Entretien avec Pierre-Evariste Dagand – Coq

Présentation générale, cursus scolaire et parcours professionnel

Une passion pour l'informatique

À la base j'ai grandi avec les ordinateurs, je voulais faire de l'informatique. Et ensuite, ce qui m'intéressait c'était les systèmes : systèmes distribués, systèmes pair-à-pair, systèmes d'exploitation. Avec toujours (quand on est jeune on est fougueux) le piratage, les trucs comme ça. Voilà, c'est ce qui m'attirait quand j'étais gamin. Et rapidement, par une sorte de hasard, je suis tombé sur des gens à l'ENS (École Normale Supérieure) qui disaient des choses bizarres, comme : en fait, un système d'exploitation, c'est juste un environnement pour un langage de programmation. Je suis tombé dedans un peu par hasard, et c'est en lien avec ce qui se fait en théorie des types, mais il m'a fallu dix ans pour apprendre ça. C'est ce qui m'a fait passer de "casser les systèmes" à "implémenter les systèmes" vers "étudier les langages qui servent à implémenter ces systèmes".

Parcours scolaire

Au niveau de mon cursus universitaire, je suis entré à l'ENS à Ker Lann en Bretagne, et la spécialité là-bas c'était vraiment les systèmes et les télécommunications. J'ai fait pas mal de systèmes distribués, même si c'est très frustrant : on fait des algorithmes très simples (sur 5 lignes) mais la preuve est extrêmement subtile, et beaucoup de personnes se demandent comment on peut être certain que la preuve est correcte. Le grand chef des systèmes distribués, c'est Leslie Lamporte, et ses contributions majeures à l'informatique sont justement des langages pour spécifier des systèmes. On est sur des choses qui sont très subtiles et très complexes, et pour lesquels, contrairement aux maths, l'intuition humaine n'est pas forcément très adaptée. Il s'agit de systèmes concurrents où les choses se passent en même temps, et on a du mal à garder toute cette complexité, il faut des outils.

Vision de la nature des mathématiques et de l'informatique

Je pense qu'il y a une **différence très profonde entre les maths et l'informatique ; dans les maths il y a une très grosse profondeur conceptuelle, alors qu'en informatique en termes de profondeur conceptuelle on ne va généralement pas très loin.** On fait de la théorie des ensembles très facile, sans jamais rentrer dans les détails, avec des abstractions énormes. Par contre, on va avoir énormément de largeur. Si on pense la recherche comme l'exploration dans un arbre, en profondeur on ne descend pas très bas, mais il faut gérer beaucoup de complexité en largeur. Quelqu'un me disait : "***En maths, il n'y a pas de problème de preuve fausse, on a parfois des preuves qui sont fausses mais avec un peu de boulot on arrive à les réparer.***" Ça se vérifie avec le théorème de Fermat par exemple, initialement la preuve était fausse mais elle a été réparée. Il y a des intuitions tellement puissantes que même si le raisonnement était faux, généralement l'intuition était correcte. Quand on est sur des systèmes distribués, il est très difficile d'avoir les intuitions, et on peut vraiment avoir des intuitions

fausses. C'est là que j'ai ressenti le besoin d'avoir un outil à côté pour m'assurer que mon raisonnement était juste, et ça m'a amené à me pencher sur les raisonnements vérifiés par ordinateur.

En parallèle de cela, j'étais en train de suivre un cursus qui s'orientait de plus en plus vers les langages de programmation, et en fait les deux se connectent : la preuve par ordinateur et la programmation se rejoignent avec des outils comme Coq. Pour moi, la boucle était bouclée à ce moment-là. D'un côté j'étais en train de construire des langages de programmation, et la meilleure façon de le faire aurait été d'utiliser Coq, et de l'autre je voulais m'assurer que mes systèmes étaient corrects, et la meilleure façon aurait été d'utiliser Coq. J'ai donc fait ma thèse en Ecosse sur un système qui n'était pas Coq mais plus ou moins équivalent, avec des différences de techniques mais la même idée de base : la théorie des types dépendants, et comment l'exploiter pour faire des programmes et des preuves dans un environnement uniforme.

Nous nous intéressons aussi à cette relation, quand les mathématiques s'approprient des outils informatiques pour démontrer des théorèmes. Nous nous sommes intéressés à la controverse autour du théorème des 4 couleurs, et nous cherchons à savoir si cette controverse entre mathématiques et informatique est vraiment finie, si les mathématiciens ont complètement accepté qu'on puisse prouver formellement des théorèmes mathématiques via l'informatique.

Je ne peux pas répondre car je ne suis pas mathématicien, j'en ai fortement conscience au quotidien d'ailleurs. Je ne peux pas répondre à leur place. Du point de vue des informaticiens, on est en train de grignoter les logiciens, mais les logiciens eux-mêmes sont-ils des mathématiciens ? Je connais des logiciens qui sont dans une névrose car ils ont l'impression d'être des moins-que-mathématiciens. Ce que je peux dire, c'est que le théorème des quatre couleurs, et plus récemment, parce que le 4CT ça commence un peu à sentir la poussière, le théorème de Feit-Thompson, j'aurais tendance à te dire : **Gonthier a pris des preuves qui étaient énormes, et il les a vérifiées par ordinateur ; mais à la fin, il n'a pas trouvé que la preuve était fausse**, donc est-ce qu'on a gagné quelque chose, je ne sais pas. **Je pense que chez les mathématiciens, il y a toujours la frustration de ne pas avoir une preuve élégante d'un résultat** : même si elle est vérifiée par ordinateur, je pense que mathématiquement ce n'est pas très satisfaisant. De mon point de vue, les démonstrations de Feit-Thompson et du 4CT c'est génial, car en faisant ces efforts, ils ont développé des outils que j'utilise pour faire la correction de vrais systèmes. Ça c'est fabuleux.

La théorie des types homotopiques, il faut absolument que vous regardiez ça. Voevodsky c'est un vrai mathématicien, il a eu la médaille Fields, il sait ce qu'il fait, et il est en train de mettre l'informatique, les outils de démonstrations mécanisées, au cœur de son programme de recherche en théorie des types homotopiques. Pourquoi ? Parce qu'il s'est rendu compte qu'en lisant les papiers de ses petits camarades il ne pouvait pas savoir s'ils étaient vrais ou faux, c'était devenu trop technique. Ils essayent de construire un **système formel dans lequel ils vont pouvoir faire leurs preuves et les vérifier mécaniquement**. Il a donné une leçon inaugurale à Princeton dans laquelle il dit : "Je ne pense pas,

personnellement, que les mathématiques soient consistantes ; je pense que quelqu'un arrivera un jour avec une preuve que la théorie de Peano*, c'est-à-dire des mathématiques avec pas grand-chose dedans, cette théorie-là est fausse.". Ça a l'air d'être sa conviction intime.

** 5 axiomes de base :*

- L'élément appelé zéro et noté 0, est un entier naturel.
- Tout entier naturel n a un unique successeur, noté s(n) ou Sn.
- Aucun entier naturel n'a 0 pour successeur.
- Deux entiers naturels ayant le même successeur sont égaux.
- Si un ensemble d'entiers naturels contient 0 et contient le successeur de chacun de ses éléments, alors cet ensemble est égal à N

Est-ce en lien avec les théorèmes d'incomplétude de Gödel ?

Exactement. **A priori on ne peut pas savoir si une théorie est consistante rien qu'en la regardant**, et il pense que même dans une théorie relativement faible, on pourrait avoir une preuve de fausse qui se balade à l'intérieur. A partir de là, la seule manière de vérifier que les preuves soient vraies, c'est de les exécuter. D'où l'intérêt d'avoir l'ordinateur derrière, qui permet de prendre la preuve et de la calculer, de voir si la preuve correspond vraiment à un programme qui termine. C'est la correspondance preuve-programme.

Nous avions une question plutôt autour du fonctionnement de Coq. Nous nous demandions si Coq faisait vraiment un travail qu'aurait pu faire un mathématicien, mais en balayant un grand nombre de cas possibles.

Non. **Coq ne fait rien seul**. Je peux vous donner l'implémentation de Coq sur le tableau ici. Si mathématiquement et scientifiquement, quand Coq nous dit "Je crois à cette preuve" on y croit aussi, **c'est parce que le noyau est hyper simple.**

La première chose, c'est que dans le système il y a une notion de type. Vous pouvez voir ça comme un ensemble (construit de manière inductive, intervention de Tom). Pour définir un type, on dit qu'un type est un type : mais comme c'est un peu circulaire comme définition on dit qu'il y a des types de taille n et des types de taille supérieure, c'est ce que Russel appelait "typical ambiguity". Il faut s'assurer qu'on puisse toujours stratifier, avec des types de taille inférieure à d'autres.

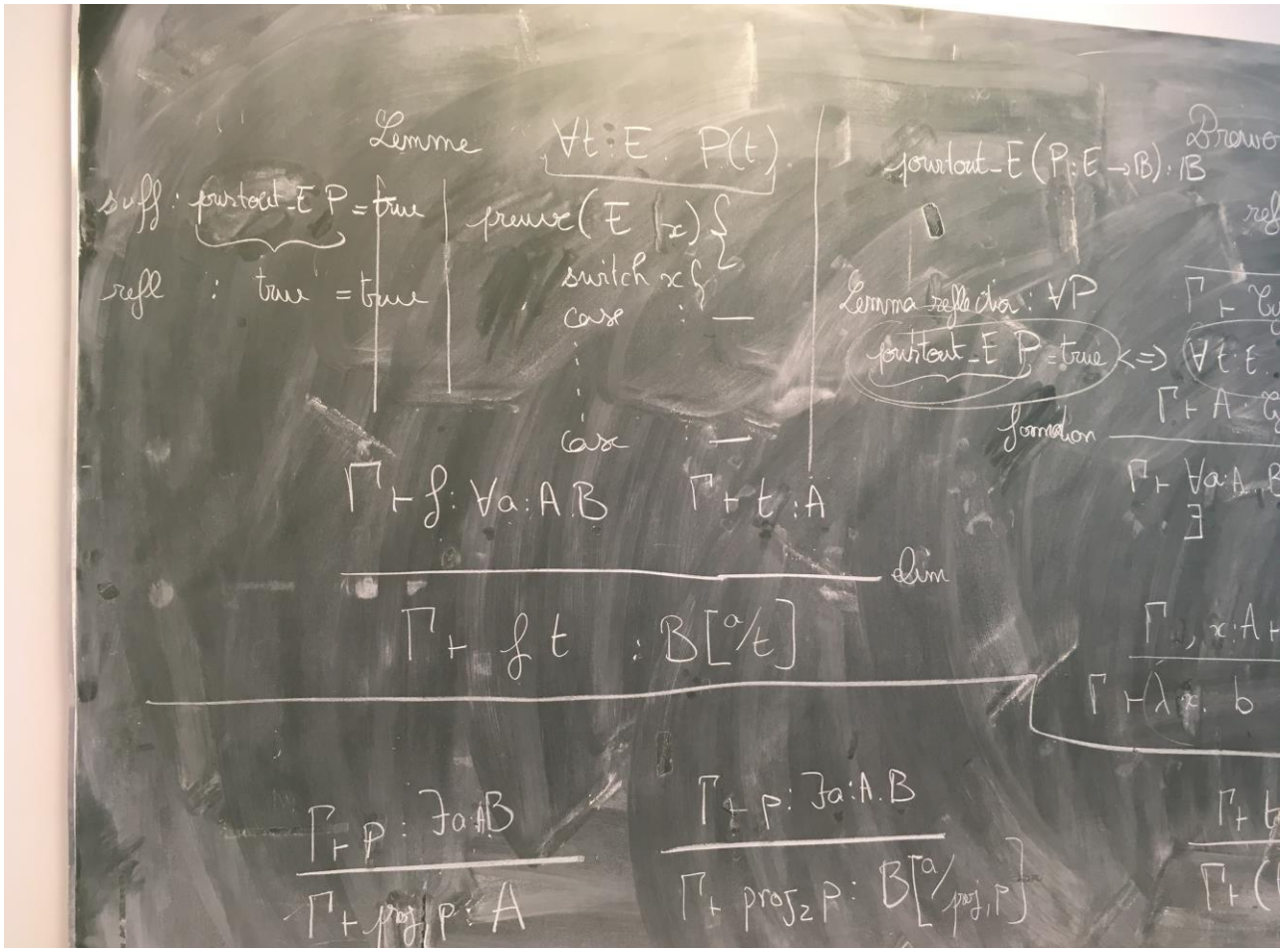
Et en fait ce que fait Coq, c'est qu'il cherche à construire un témoin. Cette construction-là est complètement décidable, complètement mécanique, il n'y a pas de travail d'invention : Coq ne fait que vérifier des choses. Mais écrire des preuves comme ça sur Coq c'est extrêmement douloureux, donc au-dessus il y a des mécanismes pour automatiser, avoir des notations sympathiques, etc. mais c'est juste du confort, ça n'a rien à voir avec la théorie des types en tant que telle.

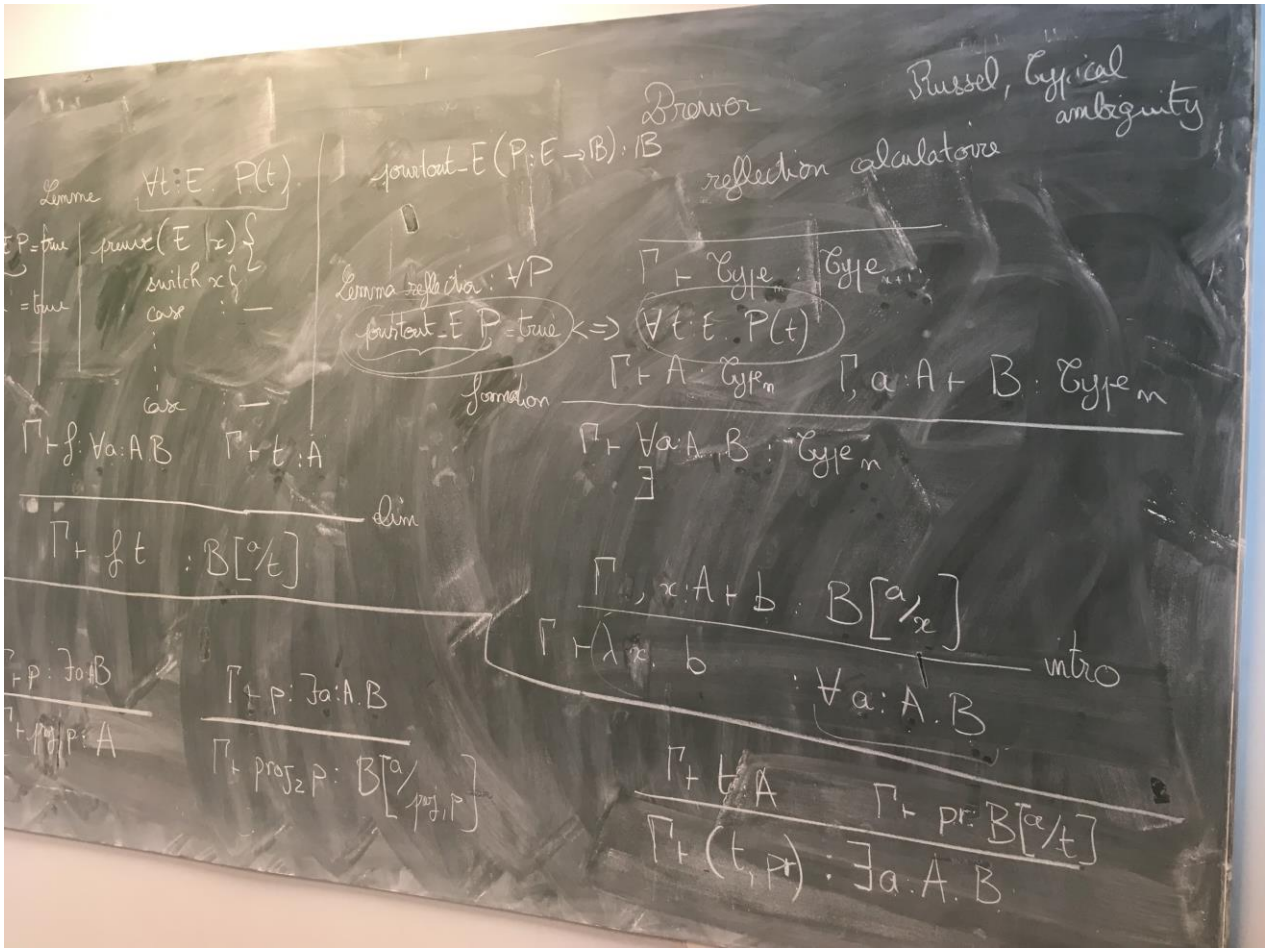
En pratique, notamment dans les théorèmes des 4 couleurs et de Feit-Thompson, il n'y a pas d'axiomes. Ce que vous appelleriez des axiomes, c'est cette définition-là qui est au tableau. Là on est en train de faire des mathématiques, et on retombe sur Gödel : on est en train de définir un système formel. Comment ça se fait que le système formel ne permet pas de prouver faux

? On ne peut pas le prouver en utilisant Coq. Après il faut se plonger dans la théorie des ensembles, et se dire : ça me semble correct. On ne peut pas prouver en Coq que Coq est consistant mais on peut s'en approcher très fortement.

Question de l'élégance

Je pense aussi qu'une preuve est plus élégante si elle fait 10 pages plutôt que 200 pages, et encore plus si elle ne fait qu'une page. Je ne pense pas qu'il y ait à défendre l'intérêt de l'informatique pour de telles preuves, je pense que les assistants de preuve peuvent fournir autre chose : moi ils me servent d'un point de vue informatique, car on est sur des preuves qui sont énumératives et combinatoires.




Lemme ∀t:E. P(t)
prouve(E x) {
switch x {
cas :
cas :
fonction_E(P : E → B) : B
Lemma reflection : ∀P
fonction_E P = true ⇔ ∀t:E. P(t)
Brouwer
Russel, Typical ambiguity
réflection calculatoire
Γ ⊢ Type_n : Type_{n+1}
formation
Γ ⊢ A : Type_n   Γ, a:A ⊢ B : Type_n
Γ ⊢ ∀a:A. B : Type_n
Γ ⊢ f : ∀a:A.B   Γ ⊢ t : A
Γ ⊢ f t : B[a/t]
elim
Γ, x:A ⊢ b : B[a/x]
Γ ⊢ (λx. b) : ∀a:A.B
intro
Γ ⊢ p : ∃a:A.B
Γ ⊢ proj1 p : A
Γ ⊢ p : ∃a:A.B
Γ ⊢ proj2 p : B[a/proj1 p]
Γ ⊢ t : A   Γ ⊢ pr : B[a/t]
Γ ⊢ (t, pr) : ∃a.A.B