Conception et développement d'un outil d'aide à la rédaction de démonstrations mathématiques reposant sur Coq

Sujet de thèse de Théo Zimmermann, sous la direction d'Hugo Herbelin

Contexte

Les mathématiques formelles, c'est-à-dire la rédaction de démonstrations dans un langage tel qu'un ordinateur soit capable de les vérifier, sont un champ d'études depuis plus de quarante ans (de Bruijn, 1970) mais n'ont commencé à attirer une communauté importante d'utilisateurs que très récemment. Les assistants de preuve, qui sont des logiciels qui aident à écrire et à vérifier ce genre de démonstrations, intéressent à la fois la communauté de la certification (logicielle, matérielle) et la communauté mathématique. Cette dernière a, dans un passé récent, produit des démonstrations d'une taille telle qu'elle commence à craindre que la relecture par les pairs ne soit plus suffisante pour en garantir la validité, d'où des initiatives de formalisation de preuves difficiles telles que celle de la conjecture de Kepler (Hales, 2008) ou celle du théorème des quatre couleurs (Gonthier, 2008). Même s'ils ont atteint un certain niveau de maturité, les assistants de preuve sont encore difficiles à utiliser et les rendre plus accessibles, pour permettre qu'une communauté bien plus importante les utilise, est une préoccupation de plus en plus grande parmi les chercheurs du domaine.

Le système Coq (Coq development team, 2016) est un assistant de preuve basé sur un formalisme qui est à la fois un système logique expressif et un langage de programmation fortement typé. Il a récemment gagné en popularité grâce à deux prix ACM et l'exploit de formalisation de la preuve du théorème des groupes d'ordre impair par Georges Gonthier et son équipe (Gonthier, et al., 2013). Cependant, cela reste un outil plutôt réservé aux utilisateurs informaticiens, comme l'illustre la manchette du Monde sur la formalisation du théorème des groupes d'ordre impair présentant à juste titre Georges Gonthier comme un « hacker de théorèmes ».

Des mathématiciens commencent malgré tout à s'intéresser à Coq. Un lien a été établi entre la théorie des types, c'est-à-dire le formalisme implanté par Coq, et la théorie de l'homotopie via les catégories d'ordre supérieur, plaçant celle-ci au centre d'une constellation reliant algèbre, informatique et logique. Par exemple, le médaillé Fields Vladimir Voevodsky s'est rallié à l'idée que la théorie des types allait fournir une meilleure fondation aux mathématiques que la théorie des ensembles, de par sa versatilité, la richesse de ses types, la richesse de son expressivité calculatoire, la diversité des notions d'égalité (calculatoire, extensionnelle), et, surtout son aptitude à formaliser de manière algébrique des structures géométriques complexes (The Univalent Foundations Program, 2013).

Pour élargir encore plus la communauté de mathématiciens utilisant Coq, au-delà des « hackers » et de ceux qui s'intéressent aux questions des fondations des mathématiques, il faudra cependant apporter une valeur ajoutée supplémentaire. Nous proposons de rendre Coq utile à l'ensemble de la communauté mathématique en en faisant un outil d'aide à la rédaction de démonstrations (et par extension d'articles mathématiques), intervenant en remplacement de LaTeX.

Objectifs

Coq est un outil interactif. Les preuves sont construites comme un enchaînement de tactiques (*scripts de preuve*) manipulant le but courant et le contexte, et construisant au passage un *terme de preuve* qui sera lui-même vérifié par le système.

Un des principaux reproches qui est régulièrement fait à Coq est l'obscurité de ses scripts de preuve (Strickland, 2014). Conséquemment, une preuve Coq a besoin, pour être bien comprise, d'être accompagnée d'une documentation, qui s'apparente à la preuve informelle. Cette documentation

n'est bien entendu pas garantie : seuls les scripts obscurs sont vérifiés et c'est à l'utilisateur de voir le lien entre la documentation et le script. Pour comprendre le script de preuve lui-même, il est généralement conseillé de le rejouer, c'est-à-dire de demander à Coq de l'évaluer ligne par ligne pour comprendre ce que fait chaque ligne et comment l'état de la preuve s'en retrouve transformé.

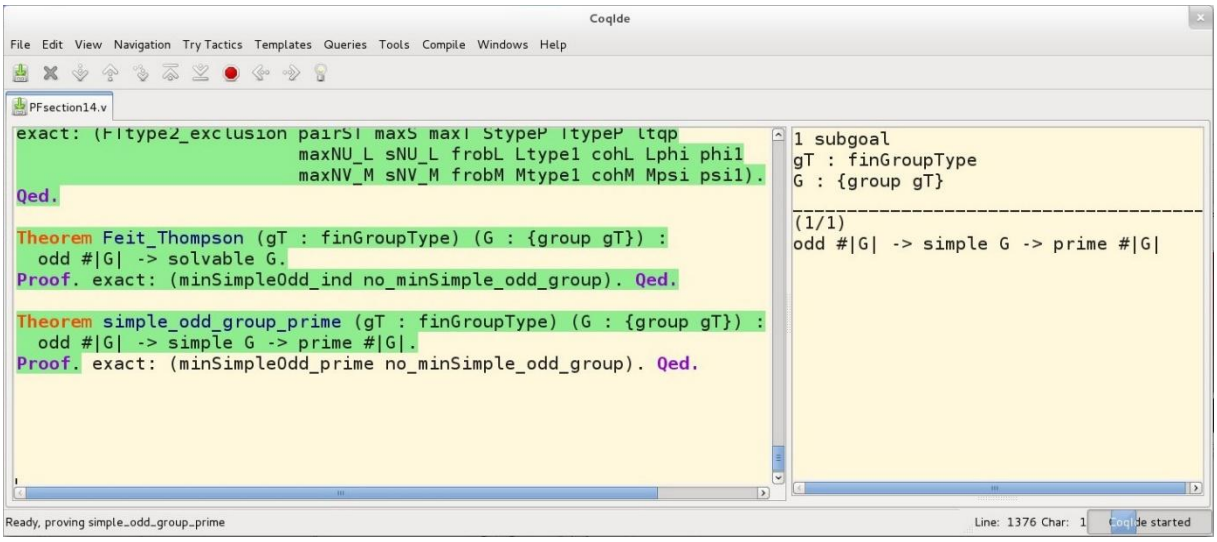

Figure 1 : Extrait de la preuve formelle du théorème de Feit-Thompson (Gonthier, et al., 2013), affichée dans CoqIDE.

Nous nous donnons comme objectif de créer un outil capable d'interpréter des scripts de preuve et d'en générer, au fur et à mesure de leur construction, une représentation en langue naturelle. De plus, cette représentation devra garder un niveau de rigueur suffisant pour pouvoir être traduite à nouveau en un script de preuve accepté par Coq.

De nombreux travaux de recherche ayant pour but la verbalisation de preuves formelles ont déjà eu lieu. En ce qui concerne Coq, Coscoy (2000) a travaillé sur la traduction des termes de preuve. Le principal problème de son approche était que les textes produits étaient très détaillés et beaucoup trop longs. Nous pensons qu'un inconvénient supplémentaire consiste en la perte d'une partie du sens implicite contenu dans les scripts de preuve (notamment quelles sont les étapes faciles qui valent la peine d'être traitées automatiquement).

Par conséquent, nous travaillerons pour notre part sur la traduction des scripts de preuve. Nous envisageons une traduction passant par plusieurs étapes intermédiaires (à la manière de ce que font les compilateurs modernes), qui seraient toutes des scripts Coq valides, mais de plus en plus proches de la forme rédigée. La première étape se contenterait d'ajouter de la structure tandis que l'étape finale serait la traduction du dernier script (aussi proche sémantiquement de la forme rédigée que possible) en anglais, et cette étape serait réversible.

Une expérimentation préliminaire a permis de montrer que les scripts de preuves peuvent en effet être rendus très proches d'une preuve rédigée, dans l'état actuel du langage de tactiques. La lisibilité est grandement améliorée par la mise en valeur des grandes étapes de la preuve et le masquage des détails de chaque étape (qui peuvent cependant être dévoilés progressivement, à la demande de l'utilisateur). Cette présentation a été inspirée par le concept de preuves hiérarchique structurées, défendu à l'origine pour des démonstrations mathématiques informelles (Lamport, 2012).

Cette forme de présentation se traduit très facilement en langue naturelle comme le montrent les deux figures suivantes.

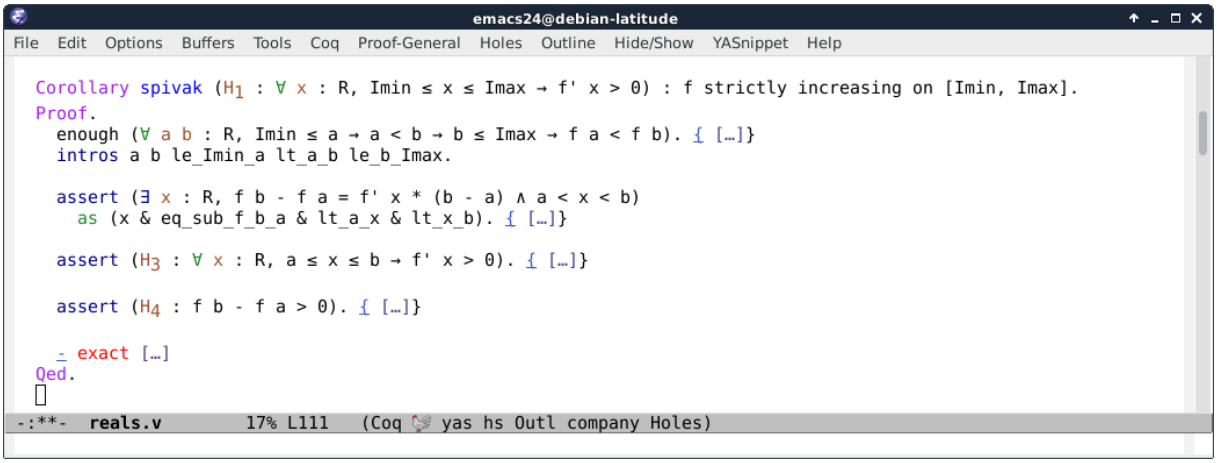

Figure 2 : Preuve en Coq d'un corollaire du théorème des valeurs intermédiaires, affichée dans Emacs avec Proof-General et Company-Coq.

Corollary: If f'(x) > 0 for all x in [I_{min} , I_{max}], then f is strictly increasing on [I_{min}, I_{max}].
Proof:

1. It suffices to take two reals a < b in [I_{min} , I_{max}], and prove f a < f b.
2. There is some x in [a , b] such that f(b) – f(a) = f'(x) (b – a).
3. But f'(x) > 0 for all x in [a , b].
4. Thus, f(b) – f(a) > 0.
5. From this we conclude.

Qed.

Figure 3 : Preuve d'un corollaire du théorème des valeurs intermédiaires, traduite en anglais.

De même que l'on peut masquer ou montrer des détails dans le script de preuve, on pourra masquer ou montrer des détails dans la traduction textuelle. Dans le cas d'un affichage sur ordinateur, notamment sur le web, le document produit pourra comporter des éléments d'interactivité, comme proposé par (Oostdijk, 2001), par exemple le dépliage de plus de détails bien sûr mais aussi des info-bulles indiquant les énoncés des lemmes utilisés et les définitions, des hyperliens vers les preuves de ces lemmes, etc.

Dans le cas d'une traduction destinée à une impression papier, ce sera à l'auteur de déterminer le niveau de détail à afficher. D'autres éléments de personnalisation seront disponibles, tel que le choix des mots de liaisons et de certaines formulations, le fait de quantifier avant ou après une formule, ou encore d'utiliser plus ou moins de symboles ou de texte. La contrainte sera que tout le texte généré devra être ré-interprétable. Ainsi l'intégralité de la « documentation » ainsi produite sera vérifiable.

Les scripts de preuve peuvent être extrêmement variés et comporter de nombreuses tactiques personnalisées. C'est pourquoi nous restreignons l'objectif de cette thèse à la traduction des théorèmes d'arithmétique (et des tactiques associées). Cependant, la conception de cet outil devra être modulaire et extensible, de sorte que d'autres puissent très facilement contribuer à en étendre les capacités.

Au niveau théorique, un des enjeux est de comprendre comment les transformations entre différents styles de preuve préservent la capacité de reconstruire l'information implicite (information reconstructible par l'unification ou par une méthode de décision ou de semi-décision).

Discussion

On pourrait craindre que les mathématiciens aient du mal à se servir du langage d'entrée (langage de tactiques de Coq) pour la rédaction de leurs démonstrations car il serait éloigné de la forme finale. Or deux exemples devraient permettre de nous rassurer : les mathématiciens pratiquent tous ou presque à l'heure actuelle le langage LaTeX, lui aussi complexe et différent du rendu obtenu ; et ils sont nombreux à utiliser des logiciels de calcul formel qui eux aussi ont un langage d'entrée bien différent des jolies formules mathématiques qui peuvent être affichées en sortie.

Quant à la difficulté inhérente à la formalisation d'une démonstration, elle peut se réduire à deux questions principales : le manque d'automatisation, sur lequel de nombreuses avancées ont récemment été accomplies et continuent de l'être ; ainsi que le besoin de formaliser également les concepts et les lemmes élémentaires sur lesquels on se repose. Ce dernier point justifie que l'on fournisse des bibliothèques avancées, basées sur les formalisations ayant déjà eu lieu. Le cas d'un lemme manquant est cependant moins problématique dans le cas de l'application à la rédaction car celui-ci peut très facilement être admis.

Le style de preuve principalement utilisé dans Coq, qualifié de style procédural et correspondant à une vision dans laquelle l'utilisateur décompose un but en plusieurs sous-buts, peut aussi paraître surprenant. Différents assistants de preuve présentent en effet des visions différentes du style que devrait adopter un script de preuve.

Mizar (Naumowicz & Kornilowicz, 2009) est un assistant de preuve qui a toujours défendu le style déclaratif (souvent considéré plus proche du style des démonstrations informelles, dans lequel on enchaîne les énoncés découlant des résultats précédant). Il a inspiré un langage « mathématique » pour Coq (Corbineau, 2008) qui n'a jamais eu le succès escompté. Au contraire, Isabelle (Nipkow, Wenzel, & Paulson, 2002) a récemment imposé à ses utilisateurs une transition du style procédural vers un style déclaratif.

Le style déclaratif correspond en fait à peu de choses près à celui des démonstrations mathématiques rédigées. En revanche, le style de recherche de preuve des mathématiciens n'est pas aussi strict. Un bon assistant de preuve devrait pouvoir suivre le fil de la pensée du mathématicien tout en étant capable de produire une version rédigée correspondant au plus près aux attentes stylistiques d'une publication. C'est l'objectif recherché par cette thèse.

Finalement, un élément essentiel du succès de notre outil résidera dans la qualité de son interface, notamment la présence de fonctions telles que la suggestion automatique et l'auto-complétion, qui facilitent l'apprentissage du langage. Nous ne prévoyons pas de nous attaquer nous-mêmes à ces questions d'interfaces mais nous suivrons de près les innovations les concernant. En effet, on peut se féliciter d'un bouillonnement autour de ces questions à l'heure actuelle, avec de nombreux outils et idées qui sont développés et testés. On peut citer en particulier l'extension Company-Coq pour Emacs (Pit--Claudel & Courtieu, 2016) qui comporte des fonctions d'auto-complétion, PeaCoq (Robert, 2015) qui teste différentes voies possibles pour la preuve et les propose à l'utilisateur avec le résultat associé, ou encore JsCoq (Gallego Arias, 2016) qui fonctionne entièrement dans le navigateur du client.

Bibliographie

Coq development team. (2016). *The Coq proof assistant reference manual, Version 8.5.* Inria.

Corbineau, P. (2008). A declarative language for the Coq proof assistant. *Types for Proofs and Programs* (pp. 69-84). Springer.

Coscoy, Y. (2000). *Explication textuelle de preuves pour le calcul des constructions inductives.* Thèse de doctorat, Nice.

Gallego Arias, E. J. (2016). *JsCoq*. Récupéré sur https://github.com/ejgallego/jscoq

Gonthier, G. (2008). Formal proof-the four-color theorem. *Notices of the AMS*, 55(11):1382-1393.

Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., . . . Théry, L. (2013). A machine-checked proof of the odd order theorem. *Interactive Theorem Proving* (pp. 163-179). Springer.

Hales, T. C. (2008). Formal proof. *Notices of the AMS, 55*(11), 1370-1380.

Lamport, L. (2012). How to write a 21st century proof. *Journal of Fixed Point Theory and Applications*, 43-63.

Naumowicz, A., & Kornilowicz, A. (2009). A brief overview of Mizar. *Theorem Proving in Higher Order Logics* (pp. 66-72). Springer.

Nipkow, T., Wenzel, M., & Paulson, L. C. (2002). *Isabelle/HOL: a proof assistant for higher-order logic* (Vol. 2283). Springer.

Oostdijk, M. (2001). *Generation and presentation of formal mathematical documents.* Thèse de doctorat, Eindhoven.

Pit--Claudel, C., & Courtieu, P. (2016). Company-Coq: taking Proof General one step closer to a real IDE. *CoqPL'16: The Second Internationl Workshop on Coq for PL.* Récupéré sur Github.

Robert, V. (2015). *Introducing PeaCoq*. Récupéré sur http://goto.ucsd.edu/~vrobert/coq-en-stock/blog/2015/06/03/introducing-peacoq/

Strickland, N. (2014). Proof assistants as a routine tool. *Homotopy Type Theory Workshop.* University of Oxford.

The Univalent Foundations Program. (2013). *Homotopy type theory: univalent foundations of mathematics.* Institue for Advanced Studies.