et la théorie des espaces profinis). Comme les bibliothèques des solveurs deviennent meilleures et qu'elles commencent à contenir la sorte de matériau que les mathématiciens tiennent pour garanti, il y aura de moins en moins de tels "coûts de démarrage".

1.8. Autres résultats. Il existe bien d'autres exemples d'efforts sérieux de formalisation que nous n'avons pas la place de découvrir. Nous énumérons ici quelques exemples. Gouëzel a formalisé les définitions de base des variétés C^k et C^∞ dans Lean en étendant un travail précédent écrit en Isabelle/HOL. Mahboubi et Sibut-Pinote ont prouvé l'irrationalité de $\zeta(3)$ en Coq [CMSPT14] et Eberl l'a prouvée en Isabelle/HOL [Ebe19a]. Mahboubi a également fait des travaux approfondis sur les valeurs numériques rigoureuses des intégrales en Coq et, également en Coq, Bertot, Rideau et Théry ont formellement vérifié le premier million de décimales

de [BRT18]. Eberl a formalisé une grande partie du manuel d'Apostol sur la théorie analytique des nombres en Isabelle/HOL [Ebe19b]. Han et van Doorn ont prouvé l'indépendance de l'hypothèse du continu dans Lean [HvD20]. Immler a formellement vérifié les calculs de Tucker utilisés pour vérifier l'existence de l'attracteur étrange [Imm18]. Mehta et Dillies ont formellement vérifié le lemme de régularité de Szemerédi et le théorème de Roth sur les progressions arithmétiques dans Lean, et Edmonds, Koutsoukou-Argyraki et Paulson les ont vérifiés dans Isabelle/HOL [EKAP21]. Le théorème de Poincaré-–Bendixson a été formalisé par Immler et Tan [IT20] dans Isabelle/HOL ; notez que la preuve habituelle telle que comprise par les mathématiciens repose sur des dessins, et la formalisation des dessins peut être un travail difficile. La résolution d'Ellenberg—Gijswijt de la conjecture de cap set a été vérifiée dans Lean par Dahmen, Hölzl et Lewis [DHL19]. Commelin et Lewis ont construit des vecteurs de Witt et montré que $W(\mathbb{F}_p) = \mathbb{Z}_p$ dans [CL21] ; ce travail est intéressant car non seulement ils ont formalisé les mathématiques délicates impliquées, mais ils ont aussi écrit des tactiques qui leur permettraient de réduire sans douleur diverses questions au cas universel. La finitude du groupe de classes d'un corps global a été prouvée en Lean par Baanen, Dahmen, Narayanan et Nuccio dans [BDNdC21] (cela m'étonne encore que ce résultat, dont les cas particuliers étaient connus de Gauss et qui est un théorème standard qu'on étudie en préparant un diplôme de premier cycle en mathématiques, a été formalisé dans un assistant de preuve pour la première fois en 2021 ; ce développement est une démonstration de la façon dont les intérêts mathématiques de la communauté de la formalisation englobent désormais des travaux que les personnes dans les départements de mathématiques pourraient considérer comme "courantes").

Il y a aussi du travail en cours (au moment où cet article est écrit). Des équipes de personnes qui collaborent au forum d'entraide Zulip de Lean [Zul] travaillent en ce moment sur une preuve du dernier théorème de Fermat pour les nombres premiers réguliers, et sur le théorème de Smale qu'il est possible d'inverser une sphère. Un projet général de formaliser de nombreux résultats basiques dans la théorie des schémas est également en cours.

2. `mathlib`

Dans cette section, je donnerai un aperçu de la bibliothèque mathématique de Lean, l'une des plus grandes collection monolithique de mathématiques formalisées existante, et, de façon plus importante, celle qui connaît une croissance rapide. D'une certaine façon, c'est une perspective personnelle ; un point de vue différent, qui parle davantage de l'informatique permettant cette bibliothèque est présenté dans [`mathlib`c20].

Le développeur principal de l'assistant de preuve de théorèmes Lean est Leonardo de Moura, qui a commencé ce projet en 2013. Lors de la Conférence *Grande preuve* de 2017 à Cambridge, il a été décidé de séparer la plupart des parties mathématiques du solveur de la partie "cœur", et de déplacer les mathématiques dans une bibliothèque spécifique. Ainsi est né `mathlib`. Maintenant, `mathlib` contient des définitions des groupes, des anneaux et des espaces topologiques, des filtres, une construction des nombres rationnels (les nombres naturels et les entiers sont restés dans le cœur de Lean), et peu de choses autres. Johannes Hölzl et Mario Carneiro maintiennent la bibliothèque, et entre eux, ils ont commencé à lentement construire davantage de mathématiques, par exemple, les nombres réels. Hölzl a écrit beaucoup de la partie topologique du dépôt, suivant l'approche de Isabelle/HOL qui repose profondément sur le concept de filtre. Carneiro a écrit une théorie robuste de la finitude, et lentement, la bibliothèque a commencé à devenir pertinente pour le "mathématicien au travail".

La bibliothèque est un projet libre et open source. Elle est monolithique au sens où il y a une définition d'un groupe, une définition d'un anneau, une définition des nombres réels, et toutes ces définitions peuvent être importées simultanément et interagir les unes avec les autres. Initialement, les buts de la bibliothèque n'étaient pas clairs, si ce n'est qu'elle offrirait un environnement où les gens pourraient expérimenter le fait de faire des mathématiques en Lean. Des mathématiciens comme Scott Morrison, moi-même et Patrick Massot nous sommes impliqués au tout début, et parce que nous avions reçu une formation en mathématiques et en logique classique (i.e. loi du tiers exclu) et d'autres axiomes tels que l'axiome du choix, la bibliothèque s'est développée avec ces supposés classiques dans son cœur. Chaque projet mathématique réussi écrit en Lean et rendu possible par `mathlib` a semblé attirer davantage de mathématiciens dans le forum d'entraide, ce qui en retour a amené à davantage de projets. En une paire d'années, Lewis avait formalisé les nombres p-adiques [Lew19], moi-même et un groupe d'étudiants (Lau, Hughes, Livingston et Fernández Mir) avions

formalisé les schémas [BHL+21], Dahmen, Hölzl et Lewis ont formalisé la preuve de 2017 de Ellenberg–Gijswijt des Annals de la conjecture cap set [DHL19], et Massot, Commelin et moi-même avons formalisé la définition des espaces perfectoïdes [BCM20]. Chacun de ces projets n'aurait pu avoir lieu sans `mathlib` ; inversement chacun de ces projets a contribué à la croissance de `mathlib`.

De nombreux développements ont également été effectués sans donner lieu à l'écriture d'articles, et dont le but principal était simplement d'augmenter `mathlib`. J'ai supervisé des projets d'étudiants dans lesquels en premier cycle, ils pouvaient formaliser le matériau qu'ils étaient en train d'apprendre en classe et ils l'ajoutaient à la bibliothèque ; par exemple, les théorèmes de Sylow (Chris Hughes), les groupes nilpotents (Ines Wright), les applications conformes (Yourong Zang) et le théorème de Radon–Nikodym (Kexing Ying) ont été ajoutés de cette manière. Amelia Livingston a développé une théorie de la localisation des monoïdes et des anneaux dont nous avions besoin en géométrie algébrique. J'ai demandé à des étudiants de premier cycle (Hughes, Lau, Lee) de formaliser un cours de théorie standard de Galois en Lean ; ils ont développé une théorie des extensions de corps, et le projet a ensuite été repris par un groupe d'étudiants en mathématiques de Berkeley (Miller, Browning, Lutz) qui ont fini le travail, en prouvant le théorème fondamental de la théorie de Galois et l'irrésolubilité de l'équation du cinquième degré [BL21] (notons que cela avait été formalisé en Coq juste deux mois avant [BCMS21]). Baanen, Dahmen, Narayanan et Nuccio ont formalisé une preuve de la finitude du groupes des classes d'un corps global [BDNdC21]. Je déposais de l'algèbre, mais d'autres déposaient de la géométrie et de l'analyse. Gouëzel et Macbeth ont développé une théorie des variétés, et Gouëzel et Kudryashov ont développé une théorie extensive du calcul à une ou plusieurs variables, incluant le théorème de fonction implicite et le théorème de Picard–Lindelöf. Gouëzel a aussi formalisé l'espace de Gromov-Hausdorff : un espace métrique paramétrant des espaces métriques compacts de Hausdorff à isométrie près.

Morrison a développé une grande quantité de théorie des catégories, et lui et Topaz ont maintenant formalisé les définitions des catégories abéliennes et le début du développement des foncteurs dérivés et l'algèbre homologique. Massot a développé la théorie de la valuation et une théorie de la complétion des espaces uniformes et des groupes topologiques et des anneaux. Tuma a développé la théorie des anneaux de Jacobson, et j'ai développé quelques éléments de base d'autres idées standards de l'algèbre commutative (modules projectifs et plats, anneaux de valuation discrète), et Springer, Kuelshammer, et beaucoup d'autres ont également contribué à l'algèbre. Hölzl a développé la théorie de la mesure de Lebesgue, et van Doorn a formalisé la mesure de Haar. Il y a beaucoup d'autres personnes qui ont fait des contributions (`mathlib` a maintenant plus de 200 contributeurs) et les nouveaux contributeurs sont toujours bienvenus. Les contributions sont revues par les personnes qui maintiennent Lean. Un des principes de la bibliothèque est de faire les choses "au niveau correct de généralité". Cela signifie, par exemple, que le calcul multi-variable et quelques intégrales exotiques prenant leur valeur dans des espaces vectoriels de Banach ont été développés en premier, et que le calcul à une seule variable a été déduit comme corollaire. La bibliothèque n'est pas optimisée pour être pédagogique ou lisible ; l'idée est de continuer à construire une solide base pour la sorte de mathématiques qui est étudiée dans un département de mathématiques contemporain.

Il est intéressant de noter que Lean semble apprendre les mathématiques à peu près à la même vitesse qu'un étudiant de premier cycle. Pendant les quatre années pendant lesquelles la bibliothèque a grossi, elle est allée à peu près d'un niveau nul a un niveau correspondant à un solide niveau de mastère en théorie des nombres et en algèbre commutative, et un niveau second cycle en analyse réelle. En analyse complexe, géométrie différentielle et théorie des représentations, ce n'est peut-être pas tout à fait un niveau de dernière année de second cycle, mais les choses bougent rapidement et cette phrase, écrite en 2021, sera vite obsolète. Pour une idée à jour de l'état courant de `mathlib`, la meilleure idée est de jeter un regard à l'aperçu complet fourni par la communauté de Lean mathlib [pcb], ou au résumé de mathématiques de niveau premier cycle que cet aperçu contient [pcc].

3. Un bref guide de la théorie des types

Dans cette section, nous expliquons les bases de la théorie des types et la façon dont elle peut être utilisée comme fondements des mathématiques. De nombreux assistants de preuves de théorèmes modernes utilisent une version de la théorie des types. Par exemple, Isabelle/HOL et les autres systèmes HOL utilisent la théorie des types simple, Lean et Coq utilisent la théorie des types dépendants, et les différents systèmes

HoTT développés par Voevodsky et autres utilisent la théorie des types d'homotopie. Il y a quelques assistants de preuves qui utilisent la théorie des ensembles – Metamath et Mizar sont les deux principaux – pourtant il n'est pas injuste de dire que de nos jours, la plupart des mathématiques faites dans des assistants sont faites dans un système de théorie des types, ainsi un mathématicien souhaitant se plonger dans les preuves formelles devrait au moins en connaître les bases, et ce sont ces bases que cette section tente de décrire.

3.1. Qu'est-ce qu'un type ? Les mathématiciens de nos jours sont habitués au mot "ensemble" qui flotte autour d'eux quand ils en viennent aux définitions de base. Par exemple, on apprend qu'un groupe est un ensemble équipé d'une multiplication telle que certains axiomes sont vérifiés. On ne nous dit cependant pas ce qu'est un ensemble ; un cours de théorie des ensembles ZFC nous dit une liste de propriétés que les ensembles *ont*, mais elles ne nous disent pas ce qu'*est* un ensemble. En effet, dans ce contexte, le mot "ensemble" n'a pas de définition formelle ; c'est simplement un terme générique pour un objet dans notre modèle des axiomes des mathématiques, et nous construisons d'autres objets mathématiques au-dessus de cet objet mathématique de base.

Dans les définitions telle que la définition d'un groupe, le mot "ensemble" est utilisé pour ne signifier rien de plus qu'une "collection d'éléments". En théorie des types, le rôle d'une "collection d'éléments" est joué par le *type*. Un type est une collection de termes. La définition d'un groupe en théorie des types est : un groupe est un type équipé d'une multiplication telle que certains axiomes sont vérifiés. La seule différence est la notation : le $a \in X$ de la théorie des ensembles est remplacé par le `a:X` de la théorie des types.

Comme mentionné ci-dessus, ceux d'entre nous qui ont suivi un cours de théorie des ensembles savent que, quand on utilise la théorie des ensembles comme fondements des mathématiques, *tout* est ensemble. Par exemple, les éléments d'un groupe sont, pour parler strictement, également des ensembles, donc on pourrait parler en théorie de leurs éléments aussi, bien que dans le contexte de la théorie des groupes, de telles questions semblent ne pas avoir de sens mathématiquement parlant, puisqu'ils ne sont pas invariants par isomorphismes. En théorie des types, cela n'est pas possible ; les éléments d'un type sont appelés des *termes*, et en général, les termes ne sont pas des types. En théorie des types, tout est un terme, et tout terme a un type, mais tous les termes ne sont pas des types. Par exemple, en théorie des types, $37\pi^2$ est un terme, dont le type est $\mathbb{R}$, le type des nombres réels. On écrit $37\pi^2 : \mathbb{R}$. Pourtant $x : 37\pi^2$ *n'a pas de sens*, parce que $37\pi^2$ n'est pas un type. Dans un assistant de preuve basé sur la théorie des ensembles, des questions telles que celle de demander si le groupe trivial est un élément de la fonction zeta de Riemann aurait du sens mais son sens ne serait pas mathématique – il dépendrait de décisions d'implémentation. La théorie des types fournit ainsi une vérification de base que ce que vous écrivez a un sens mathématique.

Dans un système de théorie des types, le type $\mathbb{R}$ est toujours construit à partir de $\mathbb{Q}$ comme les classes d'équivalence de suites de Cauchy, ou via les coupures de Dedekind, ou comme une autre des constructions standards ; la partie mathématique de l'histoire est identique au paradigme de la théorie des ensembles, c'est juste que le langage utilisé est légèrement différent (types et termes, plutôt qu'ensembles et éléments).

Une différence entre les types et les ensembles pourtant est que *les types ne se mélangent pas* : des types distincts sont disjoints. Cela a des avantages pratiques quand on formalise les mathématiques parce que cela fournit une forte vérification que les mathématiques que vous écrivez *font sens* : en théorie des types, si g est un élément du groupe G, alors le seul type dont g puisse être un terme est G.

Cette approche a pourtant des conséquences qui peuvent être perçues initialement comme un choc par un mathématicien. Par exemple, on pourrait construire un type représentant les réels positifs $\mathbb{R}_{>0}$ et un type représentant les réels $\mathbb{R}$, mais si un terme x avait le type $\mathbb{R}_{>0}$ alors x lui-même pourrait *ne pas* avoir le type $\mathbb{R}$ au sens strict ; j'insiste à nouveau sur le fait que tout terme a un *unique* type. Pour fabriquer un terme de type $\mathbb{R}_{>0}$, on doit fournir *deux* éléments de données : un nombre réel, et une preuve qu'il est positif. Un terme de type $\mathbb{R}_{>0}$ est un objet correspondant à cette paire, donc strictement parlant, ce n'est pas un nombre réel, et un système basé sur une théorie des types s'en tiendra à cela. Pourtant bien sûr, il y a une application canonique de $\mathbb{R}_{>0}$ vers $\mathbb{R}$ – vous n'avez qu'à juste jeter la preuve. Plus généralement, une théorie des types pourrait bien avoir un *système de contraintes*, consistant en une collection de "fonctions invisibles" envoyant des types sur d'autres types à la façon dont des mathématiciens s'y attendraient. Par exemple,

étant donné un terme de type $\mathbb{R}_{>0}$, il pourrait bien être possible de le nourrir avec une fonction attendant un terme de type $\mathbb{R}$; le système jetterait juste la preuve de positivité et utiliserait le nombre réel sous-jacent en tout cas. Les mathématiciens utilisent ces fonctions invisibles partout, sans même le remarquer. Nous avons déjà mentionné ci-dessus que dans un système de fondements, les nombres réels doivent être construits en utilisant l'une des constructions standards, par exemple via les séries de Cauchy. En particulier, un nombre rationnel n'est pas littéralement un nombre réel. Pourtant, en prenant Lean comme un exemple, si on a un terme `x:Q` alors on peut simplement écrire `x:R` pour obtenir le nombre réel correspondant, bien qu'une inspection attentive du terme correspondant devrait déterrer le fait que le nombre réel est vraiment appelé `↑ x`, indiquant que la contrainte a été appliquée. La contrainte est un homomorphisme d'anneaux, et Lean a des tactiques de "normalisation" [LM20] qui savent cela et qui appliqueront des théorèmes tels que `↑(x+y)=↑ x+↑ y` et `↑(x*y)=↑ x*↑ y` automatiquement (avant que cette tactique n'ait été écrite, faire des mathématiques qui impliquaient de switcher entre les entiers naturels, les relatifs et les rationnels était assez frustrant à cause de ces applications invisibles). En résumé donc, la théorie des types vous force à penser plus précisément à ce que sont les objets avec lesquels vous travaillez, bien que des tactiques puissent être utilisées pour manipuler ces objets de la manière dont on les manipule habituellement. Apprendre comment "diriger" les mathématiques de cette manière dans un solveur de théorèmes vient simplement avec la pratique.

3.2. Types inductifs. J'ai déjà mentionné que dans un système de théorie des types, la définition des nombres réels est la même que dans un système de théorie des ensembles – ce sont les séries de Cauchy, ou les coupures de Dedekind, ou n'importe quelle construction des réels que vous préférez. De façon similaire, les définitions des rationnels et des entiers travaillent comme des quotients et le font de la même manière en théorie des types et en théorie des ensembles. Mais là où les fondements en théorie des types et en théorie des ensembles diffèrent, c'est dans la définition des entiers naturels. Les nombres entiers naturels sont un objet fondamental en mathématiques – ce sont typiquement le premier exemple d'objet infini à construire – et il n'est donc peut-être pas surprenant que différents systèmes de fondements les traitent de différentes manières.

Dans la théorie des ensembles ZFC, l'existence de l'ensemble des nombres entiers naturels est postulée comme un axiome, notamment l'axiome de l'infinité. Des théories des types comme celle de Lean par exemple autorisent l'utilisateur à définir l'usage de *types inductifs*. De tels types incluent les entiers naturels et d'autres constructions récursives. Les détails de l'implémentation du calcul sur de telles constructions inductives [CP88] diffèrent selon les systèmes ; la suite de cette section explique les détails spécifiques à la théorie des types de Lean, mais beaucoup de ce que je dis s'applique à Coq et Agda, d'autres solveurs populaires utilisant la théorie des types.

En Lean, la définition des entiers naturels ressemble à ça :

```
inductive nat
| zero :  nat
| succ (n :  nat) :  nat
```

Cette définition dit que "zéro est un entier naturel, le successeur d'un entier naturel est un entier naturel, et c'est tout". Comme on peut le deviner, la construction inductive peut être utilisée pour construire des types beaucoup plus exotiques, mais on peut montrer que n'importe quel type qui peut être défini en utilisant les règles de calcul des constructions inductives correspond à un ensemble qui peut être construit en utilisant les axiomes habituels de la théorie des ensembles.

Voyons ce qui se cache sous le capot quand les entiers naturels sont définis comme un type inductif. Quand une telle définition est faite, un nouveau type `nat` apparaît dans le système, ainsi que le terme `nat.zero` et la fonction `nat.succ : nat → nat`. Les derniers termes sont appelés des *constructeurs* : ce sont des manières de fabriquer des entiers naturels. Pourtant une chose de plus apparaît également, notamment l'*éliminateur* pour le type – l'objet qui permet à l'utilisateur de construire des fonctions dont le domaine est celui des entiers naturels et dont le codomaine est quelque chose d'autre. Cela représente l'idée que la seule manière qu'on a de construire des entiers naturels est de le faire via `nat.zero` et `nat.succ`, et cela énonce que pour définir une fonction qui sort des entiers naturels, il suffit de (1) dire où va `nat.zero`, et (2) dire où va `nat.succ n` selon où `n` est allé. En d'autres termes, c'est le principe de récursion.

C'est donc ainsi que de nouveaux types inductifs naissent en Lean ; après leur définition, ils sont, avec leurs constructeurs et leur éliminateur, automatiquement ajoutés par l'assistant de preuves au système comme de nouvelles constantes, ou axiomes, ou quelque soit la manière dont vous voudriez y penser. Il y a bien sûr des règles précises nous disant la forme exacte de l'éliminateur pour chaque type inductif ; nous n'allons pas entrer dans le détail ici. Du point de vue des fondements, cette approche, où de nouveaux axiomes apparaissent "par magie" lorsque des types sont construits, est très différente du point de vue de la théorie des ensembles, bien que dans [Wer97], il soit montré que la théorie des types avec ces constructions est équi-consistante avec la théorie des ensembles. La stratégie de la preuve est de faire un modèle de la théorie des ensembles dans la théorie des types, et de faire un modèle de la théorie des types dans la théorie des ensembles. Pour un énoncé plus précis, on doit être plus explicite à propos de la théorie des types particulière dans laquelle on travaille. Par exemple, la thèse de troisième cycle de Mario Carneiro [Car] montre que la théorie des types de Lean est équi-consistante avec ZFC plus un nombre dénombrable de cardinaux inaccessibles.

Il est à noter, et assez amusant, que l'égalité elle-même est définie comme un type inductif dans de nombreux systèmes de théorie des types. Ceci contraste avec la théorie des ensembles, où l'égalité est typiquement considérée comme faisant partie de la logique. En effet, l'égalité en théorie des types est généralement plus subtile qu'en théorie des ensembles. Voici la définition de l'égalité de Lean :

```
inductive eq X : Type :  X → X → Prop
| refl (a :  X) : eq a a
```

Le légèrement énervant `X → X → Prop`, parenthésé ainsi `X → (X → Prop)`, signifie que l'égalité est une fonction qui prend en entrée un élément de `X` et qui fournit en sortie une fonction qui prend en entrée un élément de `X` et fournit en sortie une Proposition, c'est-à-dire un énoncé vrai-faux. En d'autres termes, si `a` et `b` sont des termes de type `X` alors `eq a b` est un énoncé vrai-faux. En utilisant la notation habituelle `a = b` pour `eq a b`, on voit que l'égalité de termes d'un certain type `X` est un type inductif avec un constructeur, notamment `eq.refl a`, une preuve que `a = a`. Il s'avère qu'à partir de cette définition, nous pouvons *prouver* toutes les propriétés habituelles de l'égalité ! L'éliminateur pour le type égalité est la *propriété de substitution*, qui si $a = b$ alors étant donné un terme de type $P(a)$ permet d'obtenir un terme de type $P(b)$. C'est un jeu assez plaisant de continuer à partir de là pour déduire que l'égalité est à la fois symétrique et transitive (pour plus de détails de cela, voir par exemple [Buzb]). Bien sûr, alors que c'est intéressant de voir comment les propriétés de base de l'égalité peuvent être prouvées dans un système de théorie des types, cela vaut aussi le coup d'insister sur le fait que pour utiliser un assistant informatique de preuve, on doit tout connaître des types.

3.3. Types dépendants. Lean et Coq utilisent tous deux une théorie des types appelée théorie des types dépendants, il vaut peut-être la peine de prendre un peu de temps pour expliquer ce qu'est un type dépendant. Imaginons que X est un objet géométrique, par exemple une variété réelle. Disons que l'on a une fibration vectorielle sur X, c'est-à-dire que, pour chaque point x de X on a un espace vectoriel V_x (qui varie continûment avec x d'une façon appropriée). Une section de cette fibration est une fonction qui prend en entrée un point x dans X et fournit en sortie un élément de V_x. Du point de vue des fondements, il y a deux manières de penser à une telle section. On peut regarder cette section comme une fonction de X vers l'union disjointe des V_x, envoyant $x \in X$ vers un élément de V_x. Alternativement, on peut regarder cette section comme une sorte légèrement différente de "fonction" qui a comme domaine X mais dont le codomaine varie continûment selon cette donnée en entrée. Certaines fois, en mathématiques, prendre l'union disjointe des codomaines est une chose naturelle à faire – par exemple dans l'exemple ci-dessus, l'union disjointe des V_x est naturellement un espace V situé au-dessus de X. Pourtant certaines fois, prendre l'union disjointe est assez non naturel. Par exemple, en géométrie algébrique, une manière de définir les sections du faisceau structurel d'un schéma affine $\mathrm{Spec}(R)$ est de les voir comme les fonctions qui envoient un idéal premier P de R vers un élément de la localisation R_P de R en P, et l'union disjointe des R_P lorsque P parcourt les idéaux premiers de R n'a pas de structure algébrique naturelle. L'ensemble ou le type consistant en l'union disjointe de ces anneaux locaux ne fait typiquement pas partie du modèle mental qu'a un géomètre algébriste lorsqu'il décrit ces sections.

Ces sortes de "fonctions" qui ont un domaine bien défini, mais un codomaine qui peut varier selon la donnée en entrée, sont appelées des fonctions dépendantes. Tous les assistants informatiques de preuves n'ont pas de telles fonctions ; par exemple, Isabelle/HOL (un assistant de preuves puissant qui contient beaucoup d'analyse

et de théorie analytique des nombres) et d'autres systèmes HOL n'ont pas de telles fonctions, ce qui signifie que certaines constructions en géométrie sont plus compliquées qu'en Coq ou Lean. Voir par exemple [BPL21], qui définit les schémas en Isabelle/HOL mais qui doit construire une nouvelle implémentation de la théorie des anneaux à partir de zéro pour pouvoir y parvenir.

3.4. Exemples. Voyons quelques exemples de ce à quoi ressemblent les mathématiques dans un assistant de preuves basé sur la théorie des types. Je donne ces exemples principalement pour convaincre le lecteur qui a appris à utiliser le langage de la théorie des ensembles qu'il y a vraiment très peu de différence.

Voici à quoi ressemble l'assertion que $\sqrt{2}$ n'est pas rationnel en Isabelle/HOL:

```
theorem sqrt2_not_rational:
     "sqrt 2 ∉ ℚ"
```

On peut voir la preuve en Isabelle dans l'article de wikipedia [Wik]. Le fait que 2 est un terme d'un type et non pas un ensemble, ou un élément d'un ensemble, est transparent.

Voici des mathématiques plus avancées, écrites en Coq :

```
Lemma prod_Cyclotomic n :
     (n > 0) % N → ∏d ← divisors n 'Phi_d = ' X^n - 1.
```

C'est l'énoncé que le produit des $d^{\text{ièmes}}$ polynômes cyclotomiques sur $d \mid n$ est $X^n - 1$. Notons l'hypothèse $n > 0$, une supposition typique qu'un humain omettrait ; les ordinateurs sont très pointilleux sur de tels "cas particuliers".

Voici la définition d'un anneau perfectoïde en Lean, pris sur le site Lean des espaces perfectoïdes [BCM] qui accompagne l'article [BCM20].

```
/-- Un anneau perfectoïde est un anneau de Huber qui est complet, uniforme,
qui a un pseudo-uniformisateur dont la puissance p-ième divise p dans le sous-anneau lié à la puissance,
et tel que le Frobenius est une surjection sur la réduction modulo p.-/
structure perfectoid_ring (R : Type) [Huber_ring R] extends
     Tate_ring R : Prop :=
(complete :  is_complete_hausdorff R)
(uniform :  is_uniform R)
(ramified :  ∃ ϖ :  pseudo_uniformizer R, ϖ^p | p ∈ R°)
(Frobenius :  surjective (Frob R°/p))
```

Le commentaire au début du code est la "documentation" pour le code – c'est l'explication lisible par un humain de ce que la définition Lean `perfectoid_ring` représente, et cette documentation est visible quand on fait passer le curseur sur le mot `perfectoid_ring` dans du code Lean ; si vous faites tourner le code dans un IDE comme Microsoft VS Code alors le clic droit sur le mot vous fera sauter vers sa définition.

La définition en Lean coïncide assez bien avec la définition humaine. Si R est un anneau de Huber qui est un anneau de Tate (ce sont des propriétés techniques des anneaux topologiques), alors on dit que R est un anneau perfectoïde s'il est complet, uniforme et s'il satisfait à un couple de propriétés techniques. Le point à observer est que le code informatique n'est ni plus ni moins difficile que la définition humaine.

3.5. Fondements. Dans mon expérience, les mathématiciens ont très peu d'intérêt pour les technicités des fondements logiques de leur sujet – ils ne peuvent pas faire la liste des axiomes de la théorie des ensembles, mais ils savent d'expérience ce que sont les "mathématiques légales". Les controverses du début du $20^{\text{ème}}$ siècle à propos du fait que les méthodes non constructives soient autorisées dans les preuves mathématiques sont mortes il y a bien longtemps ; les mathématiciens au travail utilisent la loi du tiers exclus partout, et beaucoup utilisent l'axiome du choix sous une forme ou une autre (en effet, le choix dépendant dénombrable peut être invoqué presque sans le remarquer). Un chercheur mathématicien typique sera allé au maximum à un cours sur les fondements des mathématiques ; dans un tel cours, on apprend typiquement que la théorie des ensembles de Zermelo–Frankel avec l'axiome du choix, ou ZFC, peut être utilisée comme fondement de

beaucoup de mathématiques. En effet, elle a pu être utilisée principalement par tous les mathématiciens jusqu'aux années 1960 ; pourtant les théories super-générales de la cohomologie de Grothendieck développées dans SGA4 ont introduit un nouvel "axiome des univers" (l'assertion que tout ensemble est un élément d'un ensemble qui est un modèle de ZFC). Cet axiome ne peut pas être prouvé à partir de axiomes de ZFC, par le théorème de Gödel. Les preuves originales des conjectures de Weil en théorie utilisaient cet axiome dans le sens faible à tel point qu'à un moment, la seule référence pour la cohomologie étale était SGA4. Pourtant Deligne et d'autres ont indiqué dans SGA4$\frac{1}{2}$ que la théorie de la cohomologie étale, et par conséquent la preuve des conjectures de Weil, peut être établie dans ZFC seule. Les lecteurs intéressés par les contorsions que l'on doit faire pour cela peuvent regarder la section Théorie des ensembles du projet Stacks, par exemple ici [Sta18, Tag 000H]. Pour un exemple plus extrême, voir la section 4 de [Sch17], où l'on voit un médaillé Fields forcer une théorie plus élaborée à entrer dans ZFC.

Mon opinion personnelle est que bien que ZFC soit un merveilleux fondement pour la plupart des mathématiques du début du 20$^{\text{ème}}$ siècle, le manque de l'axiome d'univers signifie maintenant que cela nécessite de plus en plus d'efforts d'obtenir que des parties des mathématiques modernes s'adaptent à ZFC. Dans les livres et les articles traitant de catégories infinies ou de mathématiques condensées, il est fréquent que se montrent des univers, et je me demande vraiment s'il n'est pas temps maintenant pour les mathématiciens d'embrasser les univers, comme Grothendieck nous encourage à le faire depuis les années 1960. La théorie des types de Coq et la théorie des types de Lean contiennent toutes deux les univers comme faisant partie de leurs fondements, même si les mathématiciens peuvent choisir de ne pas les utiliser si c'est ce qu'ils désirent.

4. Le futur

Dans cette section, je décris quelques unes des conséquences plausibles de la formalisation des mathématiques dans un solveur de théorèmes. Je souligne également quelques choses dont je crois qu'elles resteront hors d'atteinte pour quelques temps encore. Des observations plus étendues de Patrick Massot [Masb] valent également la peine d'être lues (en effet, plusieurs de mes idées ici ont été trouvées après des conversations avec Massot).

4.1. Une nouvelle sorte de document mathématique. À l'heure actuelle, les auteurs de livres ou d'articles de recherche doivent décider combien de matériau de base supposer, et quelles techniques regarder comme standards des arguments qu'ils présentent. En d'autres termes, ils doivent décider où commencer, et à quelle vitesse aller. Si un lecteur potentiel (par exemple, un nouvel étudiant en thèse, ou un étudiant universitaire intéressé par le domaine) n'a pas les prérequis alors il lui sera beaucoup plus difficile d'obtenir quoi que ce soit de la lecture de l'article.

La formalisation informatique offre la possibilité d'une nouvelle sorte de document mathématique, où le *lecteur* peut prendre les décisions concernant le niveau de détail auquel lire l'article. Patrick Massot a expérimenté de tels documents. Une version préliminaire de sa vision peut être vue dans les pages web de son projet concernant l'inversion de la sphère [Masa]. C'est un projet dont le but principal est de formaliser en Lean une preuve du théorème de Smale disant qu'une sphère peut être retournée (ou plus formellement, qu'il y a une homotopie des immersions entre l'immersion identité de S^2 dans $\mathbb{R}^3$ et l'immersion antipodale). Au moment d'écrire le présent article, la preuve n'est pas encore entièrement formalisée, mais c'est seulement une question de temps. Le blueprint est écrit en LaTeX, mais en utilisant plasTeX il a été converti en une page web contenant des liens Lean exécutables. Jusqu'ici, ces liens vous envoient sur des pages statiques contenant du code Lean, mais des outils sont en train d'être développés qui changeront cela. Alectryon est un programme utilisable pour Coq et Lean qui peut transformer un code compilé en pages web. Des outils comme Alectryon nous permettront d'écrire des documents qui contiendront des liens vers des pages web dynamiques montrant tous les détails mathématiques vers des images interactives, sous une forme humainement lisible, et qui permettront au lecteur de creuser profondément vers les axiomes, même si bien sûr il est peu vraisemblable que n'importe qui souhaite aller si profondément.

Il y a déjà des variantes de cette idée qui existent, l'idée de Lamport de "preuve structurée" est venue d'un désir d'encourager les mathématiciens à écrire beaucoup plus de détails dans leurs articles, mais l'on peut voir pourquoi une telle proposition ne descendrait pas bien. Ici on peut laisser l'automate faire le travail

pour nous. L'assistant de preuve Metamath offre aussi déjà de telles fonctionnalités, parce que Metamath a très peu d'automatisation et que par conséquent, forer jusqu'aux axiomes est essentiellement la même chose qu'inspecter la preuve.

On pourrait aussi imaginer des livres pour étudiants sans erreurs écrits de cette manière, où les énoncés qu'un étudiant ne peut comprendre (peut-être parce qu'ils sont ambigus) peuvent être inspectée avec plus de détails jusqu'à ce que les difficultés soient résolues.

4.2. Recherche sémantique dans une base de données mathématique. La chose qui va se produire prochainement, c'est une sorte de révolution où tous les mathématiciens commenceront à écrire leurs articles dans un assistant formel de preuves. Bien que l'on puisse attendre dans le futur que *quelques* articles soient partiellement, ou même complètement formalisés dans un assistant de preuves (voir par exemple [GS19], [SB19] et [KHNY21]), cette sorte d'approche ne deviendra pas la norme bientôt. Face à cette réalité, comment les mathématiques formalisées seront-elles capables de suivre les frontières des mathématiques ?

J'ai déjà mentionné l'exposé de Tom Hales en 2017 "*Grandes conjectures*" au Newton Institute à Cambridge. Dans l'exposé [Hal], Hales argumente pour une version formalisée de la revue Math Reviews/Zentralblatt. C'est-à-dire un site web dont le rôle est d'*établir* formellement les résultats annoncés dans les journaux principaux de mathématiques. Notons qu'un tel projet est loin d'être aussi farfelu que l'idée de formaliser les *preuves* mathématiques en temps réel ; les *énoncés* des théorèmes sont beaucoup plus faciles à formaliser.

Le problème dans le plan de Hales, comme il l'indique dans son exposé, c'est que pour être capable de formaliser des énoncés de théorèmes même dans une partie des mathématiques telle que la philosophie de Langlands, on devrait définir tous les objets de base que les mathématiciens de ce domaine utilisent. Dans la philosophie de Langlands, cela devrait inclure, mais cela ne devrait en aucun cas se limiter à, des définitions des formes automorphes et des représentations automorphes, des représentations de Galois, des variétés abéliennes, les anneaux définis par Fontaine et utilisés pour faire de la théorie de Hodge p-adique, des schémas, toutes les théories de la cohomologie utilisées dans ce domaine, les espaces perfectoïdes, les adèles, les idèles, ... La communauté Lean a ces quelques dernières années poussé fort pour faire entrer les principales définitions de la recherche mathématique moderne dans `mathlib`. À l'Imperial College seul, nous avons Oliver Nash qui développe les bases de la théorie des algèbres de Lie donc nous pouvons parler de centres d'algèbres universelles enveloppantes, María Inés de Frutos-Fernández qui développe la théorie des adèles et des idèles des corps globaux avec un œil sur les énoncés de la théorie du corps de classes, Amelia Livingston qui développe la cohomologie des groupes et de Galois, Jujian Zhang qui développe la cohomologie des faisceaux avec un œil sur GAGA, et Ashvni Narayanan qui développe les bases de la théorie d'Iwasawa dans sa thèse. J'ai déjà mentionné mon propre travail, avec Massot et Commelin pour définir les espaces perfectoïdes. Le travail de Scott Morrison, Bhavik Mehta, Justus Springer et Adam Topaz nous a récemment permis de commencer à développer la théorie des faisceaux sur des sites et l'algèbre homologique, ce qui fait que les théories de la cohomologie ne sont plus trop loin. Bien sûr, beaucoup reste à faire, mais nous espérons que l'idée d'être capable d'*énoncer* formellement les théorèmes des articles de théorie des nombres de la revue *Annals and Inventiones* en Lean deviendra bientôt une réalité.

Un projet relié consiste à formaliser les tags dans le projet Stacks. Le projet Stacks [Sta18] est une base de données géante de géométrie algébrique, accessible librement en ligne. Imprimée, elle représente plus de 7000 pages. Formaliser toutes les preuves dans la base de données serait une tâche extrêmement ardue impliquant de nombreuses décennies de travail de plusieurs personnes avec la technologie actuelle. En théorie, il est possible, bien qu'on ait besoin d'une équipe dont les membres seraient à la fois experts en géométrie algébrique et en formalisation. De plus, pour que cela arrive vraiment, la structure incitative des mathématiques académiques devrait radicalement changer. La publication d'articles dans des revues prestigieuses d'informatique expliquant comment vous développez la théorie de base des anneaux de Cohen–Macauley et les modules dans un solveur de théorèmes (et bien sûr qu'un tel travail serait publiable dans les proceedings d'une revue prestigieuse d'informatique – personne ne l'a jamais fait auparavant) n'est peut-être pas quelque chose qui est reconnu par les comités de promotions.

Pourtant il y a une solution qui nous est actuellement accessible. Formaliser juste les *définitions* et les *énoncés* des théorèmes dans le projet Stacks est une tâche *beaucoup* plus simple. N'importe quelle personne intéressée par la géométrie algébrique serait plus que bienvenue pour apprendre Lean et tenter de formaliser les énoncés des tags du projet Stacks. Faites pointer votre navigateur vers le forum de discussion Zulip [Zul] et demandez comment commencer dans le fil `#nouveaux membres`.

La raison pour laquelle la construction de telles bases de données est importante est que cela permettra à la communauté de construire des outils d'une sorte que les mathématiciens n'ont jamais vue précédemment. Imaginons que toutes les définitions et énoncés des théorèmes du projet Stacks aient été formalisés en Lean ou dans n'importe quel autre solveur de théorèmes. Un "marteau" est un outil qui tourne dans un solveur de théorème et qui peut tenter de construire les arguments mathématiques en assemblant par morceaux des résultats trouvés dans la base de données. Le marteau original était le *Sledgehammer* d'Isabelle/HOL [PB10]. L'intelligence derrière de tels outils est la capacité à isoler lesquels parmi les nombreux résultats de la base de données semblent les plus utiles, et de se concentrer sur ceux-ci en tentant de prouver le résultat souhaité. Maintenant considérons un étudiant de thèse qui commence à apprendre la géométrie algébrique. Un tel étudiant serait alors capable de poser une question au solveur de théorèmes, et le solveur tenterait d'utiliser la base de données pour répondre positivement (en assemblant les pièces d'une preuve) ou négativement (en produisant un contre-exemple, comme cela est fait sur le site web π-base [DC] pour des contre-exemples en topologie) à la question. La donnée résultante en sortie de l'ordinateur serait capable d'indiquer explicitement les références de la littérature, ou les preuves directes des assertions qu'il énonce dans son argumentation. Cette sorte d'outil – d'apprentissage assisté par ordinateur – a le potentiel de battre les techniques utilisées couramment par les étudiants thésards et effectuées à la main ("recherche sur google désespérément", "feuilletage d'un livre/article désespérément", "demander sur un site de maths et attendre", "demander à un autre humain"). Mais je l'ai souligné précédemment, la chose principale qui manque est la base de données de théorèmes, et c'est à nous de la construire. Plus tôt on l'aura, plus tôt les outils apparaîtront. Et plus grosse sera la base de données, plus puissants deviendront les outils.

4.3. Vérifier les démonstrations. Quelques informaticiens ont argumenté que les mathématiques sont négligentes, et que notre littérature contient des erreurs, et que ce problème peut être résolu avec des assistants informatiques de preuves. Un tel argument peut sembler plausible au premier abord, et je l'ai moi-même proposé il y a quelques années, mais il ne résiste pas à un examen approfondi. D'abord, les experts de notre communauté savent sur quels résultats il est possible de s'appuyer. Deuxièmement, de nombreuses erreurs ne sont pas des erreurs sérieuses et on peut les corriger. Troisièmement, les instances les plus sérieuses de ce problème ne peuvent être résolues par des assistants informatiques de preuves en ce moment en tout cas. Un exemple célèbre est la preuve que Mochizuki dit avoir trouvée de la conjecture ABC [Moc21]. Cette preuve a maintenant été publiée dans un journal de recherche sérieux, pourtant il est clair qu'elle n'est pas acceptée par la communauté mathématique en général. On pourrait défier Mochizuki, ou n'importe qui, de formaliser la preuve dans un assistant informatique de théorèmes. Pourtant ça serait une chose complètement déraisonnable à faire. Une formalisation informatique n'est pas attendue d'autres preuves apparaissant dans notre littérature. De plus, le point d'achoppement clef jusqu'à présent est que ceux qui ne croient pas en la preuve disent que plus de détails sont nécessaires dans la preuve du Corollaire 3.12 dans l'article principal, et l'état de l'art jusqu'à aujourd'hui est simplement que l'on ne peut commencer à formaliser ce corollaire sans avoir accès à ces détails sous une certaine forme (par exemple dans une preuve d'article papier contenant beaucoup plus d'informations à propos de l'argument).

Ce qui *serait* cependant faisable pour des mathématiciens serait de formaliser des *parties* du travail technique, ou d'obtenir que d'autres le fassent. Il pourrait y avoir plusieurs raisons pour faire une telle chose – Commelin et son équipe ont déjà montré que les solveurs de théorèmes peuvent être utilisés pour vérifier des parties de preuves complexes que des humains trouveraient difficiles de creuser, sans apprendre les mathématiques impliquées dans le processus.

4.4. Enseigner. J'ai entendu des étudiants dire "je pense que ma démonstration est OK" en parlant de leurs devoirs. Les assistants informatiques de preuves sont capables de le leur dire immédiatement si c'est le cas – à partir du moment où l'étudiant a pris la peine d'apprendre le langage de l'assistant de preuves. Devrions-nous apprendre aux étudiants de premier cycle comment utiliser des assistants informatiques de

preuves ? J'en suis certain. Patrick Massot à Orsay et moi-même à l'Imperial College de Londres enseignons tous deux des cours de premier cycle qui enseignent précisément cela.

Les étudiants veulent un retour sur leur travail dès que possible. Un assistant informatique de preuves peut fournir ce retour immédiatement.

Les étudiants débutants peuvent être confus à propos des bases. Quelle est la différence entre $\forall \epsilon > 0, \exists \delta > 0, \ldots$ et $\exists \delta > 0, \forall \epsilon > 0, \ldots$? Une fois que ces systèmes deviendront plus faciles à utiliser par les mathématiciens, les étudiants pourront expérimenter pour eux-mêmes des exemples bien trouvés fournis par un conférencier et commencer à comprendre de quoi il retourne. Un jour un étudiant m'a dit "Je ne comprenais pas les relations d'équivalence, je les ai alors formalisées en Lean, et alors je les ai comprises.". Forcer les étudiants à penser de façon pédante et logique peut être bon pour eux.

Ça vaut le coup d'insister sur le fait que demander à un étudiant faible d'à la fois suivre votre cours et simultanément d'apprendre à utiliser un assistant informatique de preuves est clairement trop demander à cet étudiant. Les solveurs doivent devenir plus faciles à utiliser, peut-être avec des interfaces graphiques et de la documentation plus approprié pour les mathématiciens. Demander aux gens de changer leur manière d'enseigner, c'est bien sûr leur demander beaucoup. Pourtant, les experts de l'enseignement des mathématiques seront assez contents de nous apprendre que notre medium préféré – "écrire pendant une heure sur un tableau" – est en train de devenir de moins en moins appropriée pour nos étudiants, qui aiment apprendre les choses en regardant une vidéo de 5 minutes ou en jouant à des jeux interactifs. Pouvons-nous rendre les mathématiques abstraites plus attractives ? Je suppose que nous le pouvons. Plus il y aura de personnes qui comprendront comment utiliser ces machines, plus tôt de nouvelles idées viendront.

4.5. Autres idées. Je ne prétends pas avoir étudié toutes les possibilités ici. Les personnes qui ont conçu le CD dans les années 1980 n'auraient sûrement pas pu envisager des services musicaux comme YouTube et Spotify, ou l'audio-livre. Les personnes qui ont commencé à penser à la manière de faciliter la visualisation de livres sur écrans d'ordinateurs n'avaient sûrement pas envisagé des services tels que les Kindle. Il est temps de regarder au-delà de la manière dont nous enseignons et apprenons les mathématiques, et d'essayer de comprendre comment nous, en tant que communauté de mathématiciens, pouvons utiliser l'inévitable digitalisation du matériau mathématique comme un outil pour rendre nos vies, et celles de nos étudiants, meilleures. Comme l'a dit un jour Carneiro, vous ne pouvez pas arrêter le progrès.

Références bibliographiques

[BCM] Kevin Buzzard, Johan Commelin, and Patrick Massot, *Lean perfectoid spaces*, https://leanprover-community.github.io/lean-perfectoid-spaces/, Accessed: 30-11-2021.

[BCM20] ———, *Formalising perfectoid spaces*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020 (Jasmin Blanchette and Catalin Hritcu, eds.), ACM, 2020, pp. 299–312.

[BCMS21] Sophie Bernard, Cyril Cohen, Assia Mahboubi, and Pierre-Yves Strub, *Unsolvability of the quintic formalized in dependent type theory*, 12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference) (Liron Cohen and Cezary Kaliszyk, eds.), LIPIcs, vol. 193, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 8:1–8:18.

[BDNdC21] Anne Baanen, Sander R. Dahmen, Ashvni Narayanan, and Filippo A. E. Nuccio Mortarino Majno di Capriglio, *A formalization of Dedekind domains and class groups of global fields*, 12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference) (Liron Cohen and Cezary Kaliszyk, eds.), LIPIcs, vol. 193, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 5:1–5:19.

[BG94] Helmut Bender and George Glauberman, *Local analysis for the odd order theorem*, London Mathematical Society Lecture Note Series, vol. 188, Cambridge University Press, Cambridge, 1994, With the assistance of Walter Carlip. MR 1311244

[BHL+21] Kevin Buzzard, Chris Hughes, Kenny Lau, Amelia Livingston, Ramon Fernández Mir, and Scott Morrison, *Schemes in Lean*, 2021, Exp. Math. published online 2021.

[BL21] Thomas Browning and Patrick Lutz, *Formalizing Galois theory*, 2021, Exp. Math. published online 2021.

[BPL21] Anthony Bordg, Lawrence Paulson, and Wenda Li, *Grothendieck's schemes in algebraic geometry*, March 2021, https://isa-afp.org/entries/Grothendieck_Schemes.html, Formal proof development.

[BRT18] Yves Bertot, Laurence Rideau, and Laurent Théry, *Distant decimals of π : Formal proofs of some algorithms computing them and guarantees of exact computation*, J. Autom. Reason. **61** (2018), no. 1-4, 33–71.

[BSD65] B. J. Birch and H. P. F. Swinnerton-Dyer, *Notes on elliptic curves. II*, J. Reine Angew. Math. **218** (1965), 79–108. MR 179168

[Buza] Kevin Buzzard, *Formalising mathematics: a first course for mathematicians.*, https://github.com/ImperialCollegeLondon/formalising-mathematics, Accessed: 30-11-2021.

[Buzb] ———, *Induction on equality*, https://xenaproject.wordpress.com/2021/04/18/induction-on-equality/, Accessed: 30-11-2021.

[Car] Mario Carneiro, *The type theory of Lean*, https://github.com/digama0/lean-type-theory/releases/download/v1.0/main.pdf, Accessed: 30-11-2021.

[Car21] Mario Carneiro, *Metamath Zero*, https://github.com/digama0/mm0, 2021.

[Cas] Davide Castelvecchi, *Mathematicians welcome computer-assisted proof in 'grand unification' theory*, https://www.nature.com/articles/d41586-021-01627-2, Accessed: 30-11-2021.

[CL21] Johan Commelin and Robert Y. Lewis, *Formalizing the ring of Witt vectors*, CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021 (Catalin Hritcu and Andrei Popescu, eds.), ACM, 2021, pp. 264–277.

[CM] Johan Commelin and Patrick Massot, *Blueprint for the Liquid Tensor Experiment*, https://leanprover-community.github.io/liquid/, Accessed: 30-11-2021.

[CMSPT14] Frédéric Chyzak, Assia Mahboubi, Thomas Sibut-Pinote, and Enrico Tassi, *A computer-algebra-based formal proof of the irrationality of ζ (3)*, International Conference on Interactive Theorem Proving, Springer, 2014, pp. 160–176.

[CP88] Thierry Coquand and Christine Paulin, *Inductively defined types*, COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988, Proceedings (Per Martin-Löf and Grigori Mints, eds.), Lecture Notes in Computer Science, vol. 417, Springer, 1988, pp. 50–66.

[DC] James Dabbs and Steven Clontz, *π-base*, https://topology.pi-base.org/, Accessed: 30-11-2021.

[DHL19] Sander R. Dahmen, Johannes Hölzl, and Robert Y. Lewis, *Formalizing the solution to the cap set problem*, 10th International Conference on Interactive Theorem Proving, LIPIcs. Leibniz Int. Proc. Inform., vol. 141, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2019, pp. Art. No. 15, 19. MR 4008934

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer, *The Lean theorem prover (system description)*, Automated Deduction - CADE-25 (Cham) (Amy P. Felty and Aart Middeldorp, eds.), Springer International Publishing, 2015, pp. 378–388.

[Ebe19a] Manuel Eberl, *The irrationality of $\zeta(3)$*, https://www.isa-afp.org/entries/Zeta_3_Irrational.html, December 2019, Accessed: 30-11-2021.

[Ebe19b] ———, *Nine chapters of analytic number theory in Isabelle/HOL*, 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA (John Harrison, John O'Leary, and Andrew Tolmach, eds.), LIPIcs, vol. 141, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 16:1–16:19.

[EG17] Jordan S. Ellenberg and Dion Gijswijt, *On large subsets of $\mathbb{F}_q^n$ with no three-term arithmetic progression*, Ann. of Math. (2) **185** (2017), no. 1, 339–343. MR 3583358

[EKAP21] Chelsea Edmonds, Angeliki Koutsoukou-Argyraki, and Lawrence C. Paulson, *Roth's theorem on arithmetic progressions*, https://www.isa-afp.org/entries/Roth_Arithmetic_Progressions.html, December 2021, Accessed: 30-01-2022.

[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*, Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings (Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, eds.), Lecture Notes in Computer Science, vol. 7998, Springer, 2013, pp. 163–179.

[GM05] Benjamin Grégoire and Assia Mahboubi, *Proving equalities in a commutative ring done right in Coq*, Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings (Joe Hurd and Thomas F. Melham, eds.), Lecture Notes in Computer Science, vol. 3603, Springer, 2005, pp. 98–113.

[Gon07] Georges Gonthier, *The four colour theorem: Engineering of a formal proof*, Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers (Deepak Kapur, ed.), Lecture Notes in Computer Science, vol. 5081, Springer, 2007, p. 333.

[Gon08] Georges Gonthier, *Formal proof—the four-color theorem*, Notices Amer. Math. Soc. **55** (2008), no. 11, 1382–1393. MR 2463991

[GS19] Sébastien Gouëzel and Vladimir Shchur, *Corrigendum: A corrected quantitative version of the Morse lemma [MR3003738]*, J. Funct. Anal. **277** (2019), no. 4, 1258–1268. MR 3959731

[HAB+17] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller, *A formal proof of the Kepler conjecture*, Forum Math. Pi **5** (2017), e2, 29. MR 3659768

[Hal] Tom Hales, *Big conjectures*, https://www.newton.ac.uk/seminar/21474/, Accessed: 30-11-2021.

[Hal14] Thomas C. Hales, *Mathematics in the age of the Turing machine*, Turing's legacy: developments from Turing's ideas in logic, Lect. Notes Log., vol. 42, Assoc. Symbol. Logic, La Jolla, CA, 2014, pp. 253–298. MR 3497663

[Har09a] John Harrison, *Formalizing an analytic proof of the prime number theorem*, J. Automat. Reason. **43** (2009), no. 3, 243–261. MR 2544285

[Har09b] John Harrison, *HOL light: An overview*, Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings (Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, eds.), Lecture Notes in Computer Science, vol. 5674, Springer, 2009, pp. 60–66.

[HvD20] Jesse Michael Han and Floris van Doorn, *A formal proof of the independence of the continuum hypothesis*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020 (Jasmin Blanchette and Catalin Hritcu, eds.), ACM, 2020, pp. 353–366.

[Imm18] Fabian Immler, *A verified ODE solver and the Lorenz attractor*, Journal of automated reasoning **61** (2018), no. 1, 73–111.

[IT20] Fabian Immler and Yong Kiam Tan, *The poincaré-bendixson theorem in isabelle/hol*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (New York, NY, USA), CPP 2020, Association for Computing Machinery, 2020, p. 338–352.

[KHNY21] Bjørn Kjos-Hanssen, Saroj Niraula, and Soowhan Yoon, *A parametrized family of Tversky metrics connecting the Jaccard distance to an analogue of the normalized information distance*, https://arxiv.org/abs/2111.02498, 2021.

[Lew19] Robert Y. Lewis, *A formal proof of Hensel's lemma over the p-adic integers*, Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (New York, NY, USA), CPP 2019, Association for Computing Machinery, 2019, p. 15–26.

[LM20] Robert Y. Lewis and Paul-Nicolas Madelaine, *Simplifying casts and coercions (extended abstract)*, Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual) (Pascal Fontaine, Konstantin Korovin, Ilias S. Kotsireas, Philipp Rümmer, and Sophie Tourret, eds.), CEUR Workshop Proceedings, vol. 2752, CEUR-WS.org, 2020, pp. 53–62.

[Masa] Patrick Massot, *The sphere eversion project*, https://leanprover-community.github.io/sphere-eversion/blueprint/index.html, Accessed: 30-11-2021.

[Masb] ———, *Why formalize mathematics?*, https://www.imo.universite-paris-saclay.fr/~pmassot/files/exposition/why_formalize.pdf, Accessed: 11-12-2021.

[mathlibc20] The `mathlib` community, *The Lean mathematical library*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (New York, NY, USA), CPP 2020, Association for Computing Machinery, 2020, p. 367–381.

[Moc21] Shinichi Mochizuki, *Inter-universal Teichmüller theory III: Canonical splittings of the log-theta-lattice*, Publ. Res. Inst. Math. Sci. **57** (2021), no. 1, 403–626. MR 4225475

[MW19] Norman D. Megill and David A. Wheeler, *Metamath: A computer language for mathematical proofs*, Lulu Press, Morrisville, North Carolina, 2019, http://us.metamath.org/downloads/metamath.pdf.

[NK09] Adam Naumowicz and Artur Korniłowicz, *A brief overview of mizar*, Theorem Proving in Higher Order Logics (Berlin, Heidelberg) (Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, eds.), Springer Berlin Heidelberg, 2009, pp. 67–72.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel, *Isabelle/hol - A proof assistant for higher-order logic*, Lecture Notes in Computer Science, vol. 2283, Springer, 2002.

[PB10] Lawrence C. Paulson and Jasmin Christian Blanchette, *Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers*, The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011 (Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, eds.), EPiC Series in Computing, vol. 2, EasyChair, 2010, pp. 1–11.

[pca] The Lean prover community, *The Lean community website.*, https://leanprover-community.github.io/index.html, Accessed: 30-11-2021.

[pcb] ———, *A mathlib overview*, https://leanprover-community.github.io/mathlib-overview.html, Accessed: 30-11-2021.

[pcc] ———, *Undergraduate mathematics in mathlib*, https://leanprover-community.github.io/undergrad.html, Accessed: 30-11-2021.

[Pet00] Thomas Peterfalvi, *Character theory for the odd order theorem*, London Mathematical Society Lecture Note Series, vol. 272, Cambridge University Press, Cambridge, 2000, Translated from the 1986 French original by Robert Sandling and revised by the author. MR 1747393

[SB19] Neil Strickland and Nicola Bellumat, *Iterated chromatic localisation*, https://arxiv.org/abs/1907.07801, 2019.

[Scha] Peter Scholze, *Half a year of the Liquid Tensor Experiment: Amazing developments*, https://xenaproject.wordpress.com/2021/06/05/half-a-year-of-the-liquid-tensor-experiment-amazing-developments/, Accessed: 30-11-2021.

[Schb] Peter Scholze, *Lectures on analytic geometry*, https://www.math.uni-bonn.de/people/scholze/Analytic.pdf, Accessed: 30-11-2021.

[Schc] Peter Scholze, *Liquid tensor experiment*, https://xenaproject.wordpress.com/2020/12/05/liquid-tensor-experiment/, Accessed: 30-11-2021.

[Sch17] ———, *Etale cohomology of diamonds*, https://arxiv.org/abs/1709.07343, 2017.

[Sch21] ———, *Liquid tensor experiment*, 2021, Exp. Math. Published online, 2021.

[Sta18] The Stacks Project Authors, *Stacks Project*, https://stacks.math.columbia.edu, 2018.

[Tea21] Coq Development Team, *The coq proof assistant*, http://coq.inria.fr, 1989-2021, Accessed: 11-12-2021.

[Wer97] Benjamin Werner, *Sets in types, types in sets*, Theoretical aspects of computer software (Sendai, 1997), Lecture Notes in Comput. Sci., vol. 1281, Springer, Berlin, 1997, pp. 530–546. MR 1608927

[Wie21] Eric Wieser, *Scalar actions in lean's mathlib*, 2021.
[Wik] Wikipedia, *Isabelle (proof assistant)*, https://en.wikipedia.org/wiki/Isabelle_(proof_assistant), Accessed: 30-11-2021.
[Zul] Zulip, *The Lean community Zulip chatroom*, https://leanprover.zulipchat.com, Accessed: every day since 2018.