HAL open science

Combinatoire certifiée

Alain Giorgietti, Nicolas Magaud

▶ **To cite this version:**

Alain Giorgietti, Nicolas Magaud. Combinatoire certifiée. GdR Genie de la Programmation et du Logiciel, Defis 2030, pp.61-65, 2021. hal-03417881

HAL Id: hal-03417881

https://hal.science/hal-03417881

Submitted on 5 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Combinatoire certifiée *

Alain Giorgetti[1] et Nicolas Magaud[2]
[1] Institut FEMTO-ST (UMR CNRS 6174), Université de Franche-Comté
alain.giorgetti@femto-st.fr
[2] ICube (UMR CNRS 7357), Université de Strasbourg
magaud@unistra.fr

Résumé

Ce document présente des défis liés au développement de la combinatoire certifiée, définie comme l'application des méthodes formelles du génie logiciel à la recherche en combinatoire.

Ce défi est lié aux thématiques du groupe de travail LTP (Langages, Types et Preuves) du GdR GPL. Il a été alimenté par des interactions au sein de ce groupe.

1 Contexte et problématique

La combinatoire est la branche des mathématiques qui étudie les familles de données munies d'une taille telle qu'il existe un nombre fini de données différentes de chaque taille. La combinatoire énumérative s'intéresse au dénombrement de ces structures combinatoires. La combinatoire bijective établit des bijections structurelles non triviales entre diverses familles de structures combinatoires.

Le domaine de la combinatoire est encore peu perméable aux méthodes formelles de spécification et de preuve de théorèmes et de programmes. Souvent, les combinatoriciens utilisent un système de calcul formel, comme Maple, Mathematica ou Sage. Rares sont ceux qui connaissent et pratiquent la démonstration automatique ou assistée par ordinateur. On peut néanmoins citer les exemples majeurs de démonstration formelle du théorème des quatre couleurs [19] et de la correction des fonctions principales du logiciel SCHUR de calcul des caractères des groupes de Lie et des fonctions symétriques [4].

2 Défis identifiés

En complémentarité de ces "tours de force", qui visent des théorèmes majeurs, qui ont occupé plusieurs chercheurs pendant plusieurs années et qui s'appuient sur de larges bibliothèques de formalisation des mathématiques, il reste à inventer, outiller et diffuser une "*combinatoire formelle pratique*" qui relève les défis suivants :

1. Applicabilité non pas à des théorèmes connus, mais à des conjectures en cours d'élaboration par les combinatoriciens. C'est un défi ambitieux, car le temps requis pour achever une preuve formelle est très supérieur à celui de l'élaboration et de la rédaction d'une preuve papier, mais c'est un enjeu important pour intéresser la communauté de la combinatoire.

*Le premier auteur est le porteur du défi.

2. Simplicité de mise en œuvre. Il s'agit de réduire la pente et la longueur de la courbe d'apprentissage des outils de vérification formelle.
3. Diffusion des méthodes et outils de verification formelle dans la communauté des chercheurs en combinatoire.

3 Moyens

Les moyens envisagés pour relever ces défis sont :

1. Viser la formalisation de problèmes combinatoires plus modestes que les "tours de force" précédemment cités, pour obtenir des résultats dans des délais plus courts, afin que les efforts investis en formalisation et recherche de preuve produisent des gains de productivité de nouveaux théorèmes.
2. Intégrer dans chaque outil de formalisation et de vérification la possibilité de tester des conjectures, des propriétés et les conditions de vérification produites. Ce levier important est détaillé dans la partie 4.
3. Proposer des théories logiques et des langages de spécification et de programmation adaptés aux objets de la combinatoire.
4. Intégrer dans les outils de vérification de ces théories, spécifications et programmes l'état de l'art en matière de test de propriété et d'automatisation des preuves.
5. Diffuser des bibliothèques d'exemples de théorèmes vérifiés formellement, et une méthodologie guidée pour en produire d'autres.

4 Test de propriétés

Dans un assistant de preuve, il est précieux de se convaincre qu'un lemme est correct avant d'investir du temps dans sa démonstration interactive. Tester une propriété avant d'en commencer une démonstration interactive permet de se convaincre qu'elle est correcte, ou sinon de gagner un temps précieux en détectant rapidement une faille de raisonnement ou une erreur de formulation.

Le *test de propriété* (PBT, pour *Property-Based Testing*) est la recherche d'un contre-exemple pour une propriété d'un programme en cours de vérification. Il est populaire pour les langages fonctionnels, notamment avec l'outil QuickCheck [6] dans Haskell. Le PBT a également été adapté à des assistants de preuve, comme Isabelle [2], Agda [10], PVS [22], FoCaLiZe [5] et plus récemment Coq [23], avec l'outil de test aléatoire QuickChick. Dans ce cadre des assistants de preuve, il permet de tester des conjectures.

Parmi les méthodes de test automatique, le test exhaustif borné (BET, pour *Bounded Exhaustive Testing*) d'une fonction ou propriété consiste à générer toutes ses données d'entrée jusqu'à une certaine taille. Le BET est particulièrement bien adapté aux structures combinatoires, car ses contre-exemples sont toujours de taille minimale (ce qui facilite le déboguage), sa couverture est bien identifiée (puisqu'il constitue une preuve par énumération de tous les cas jusqu'à la borne de test), et une donnée de petite taille suffit souvent pour révéler une erreur [20]. Ainsi, le BET est complémentaire du test aléatoire, plus adapté aux domaines de données de plus grande taille.

Certains outils de test généraux sont capables de générer des données ou de dériver des générateurs à partir de la définition de ces données, selon diverses techniques, détaillées dans les introductions de travaux récents sur ce sujet [21, 7]. Par exemple, Dubois, Giorgetti et Genestier ont complété

QuickChick avec une première approche de test exhaustif borné fondée sur des générateurs dérivés de définitions des données en programmation logique (Prolog) [9].

Cependant, pour certaines structures de données, ces techniques et outils peuvent être trop lents, échouer dans la dérivation d'un générateur ou dériver des générateurs peu efficaces. Il devient alors pertinent de concevoir un générateur dédié à chaque structure, voire de le certifier pour l'intégrer avec confiance dans un outil de test. Par exemple, Bowles et Caminati ont vérifié un algorithme d'énumération de structures d'événements [3]. Dubois et Giorgetti ont développé l'outil CUT, qui ajoute à Coq les commandes SmallCheck et SmallCheckWhy3 de test exhaustif borné avec des programmes d'énumération dédiés, respectivement définis dans les langages de Coq et de Why3 [8]. Afin d'accroître la confiance dans ces programmes, Dubois et Giorgetti proposent de les produire par extraction de programmes WhyML dont la correction et la complétude sont démontrées formellement avec Why3 et Coq [15]. Lorsque ces programmes énumèrent des structures combinatoires, cette preuve formelle devient elle-même une activité de combinatoire formelle, qui complète, voire remplace, une preuve informelle d'un ouvrage de combinatoire énumérative.

5 Jalons

Nous proposons de désigner par *combinatoire certifiée* l'application des méthodes formelles du génie logiciel à la recherche en combinatoire, en particulier pour prouver formellement des théorèmes de combinatoire énumérative ou bijective, ou des propriétés de programmes de comptage, d'énumération ou de transformation bijective de structures combinatoires.

Des collaborations avec des combinatoriciens sont essentielles pour le succès de cette démarche, comme ce fut le cas avec Florent Hivert pour le logiciel SCHUR [4] ou avec Alain Giorgetti sur les permutations et les cartes combinatoires [18, 16, 11, 9, 8, 15]. Ce dernier a apporté à ces travaux sa formation initiale de combinatoricien. S'étant ensuite formé aux méthodes formelles du génie logiciel, il a aussi participé à leur expérimentation pour vérifier formellement des théorèmes de combinatoire [14, 12, 13, 9, 8, 15]. Cette approche formelle a été présentée et défendue lors de recherches récentes avec d'autres combinatoriciens [17, 1, 9], puis lors de "Journées de combinatoire certifiée" organisées au LRI, du 3 au 5 juillet 2019, par Jean-Christophe Filliâtre et Alain Giorgetti. Ces journées ont été financées par la bourse mobilité 2019 du GdR GPL du CNRS d'Alain Giorgetti.

6 Conclusion

Nous pensons que la recherche en combinatoire peut être facilitée par une formalisation machine des problèmes dès le début de leur étude, et accompagnée par une utilisation systématique de méthodes formelles et d'outils de vérification. Dans cette démarche, le test de propriété joue un rôle essentiel. Il permet aux combinatoriciens de construire tranquillement des théorèmes, par exemple dans Coq, et d'être alertés rapidement que leur énoncé est faux, bien avant d'être bloqués dans leur démonstration formelle. Inversement, la combinatoire, comme d'autres domaines de recherche, peut devenir un nouveau champ d'application, d'adaptation et de perfectionnement des méthodes formelles de spécification et de vérification.

Quoique vaste, le sujet de la combinatoire certifiée n'a pas vocation à être l'unique thématique d'un futur GT. Il peut par contre s'intégrer dans un GT sur des thèmes proches ou plus généraux, tels que :

— les nouvelles applications des méthodes et outils du génie logiciel et de la théorie des types : algorithmes et programmes produits par la recherche scientifique actuelle, en particulier dans

le domaine de l'intelligence artificielle ;
— la diffusion des méthodes formelles, par leur enseignement dès la licence ;
— le lien théorique entre raisonnement et calcul, concrétisé par le développement d'interfaces générales et efficaces entre les outils de preuve formelle et les logiciels de calcul formel ;
— la fiabilité et la pérennité des logiciels produits par la recherche en génie logiciel, et la reproductibilité de ses expérimentations.

Références

[1] J.-L. Baril, R. Genestier, A. Giorgetti, and A. Petrossian. Rooted planar maps modulo some patterns. *Discrete Mathematics*, 339 :1199–1205, 2016. Elsevier.

[2] S. Berghofer and T. Nipkow. Random testing in Isabelle/HOL. In *Software Engineering and Formal Methods (SEFM 2004)*, pages 230–239. IEEE Computer Society, 2004.

[3] J. Bowles and M. B. Caminati. A verified algorithm enumerating event structures. In *CICM'17*, volume 10383 of *LNCS*, pages 239–254. Springer, 2017.

[4] F. Butelle, F. Hivert, M. Mayero, and F. Toumazet. Formal proof of SCHUR conjugate function. In *Intelligent Computer Mathematics*, volume 6167 of *LNCS*, pages 158–171. Springer, 2010.

[5] M. Carlier, C. Dubois, and A. Gotlieb. Constraint Reasoning in FOCALTEST. In *International Conference on Software and Data Technologies (ICSOFT 2010)*, Athens, Jul. 2010.

[6] K. Claessen and J. Hughes. QuickCheck : a lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, volume 35 of *SIGPLAN Not.*, pages 268–279, New York, NY, USA, 2000. ACM.

[7] S. Cruanes. Satisfiability modulo bounded checking. In *Automated Deduction – CADE 26*, pages 114–129, Cham, 2017. Springer International Publishing.

[8] C. Dubois and A. Giorgetti. Tests and proofs for custom data generators. *Formal Aspects of Computing*, Jul 2018.

[9] C. Dubois, A. Giorgetti, and R. Genestier. Tests and proofs for enumerative combinatorics. In *TAP'16*, volume 6792 of *LNCS*, pages 57–75. Springer, 2016.

[10] P. Dybjer, Q. Haiyan, and M. Takeyama. Combining testing and proving in dependent type theory. In *TPHOLs 2003*, volume 2758 of *LNCS*, pages 188–203, Heidelberg, 2003. Springer.

[11] R. Genestier and A. Giorgetti. Spécification et vérification formelle d'opérations sur les permutations. In *AFADL'16*, pages 72–78, 2016. `http://events.femto-st.fr/sites/femto-st.fr.gdr-gpl-2016/files/content/AFADL-2016.pdf`.

[12] R. Genestier, A. Giorgetti, and G. Petiot. Gagnez sur tous les tableaux. In *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015)*, 2015. `https://hal.inria.fr/hal-01099135`.

[13] R. Genestier, A. Giorgetti, and G. Petiot. Sequential generation of structured arrays and its deductive verification. In *Tests and Proofs (TAP)*, volume 9154 of *LNCS*, pages 109–128. Springer, 2015.

[14] A. Giorgetti. Guessing a Conjecture in Enumerative Combinatorics and Proving It with a Computer Algebra System. In *SCSS'10*, pages 5–18, July 2010.

[15] A. Giorgetti, C. Dubois, and R. Lazarini. Combinatoire formelle avec Why3 et Coq. In N. Magaud and Z. Dargaye, editors, *Journées Francophones des Langages Applicatifs 2019*, pages 139–154, Les Rousses, France, Jan. 2019. publié par les auteurs. `https://hal.inria.fr/hal-01985195`.

[16] A. Giorgetti, R. Genestier, and V. Senni. Software engineering and enumerative combinatorics. Institut Henri Poincaré, Paris, France, May 2014.

[17] A. Giorgetti and V. Senni. Specification and validation of algorithms generating planar Lehman words. In *GASCom'12, 8-th Int. Conf. on random generation of combinatorial structures*, Bordeaux, France, June 2012. `https://hal.inria.fr/hal-00753008`.

[18] A. Giorgetti and V. Senni. Combining tests and proofs to check combinatorial generation algorithms. Séminaire invité dans l'équipe CPR du laboratoire CEDRIC du CNAM, Paris, may 2013.

[19] G. Gonthier. The four colour theorem : Engineering of a formal proof. In *ASCM 2007*, volume 5081 of *LNCS (LNAI)*, pages 333–333. Springer, Heidelberg, 2008.

[20] D. Jackson and C. Damon. Elements of style : Analyzing a software design feature with a counterexample detector. *IEEE Trans. Softw. Eng.*, 22(7) :484–495, 1996.

[21] L. Lampropoulos, D. Gallois-Wong, C. Hriţcu, J. Hughes, B. C. Pierce, and L. Xia. Beginner's luck : a language for property-based generators. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 114–129. ACM, 2017.

[22] S. Owre. Random testing in PVS. In *Workshop on Automated Formal Methods (AFM)*, 2006.

[23] Z. Paraskevopoulou, C. Hriţcu, M. Dénès, L. Lampropoulos, and B. C. Pierce. Foundational property-based testing. In *ITP 2015*, volume 9236 of *LNCS*, pages 325–343, Heidelberg, 2015. Springer.