Université Paris 1 Panthéon-Sorbonne
M1 — Logique et fondements de l'informatique — LoPhiSC
Pablo Donato

Vérification formelle à l'ère de l'IA

Promesses, retournements, questions ouvertes

Séance 6 — Notes de cours

Introduction

Au terme de six semaines, un chemin a été parcouru qui peut sembler, rétrospectivement, d'une grande abstraction : du λ-calcul aux types inductifs, en passant par la correspondance de Curry-Howard et la déduction naturelle, le cours a introduit les outils techniques d'une discipline, la *vérification formelle*, dont la portée effective dans les pratiques scientifiques et technologiques n'a pas encore été prise en vue. Ces notes proposent une telle prise de vue. Elles ne présentent aucun nouveau concept technique ; elles cherchent plutôt à situer ce qui a été appris dans un paysage intellectuel plus large, en articulant trois moments.

Le premier moment — *la promesse* — reconstruit l'argument, aujourd'hui défendu par un nombre croissant de chercheurs, selon lequel la conjonction de la vérification formelle et des grands modèles de langage ouvre une ère nouvelle : d'abord pour les mathématiques, ensuite pour la sûreté des systèmes critiques, jusqu'à comprendre, à la limite, la sûreté de l'intelligence artificielle elle-même. Le deuxième moment — *le retournement* — confronte cette promesse aux événements de l'année 2026, qui ont fait apparaître, à trois niveaux distincts de la chaîne de confiance, des fractures jusque-là restées théoriques. Le troisième moment — *les questions ouvertes* — ne conclut pas ; il propose quatre lignes d'interrogation que la philosophie est particulièrement bien équipée pour poursuivre.

1 La promesse

1.1 Du rêve de Leibniz à la vérification formelle

Au XVII^{e} siècle, Leibniz formule le double projet d'une *characteristica universalis*, langue universelle dans laquelle tout raisonnement s'exprimerait sans ambiguïté, et d'un *calculus ratiocinator*, mécanisme de calcul qui trancherait les désaccords intellectuels comme on résout une équation. *Calculemus*, proposait-il : calculons plutôt que de disputer.

Ce projet est demeuré longtemps spéculatif. Il n'a trouvé d'incarnation partielle qu'à la fin du XX^{e} siècle et au début du XXI^{e}, sous la forme de ce que l'on appelle aujourd'hui la *vérification formelle*. Son principe tient en trois étapes : on écrit une spécification mathématique de ce qu'un système doit faire ; on écrit le système ; on produit, dans un assistant de preuve, une démonstration mécaniquement vérifiée que le système satisfait la spécification.

Plusieurs réalisations de premier plan attestent la fécondité de cette approche. Le micronoyau de système d'exploitation *seL4* a été prouvé correct en Isabelle/HOL : on démontre mathématiquement qu'il est exempt de toute une classe de bugs — pas de débordement de tampon, pas de fuite de mémoire, pas de violation d'invariant (Klein et al. 2009). Le compilateur *CompCert*, développé par Xavier Leroy et son équipe en Coq, produit du code machine dont on a prouvé

qu'il est sémantiquement équivalent à son code source, excluant ainsi la possibilité qu'un bug soit introduit au cours de la compilation (Leroy 2009). Dans un registre proprement mathématique, le théorème des *quatre couleurs* a été entièrement formalisé en Coq par Gonthier (Gonthier 2008), et la *conjecture de Kepler* sur l'empilement optimal des sphères a été formalisée dans le projet Flyspeck (Hales et al. 2017).

Ces réalisations représentent des *garanties mathématiques* là où l'ingénierie classique n'offre que des garanties statistiques fondées sur le test. Le saut qualitatif est réel. Il faut toutefois lui apporter immédiatement un correctif de taille : ces réalisations coûtent extrêmement cher. La preuve de seL4 a mobilisé une vingtaine de années-personnes ; celle de CompCert, environ six ; la formalisation d'un théorème mathématique important est couramment estimée à un facteur dix à vingt du coût de sa rédaction informelle. La vérification formelle, telle qu'elle a été pratiquée jusqu'ici, est freinée par un *goulot d'étranglement humain* : l'écriture des preuves exige une expertise rare et un temps considérable.

La question qui structure tout le premier acte est dès lors celle-ci : et si l'intelligence artificielle venait lever ce goulot ?

1.2 La nouvelle division du travail

Pour saisir la mutation en cours, il convient d'expliciter la division du travail qui prévaut dans le paradigme classique. D'un côté, l'humain assume deux tâches : il rédige la spécification, c'est-à-dire l'énoncé formel de ce qu'il souhaite prouver, et il construit la preuve elle-même pas à pas en combinant des *tactiques*. De l'autre, le noyau de l'assistant de preuve — un programme de quelques milliers de lignes — vérifie mécaniquement qu'un λ-terme est bien typé. Le noyau ne comprend rien au contenu mathématique ; il applique les règles du système de types. Si le terme type, la preuve est acceptée ; sinon, elle est rejetée.

Ce que l'intelligence artificielle transforme, dans ce schéma, n'est pas le noyau mais le rôle de l'humain. Le paradigme émergent peut se décrire ainsi : l'humain se limite à énoncer la spécification ; un modèle de langage (LLM) génère des tentatives de preuve, sous forme de tactiques ou directement sous forme de termes ; ces tentatives sont soumises au noyau, qui les vérifie et, en cas d'échec, renvoie au modèle un message d'erreur exploité pour une nouvelle tentative.

La propriété remarquable de ce dispositif, sur laquelle il faut insister, est la suivante : la qualité du modèle de langage n'affecte pas la correction du résultat final. Le modèle peut halluciner, inventer des lemmes inexistants, produire des raisonnements spécieux ; tout terme qui ne type pas sera rejeté par le noyau. Une *asymétrie* fondamentale se met en place entre la recherche de la preuve (confiée à une source faillible) et la validation de la preuve (assurée par un composant vérifié). C'est cette asymétrie qui rend l'articulation entre IA et vérification formelle si prometteuse : contrairement à la plupart des applications de l'IA, il n'est pas nécessaire de faire confiance au modèle ; on le sollicite pour effectuer une tâche de recherche coûteuse, tout en gardant le dernier mot sur la validité du résultat.

Cette asymétrie peut être résumée, sur le mode du proverbe, par la formule : *l'IA propose, le noyau dispose.*

1.3 Les mathématiques : l'appétit de l'IA

La mutation est déjà visible dans la pratique mathématique. Le philosophe et logicien Jeremy Avigad parle à ce sujet d'un *tournant formel* des mathématiques (Avigad 2023). Le mathématicien Kevin Buzzard, à Imperial College, formule cela d'une manière frappante : si les ordinateurs savent *calculer* depuis les années 1960, ils commencent désormais à savoir *raisonner* (Buzzard 2025 ; *Professor Kevin Buzzard* 2025).

Plusieurs jalons témoignent de ce tournant. Le *Liquid Tensor Experiment*, initié par Peter

Scholze, a consisté à formaliser en Lean un résultat de ses propres travaux dont il doutait ; la formalisation a effectivement mis au jour un détail subtil de la démonstration originale (AL. 2021). La bibliothèque *Mathlib* contient désormais une proportion croissante du contenu d'un cursus universitaire. En 2024, le système *AlphaProof* de DeepMind a atteint un niveau de médaille d'argent à l'Olympiade internationale de mathématiques, en formalisant automatiquement ses solutions en Lean (DEEPMIND 2024).

Un seuil qualitatif a toutefois été franchi au début de l'année 2026. La base collaborative *erdosproblems*, qui recense des problèmes ouverts posés par Paul Erdős, est devenue le terrain où des systèmes d'IA ont commencé à produire, de manière quasi autonome, de véritables résolutions (NATSOTHANAPHAN 2026). En janvier 2026, les *Problems #397, #728* et *#729* sont résolus en une semaine par une combinaison entre GPT-5.2 Pro (OpenAI) et *Aristotle* (Harmonic), ce dernier prenant en charge l'*autoformalisation* en Lean de la preuve produite par le modèle de langage (ARISTOTLE TEAM (HARMONIC) AND NEEL SOMANI 2026). En février 2026, l'agent *Aletheia* de Google DeepMind, appuyé sur Gemini 3 Deep Think, résout à son tour, de manière entièrement autonome, quatre autres problèmes ouverts, dont le *Problem #1051* sur l'irrationalité de séries rapidement convergentes (GOOGLE DEEPMIND 2026). Plus largement, une centaine d'énoncés ont basculé dans la colonne des « résolus » depuis octobre 2025.

Ce seuil est remarquable à deux titres. Premièrement, il marque un passage des *mathématiques de compétition* (où les énoncés sont connus pour admettre une solution accessible à un bon étudiant, comme à l'IMO) aux *mathématiques de recherche* (où les énoncés sont des questions ouvertes, souvent depuis des décennies). Deuxièmement, il réalise concrètement la division du travail décrite plus haut : la recherche de la preuve est confiée à un LLM faillible, l'autoformalisation à un système spécialisé, et la validation à un noyau vérifié. La confiance accordée au résultat ne dépend en rien de la correction des deux premiers étages ; elle ne dépend que du troisième.

Un argument récent de Simone Severini, distinguished engineer chez Google, éclaire particulièrement la dynamique à l'œuvre (SEVERINI 2026). Selon lui, l'état actuel est que moins de 0,05% des mathématiques publiées sont formalisées, ce qui représente une proportion négligeable. Mais les IA contemporaines manifestent un *appétit* pour les corpus vérifiés, qui leur permettent de s'entraîner sur des données dont la correction est garantie. La formalisation, qui relevait jusqu'ici d'un intérêt essentiellement scientifique, devient ainsi une *application phare* de portée économique et infrastructurelle : les grands laboratoires d'IA ont un intérêt direct à la financer. Philosophiquement, cela marque une inversion du moteur : la formalisation n'est plus motivée seulement par la quête de rigueur mathématique, mais aussi par les besoins d'entraînement des systèmes d'apprentissage.

Il importe toutefois de nuancer ce tableau. Comme le souligne Patrick Massot, l'un des contributeurs les plus actifs de Mathlib, la formalisation est pour les mathématiciens un *outil*, non un *substitut* (MASSOT 2021). Aucune position sérieuse, parmi les praticiens, ne soutient que la pratique mathématique serait remplacée à moyen terme par ses versions formalisées. Une pluralité de styles — preuve informelle, preuve semi-formelle, preuve intégralement mécanisée — coexiste et s'enrichit mutuellement.

1.4 Au-delà des mathématiques : la sûreté de l'IA par la preuve

L'argument peut être poussé au-delà du domaine mathématique. Max Tegmark et Steve Omohundro défendent dans leur article *Provably Safe Systems* la thèse selon laquelle la voie la plus crédible vers une intelligence artificielle puissante et contrôlable passe par la preuve formelle (TEGMARK et OMOHUNDRO 2023).

Leur raisonnement peut se reconstruire en trois temps. *Premièrement*, les techniques actuelles de contrôle des IA, telless que l'apprentissage par renforcement à partir de retours humains (RLHF) et les dispositifs de surveillance en temps réel, produisent des garanties de nature *statistique* : on peut constater que le système se comporte de manière satisfaisante dans une

grande proportion des cas testés, mais non dans tous les cas possibles. *Deuxièmement*, lorsque le système est déployé à grande échelle et opère dans des domaines à fort enjeu — infrastructures critiques, systèmes financiers, à la limite pilotage de la prochaine génération d'IA, ces quelques cas sur mille deviennent inacceptables. Seule une garantie de nature *mathématique*, obtenue par preuve formelle, peut fournir l'assurance requise. *Troisièmement*, c'est précisément l'arrivée de l'IA qui rend cette approche *faisable* à grande échelle : parce que c'est désormais elle qui peut rédiger les preuves, on peut envisager de vérifier formellement des volumes de code auparavant inaccessibles.

Au sommet de cet arc argumentatif se tient ce qu'on pourrait appeler la *promesse ultime* : la conjonction de la vérification formelle et de l'intelligence artificielle deviendrait la réponse canonique au problème de la sûreté des systèmes critiques, y compris des systèmes d'IA eux-mêmes.

Une objection doit cependant être introduite d'emblée, car elle orientera toute la suite du propos. Il est une chose de prouver formellement qu'un programme ne dépasse pas la pile mémoire allouée, ou qu'un protocole cryptographique préserve la confidentialité ; c'en est une autre, beaucoup plus difficile, de formaliser des notions telles que « ne pas nuire à un humain » ou « être honnête ». Le problème de la *spécification* — la détermination même de ce qu'il s'agit de prouver — reste entier, et il constitue peut-être le problème le plus profond soulevé par l'approche Tegmark-Omohundro.

2 Le retournement

2.1 Qui vérifie le noyau ?

Si la sûreté est fondée sur la preuve, il convient d'examiner la chaîne de confiance dans toutes ses dépendances. Le premier maillon est le vérificateur lui-même.

La vérification formelle moderne repose sur un principe architectural énoncé par le mathématicien néerlandais Nicolaas Govert de Bruijn dans le cadre du projet *AUTOMATH* (Bruijn 1980) : l'assistant de preuve doit être structuré autour d'un *noyau minimal*, dont la seule fonction est de vérifier que les λ-termes sont bien typés, et dont la taille doit être suffisamment réduite pour qu'un humain puisse en auditer intégralement le code. Ce principe, connu sous le nom de *critère de de Bruijn*, a plusieurs conséquences importantes. Il implique, notamment, que l'on *ne fait pas confiance aux tactiques* : celles-ci peuvent être aussi complexes et aussi buguées que l'on voudra, puisqu'elles ne produisent que des termes que le noyau vérifiera. La confiance porte exclusivement sur le noyau — lequel, dans Rocq, compte quelques milliers de lignes de code OCaml. Si le noyau est correct, toute preuve qu'il accepte est correcte.

Or cette dernière formulation laisse entière la question de la correction du noyau lui-même. On peut certes imaginer prouver le noyau dans un autre assistant de preuve, mais la question se déplace alors au noyau de ce dernier. Et même à supposer le noyau correct, il reste à le compiler ; le compilateur est-il lui-même correct ? Et le matériel qui exécute le binaire ? On voit apparaître une *régression* potentiellement infinie. C'est précisément cette régression que Ken Thompson thématise en 1984.

2.2 Thompson 1984 : *Reflections on Trusting Trust*

Ken Thompson, co-inventeur d'Unix et lauréat du prix Turing, consacre son discours de réception à un argument devenu classique, publié sous le titre *Reflections on Trusting Trust* (Thompson 1984).

L'expérience de pensée est la suivante. On considère un compilateur C malveillant possédant deux propriétés. *Premièrement*, lorsqu'il compile le programme `login` d'Unix (chargé de vérifier les mots de passe), il injecte silencieusement une porte dérobée : un mot de passe maître permet

d'accéder à tout compte. *Deuxièmement*, lorsqu'il compile un autre compilateur C, il injecte dans le binaire résultant ces deux mêmes comportements : il s'*auto-reproduit* à la compilation.

Supposons maintenant qu'un utilisateur, soucieux de s'assurer de l'innocuité de son compilateur, en examine le code source. Le code est propre : aucune trace de porte dérobée. Il recompile le compilateur à partir de ce code source, pour plus de sûreté. Mais le compilateur qui effectue cette recompilation est déjà contaminé : il réinjecte les comportements malveillants dans le nouveau binaire. Le code source est parfaitement propre ; le binaire est parfaitement compromis. La vérification visuelle du source n'y peut rien.

La thèse de Thompson se résume dans la formule : *« You can't trust code that you did not totally create yourself. »* — on ne peut faire confiance à du code que l'on n'a pas entièrement créé soi-même. L'argument n'est pas un argument contre un bug particulier ; il vise une structure générale de la confiance en informatique. Toute vérification s'exécute sur une machine, et cette machine n'est pas elle-même vérifiée. La vérification n'élimine pas la confiance : elle la *déplace*. C'est ce qui explique l'importance que l'ingénierie de la sécurité accorde à la notion de *trusted computing base*, c'est-à-dire la « base de confiance » minimale à laquelle il faut, en dernière instance, s'en remettre. Le critère de de Bruijn trouve là sa justification la plus profonde : minimiser le noyau, c'est minimiser la surface de cette base de confiance. Mais cette surface n'est jamais nulle.

Jusqu'à une date récente, l'argument de Thompson demeurait en grande partie théorique. L'année 2026 lui a fourni plusieurs incarnations simultanées, dont trois méritent un examen.

2.3 Le retournement 2026 : trois fractures simultanées

Mythos (avril 2026). La société Anthropic publie un rapport intitulé *Claude Mythos Preview* (Anthropic 2026), dans lequel sont exposés les résultats d'un système d'IA spécialisé dans la recherche de vulnérabilités de sécurité. En quelques mois, le système a identifié des vulnérabilités inconnues (*0-days*) dans les noyaux OpenBSD, Linux et FreeBSD, avec des chaînes d'exploitation permettant l'exécution de code à distance. L'ordre de grandeur est celui du millier. Le système est en outre capable de reconstruire du code source à partir de binaires fermés et d'y identifier des failles dont les sources ne sont pas publiquement disponibles. L'échelle de découverte excède la capacité de relecture humaine des équipes de sécurité des projets concernés.

La portée philosophique de *Mythos* est de retourner l'argument de Tegmark et Omohundro. La capacité qui permet à l'IA de vérifier à grande échelle est aussi celle qui lui permet de casser à grande échelle. L'IA qui vérifie coexiste nécessairement avec l'IA qui exploite, et il n'est pas établi *a priori* que la première l'emportera sur la seconde : la dynamique est celle d'une course, non d'une victoire acquise.

`False` dans Rocq (mars 2026). Tristan Stérin, en collaboration avec d'autres chercheurs humains et en mobilisant le modèle Claude Opus 4.6, publie un billet intitulé *In Search of Falsehood* (Stérin 2026 ; Rocq Zulip community 2026). Leur résultat est qu'ils exhibent sept preuves indépendantes de `False`, acceptées par le noyau de Rocq. Les bugs exploités concernent principalement le *guard checker* (vérificateur de terminaison des fonctions récursives) et la gestion des *modules*. Le travail est issu d'une collaboration mixte IA/humains, menée en concertation avec l'équipe de développement de Rocq.

La portée de ce résultat est considérable. Le critère de de Bruijn garantit que le noyau est *auditable* ; il ne garantit pas qu'il est *correct*. L'auditabilité est une condition *nécessaire* de la correction, non une condition *suffisante*. Le noyau que l'enseignement présente habituellement comme une « base de confiance » infaillible possède en réalité plusieurs trous, dont l'exploration systématique devient, avec l'aide des LLM, particulièrement efficace.

***Watchers* (avril 2026).** Kiran Gopinathan publie un article décrivant un exploit dans la bibliothèque `lean-zip` (Gopinathan 2026). Cette bibliothèque implémente en Lean 4 un *décodeur*

ZIP prouvé correct : la fonction qui décompresse les données est formellement vérifiée. Toutefois, le *parseur d'archive*, c'est-à-dire le code qui lit la structure de fichier englobante, n'est pas vérifié. Il contient une vulnérabilité, exploitable à partir d'un fichier ZIP de 156 octets, qui provoque un débordement de tas dans le *runtime* Lean (dans la fonction `lean_alloc_sarray`), et aboutit à une exécution de code arbitraire. Un examen plus approfondi révèle en outre qu'un second défaut, distinct du précédent, n'était tout simplement pas *couvert* par la spécification du décodeur : le comportement exploité n'y était pas contraint, en sorte que la preuve, bien que correcte au regard des propriétés énoncées, ne pouvait en principe le prévenir. Fait notable pour la dialectique qui nous occupe, l'ensemble de ces défauts a été découvert de manière quasi autonome par *Claude Code*, l'agent de programmation d'Anthropic : le rôle humain s'est largement limité à la mise en place du protocole et à la rédaction finale du rapport.

Cet exemple est philosophiquement le plus riche des trois. Il réalise, littéralement, l'argument de Thompson vu précédemment : la preuve est correcte, le théorème est vrai, mais le *runtime* qui exécute la fonction prouvée n'est pas vérifié, et un défaut de ce runtime suffit à invalider toutes les garanties tirées de la preuve. La confiance, déplacée vers le runtime, s'y trouve trahie. À cela s'ajoute une limite plus fondamentale encore, touchant non plus l'exécution mais la formulation. Une preuve ne peut écarter que ce qu'elle a explicitement énoncé ; elle est impuissante face aux comportements que la spécification a laissés libres, sciemment ou par omission. La difficulté se déplace alors d'un cran : il ne suffit pas de prouver ce que l'on a spécifié, encore faut-il spécifier *exhaustivement* ce que l'on veut. Or cette exhaustivité, qui suppose d'anticiper tous les scénarios d'usage et tous les angles d'attaque, se heurte aux limites ordinaires de la pensée humaine. La formalisation n'exempte pas de cet effort, elle le rend au contraire décisif.

Bilan. Les trois événements affectent trois niveaux distincts de la chaîne de confiance : le niveau de la *spécification* (Mythos : l'IA exploite les angles morts laissés par toute spécification humaine), le niveau du *noyau* (`False` dans Rocq : le vérificateur lui-même est faillible), le niveau du *runtime* (*Watchers* : l'exécution de code prouvé demeure exposée). Trois fractures simultanées trouvées par l'IA, en quelques semaines de la même année.

2.4 Trois niveaux de confiance, ré-examinés

Il serait erroné de tirer de ce constat une conclusion nihiliste selon laquelle la vérification formelle serait vaine. Ce que ces événements établissent, c'est que la chaîne de confiance se décompose en trois couches, et que chacune ne fournit qu'une garantie *relative* à des hypothèses explicites.

Au niveau de la **spécification**, la question est de savoir si l'on a effectivement décrit ce que l'on entendait prouver. Ce travail est humain, en dernière instance philosophique, et il dépend de notre capacité à formaliser les concepts pertinents. Certaines propriétés — absence de débordement de pile, préservation de la confidentialité — se laissent spécifier avec précision. D'autres — « ne pas nuire », « agir honnêtement » — résistent à toute formalisation complète.

Au niveau du **noyau**, la garantie est relative à la correction du vérificateur. Le cas de Rocq montre que cette hypothèse n'est pas gratuite : le noyau a des bugs, qui se révèlent à mesure qu'il est soumis à des campagnes de recherche plus systématiques.

Au niveau du **runtime**, du **compilateur** et du **matériel**, la garantie est relative à la fidélité entre la sémantique formelle du programme et son exécution effective. Le cas de `lean-zip` montre que cette fidélité n'est pas acquise non plus.

L'analogie avec la pratique mathématique ordinaire est ici éclairante. Démontrer un théorème en théorie des ensembles ne revient pas à démontrer qu'il est vrai dans l'absolu, mais à démontrer qu'il est vrai *si* les axiomes de ZFC sont cohérents. De même, prouver un énoncé en Rocq ne consiste pas à établir sa vérité *sans reste*, mais à établir sa vérité *sous hypothèse* que le noyau est correct, que le runtime est fidèle, que le matériel exécute correctement le binaire. La bonne

question n'est jamais « peut-on faire confiance ? », mais toujours « à quoi fait-on confiance, et cette confiance est-elle justifiée ? »

La vérification formelle n'est donc pas réfutée par 2026 : elle est *recadrée*. Elle demeure l'outil le plus puissant dont on dispose pour produire des garanties sur des systèmes logiciels. Mais ces garanties sont conditionnelles — et la tradition philosophique rappelle depuis longtemps que tout savoir l'est.

3 Questions ouvertes

3.1 Quatre questions pour la discussion

Le déroulement précédent laisse ouvertes plusieurs questions, dont quatre méritent d'être posées explicitement.

Première question : l'IA, adversaire ou garante ? Une même capacité — explorer efficacement l'espace des programmes et des preuves — se prête à deux usages antagonistes : produire des systèmes sûrs, produire des attaques. Comment arbitrer entre ces deux usages ? Est-il concevable que la balance penche durablement d'un côté, ou faut-il se résigner à une course offensive-défensive permanente dans laquelle chaque progrès d'un camp appelle celui de l'autre ?

Deuxième question : formaliser Spinoza, formaliser Gödel. Plusieurs projets contemporains entreprennent de soumettre des textes philosophiques classiques à l'examen d'un assistant de preuve : la formalisation de l'*Éthique* de Spinoza (Pommeret 2025), les formalisations successives de la preuve ontologique de Gödel par Benzmüller (Benzmüller et Paleo 2013) puis par Kirchner et Zalta (Kirchner 2017). S'agit-il de *traductions fidèles*, qui permettraient de tester la validité formelle des raisonnements originaux ? Ou s'agit-il nécessairement de *ré-interprétations*, qui introduisent leur propre ontologie et transforment du même coup l'objet qu'elles prétendent vérifier ? À cet égard, on peut mentionner le projet plus général de Christoph Benzmüller visant à faire de la logique d'ordre supérieur une *méta-logique universelle* capable d'accueillir et de comparer des systèmes philosophiques hétérogènes (Benzmüller 2017).

Troisième question : comprendre versus vérifier. La preuve formalisée par Gonthier du théorème des quatre couleurs compte des dizaines de milliers de lignes de Coq. Personne n'en a jamais fait une lecture intégrale : seul le noyau en a vérifié la correction. Doit-on considérer cette activité comme *faire des mathématiques* ? Existe-t-il une différence épistémologique entre une preuve *comprise* et une preuve *seulement vérifiée* ? Si oui, comment la caractériser ?

Quatrième question : la sûreté prouvable, voie royale ou illusion ? L'argument de Tegmark et Omohundro repose sur la possibilité de spécifier formellement ce que signifie « sûr ». Cette possibilité est-elle effective ? Peut-on formaliser des valeurs éthiques sans altérer leur nature même ? Existe-t-il, dans l'éthique, un résidu qui résiste par essence à la formalisation ?

3.2 Bilan des six semaines

Sur le plan des *savoir-faire*, les six semaines ont permis l'acquisition de compétences précises : construire un terme qui habite un type, utiliser les constructions `Inductive`, `Fixpoint` et la tactique `induction` dans Rocq, formaliser un énoncé simple et ébaucher sa preuve. Il s'agit d'une compétence technique qui, dans un parcours philosophique, n'est pas commune.

Sur le plan *philosophique*, plusieurs thèses ont été croisées, dont on retient : la correspondance de *Curry-Howard*, qui identifie preuves et programmes au sein d'un unique langage ; le *constructivisme*, pour lequel une proposition est vraie par la production effective d'un témoin ; le *vérificationnisme*, selon lequel le sens d'une proposition est déterminé par la manière dont elle se vérifie — thèse que ce cours, dans le sillage de Dummett, a cherché à prendre au sérieux ; la *mécanisation* progressive du travail épistémique, dont une part croissante est désormais déléguée à des machines, et qui pose des questions renouvelées sur ce qui constitue une pratique rationnelle ; enfin, et peut-être surtout, le principe selon lequel aucune vérification n'abolit la confiance : elle la *déplace*, et il est crucial de savoir vers où.

Leibniz disait *calculemus*. 2026 nous rapproche de ce rêve plus que toute autre époque de l'histoire ; elle nous montre dans le même mouvement que le calcul lui-même repose sur des fondations, et que ces fondations ne sont jamais neutres. *Calculemus*, donc — mais lucidement.

Références

AL., Peter Scholze et (2021). "Liquid Tensor Experiment". In : URL : https://leanprover-community.github.io/liquid/.

ANTHROPIC (2026). *Claude Mythos Preview*. URL : https://red.anthropic.com/2026/mythos-preview/ (visité le 16/04/2026).

ARISTOTLE TEAM (HARMONIC) AND NEEL SOMANI (2026). *Resolution of Erdős Problem #728 : a writeup of Aristotle's Lean proof*. arXiv : 2601.07421. URL : https://arxiv.org/abs/2601.07421 (visité le 21/04/2026).

AVIGAD, Jeremy (4 nov. 2023). *Mathematics and the Formal Turn*. DOI : 10.48550/arXiv.2311.00007. arXiv : 2311.00007 [math]. URL : http://arxiv.org/abs/2311.00007 (visité le 17/03/2026). Prépubl.

BENZMÜLLER, Christoph (28 mars 2017). *Universal Reasoning, Rational Argumentation and Human-Machine Interaction*. DOI : 10.48550/arXiv.1703.09620. arXiv : 1703.09620 [cs]. URL : http://arxiv.org/abs/1703.09620 (visité le 17/03/2026). Prépubl.

BENZMÜLLER, Christoph et Bruno Woltzenlogel PALEO (2013). "Formalization, Mechanization and Automation of Gödel's Proof of God's Existence". In : *arXiv preprint*. URL : https://arxiv.org/abs/1308.4526.

BRUIJN, N. G. de (1980). "The Mathematical Language Automath, Its Usage, and Some of Its Extensions". In : *Synthese* 43.1, p. 1-29. URL : https://doi.org/10.1007/BF00485013.

BUZZARD, Kevin (11 fév. 2025). *Mathematical Reasoning and the Computer*. DOI : 10.48550/arXiv.2502.07850. arXiv : 2502.07850 [cs]. URL : http://arxiv.org/abs/2502.07850 (visité le 17/03/2026). Prépubl.

DEEPMIND, Google (2024). "AlphaProof and AlphaGeometry 2 solve IMO problems". In : URL : https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/.

GONTHIER, Georges (2008). "Formal Proof—The Four Color Theorem". In : *Notices of the AMS*. URL : https://www.ams.org/notices/200811/tx081101382p.pdf.

GOOGLE DEEPMIND (2026). *Aletheia : Autonomous Mathematical Discovery with Gemini 3 Deep Think*. URL : https://deepmind.google/blog/accelerating-mathematical-and-scientific-discovery-with-gemini-deep-think/ (visité le 21/04/2026).

GOPINATHAN, Kiran (2026). *Who Watches the Watchers ? Fuzzing a Formally-Verified ZIP Parser*. URL : https://kirancodes.me/posts/log-who-watches-the-watchers.html (visité le 16/04/2026).

HALES, Thomas et al. (2017). "A Formal Proof of the Kepler Conjecture". In : *Forum of Mathematics, Pi*. URL : https://doi.org/10.1017/fmp.2017.1.

KIRCHNER, Daniel (2017). "Representation and Partial Automation of the Principia Logico-Metaphysica in Isabelle/HOL". In : *Archive of Formal Proofs*. URL : https://www.isa-afp.org/entries/PLM.html.

KLEIN, Gerwin et al. (2009). "seL4 : Formal verification of an OS kernel". In : *SOSP*. URL : https://dl.acm.org/doi/10.1145/1629575.1629596.

LEROY, Xavier (2009). "Formal verification of a realistic compiler". In : *Communications of the ACM* 52.7, p. 107-115. URL : https://dl.acm.org/doi/10.1145/1538788.1538814.

MASSOT, Patrick (5 déc. 2021). *Why Formalize Mathematics ?* URL : https://www.imo.universite-paris-saclay.fr/~patrick.massot/files/exposition/why_formalize.pdf.

NATSOTHANAPHAN (2026). "AI contributions to Erdős problems". In : URL : https://github.com/teorth/erdosproblems/wiki/AI-contributions-to-Erd%5C%C5%5C%91s-problems/.

POMMERET, Luc (2025). "Preuves en déduction naturelle du livre I de l'Éthique de Spinoza". In : *Document de travail (lucpommeret.com)*. URL : https://lucpommeret.com/assets/De_Deo.pdf.

Professor Kevin Buzzard (1er oct. 2025). *Professor Kevin Buzzard : What Is Formalization and Why Does It Matter ?* URL : https://www.youtube.com/watch?v=0fy8VURvwac (visité le 17/03/2026).

ROCQ ZULIP COMMUNITY (2026). *Proof of False Found by Opus 4.6 and mxdys (bbchallenge)*. URL : https://rocq-prover.zulipchat.com/#narrow/channel/237977-Rocq-users/topic/Proof.20of.20false.20found.20by.20Opus.204.2E6.20and.20mxdys.20.28bbchallenge.29/with/580830257 (visité le 16/04/2026).

SEVERINI, Simone (2026). *Infrastructure for Mathematics*. URL : https://simoneseverini.github.io/infrastructure-for-mathematics.html (visité le 16/04/2026).

STÉRIN, Tristan (2026). *In Search of Falsehood : AI-Assisted Hunt for Inconsistencies in Rocq*. URL : https://tristan.st/blog/in_search_of_falsehood (visité le 20/04/2026).

TEGMARK, Max et Steve OMOHUNDRO (5 sept. 2023). *Provably Safe Systems : The Only Path to Controllable AGI*. DOI : 10.48550/arXiv.2309.01933. arXiv : 2309.01933 [cs]. URL : http://arxiv.org/abs/2309.01933 (visité le 16/04/2026).

THOMPSON, Ken (1984). "Reflections on Trusting Trust". In : *Communications of the ACM* 27.8, p. 761-763. URL : https://dl.acm.org/doi/10.1145/358198.358210.