Vérification formelle de programmes de génération de données structurées

Richard Genestier

Vérification formelle de programmes de génération de données structurées

Résumé des travaux de thèse de doctorat

Richard Genestier
FEMTO-ST UMR CNRS 6174 (UBFC/UFC/ENSMM/UTBM)
Université de Franche-Comté, France
richard.genestier@femto-st.fr

Résumé

Ce document résume les travaux de recherche de ma thèse d'informatique intitulée "Vérification formelle de programmes de génération de données structurées", dirigée depuis septembre 2012 par O. Kouchnarenko, professeur et co-encadrée par A. Giorgetti, maître de conférences.

Les méthodes du génie logiciel, telles que la spécification formelle et la preuve déductive, rendent la conception de programmes plus rationnelle, augmentent leur qualité et accroissent la confiance du développeur dans la correction de sa production. En raison de leur fort coût, ces méthodes n'ont longtemps été appliquées que dans des domaines à fort enjeu de sûreté ou de sécurité. De nombreux progrès récents rendent ces pratiques vertueuses accessibles à d'autres domaines du développement logiciel.

Par différentes études de cas, ce travail de thèse évalue la faisabilité de démonstrations automatiques de programmes manipulant des tableaux d'entiers. Le cadre d'investigation choisi est le domaine de la combinatoire énumérative car il fournit de nombreux algorithmes dont la difficulté et la structure sont bien adaptées aux capacités des prouveurs actuels. Dans la combinatoire, les efforts de formalisation et de démonstration se sont concentrés sur certains objets, appelés *cartes*.

La combinatoire est la partie des mathématiques discrètes qui étudie des ensembles finis ou dénombrables d'objets appelés *structures combinatoires*. Les permutations de n objets distinguables et les permutations particulières, telles que les involutions, sont des exemples de structures combinatoires. La combinatoire énumérative propose des algorithmes pour générer ces structures et pour compter le nombre de structures composées d'un nombre donné d'éléments. Le lecteur désirant davantage de précisions peut par exemple se référer à [Sta97].

Les cartes sont des objets mathématiques qui apparaissent naturellement dans l'étude de nombreux problèmes, de nature mathématique ou physique. Une *carte (combinatoire) étiquetée* est un triplet (D, R, L) où D est un ensemble fini à $2n$ éléments (pour un entier naturel $n \geq 0$), appelés *étiquettes* ou *brins*, R est une permutation sur D et L est une involution sans point fixe sur D, telles que le groupe $\langle R, L\rangle$ engendré par R et L agit transitivement sur D [LZ04]. L'action transitive des permutations R et L correspond au fait que deux brins quelconques d'une carte peuvent être envoyés l'un sur l'autre par un nombre fini d'applications de R et L. Les combinatoriciens s'intéressent plus particulièrement aux cartes enracinées. Combinatorialement, une *carte enracinée* est une classe d'équivalence de cartes étiquetées isomorphes par les isomorphismes renommant leurs brins mais préservant l'un d'entre eux, appelé *racine*.

Dans cette thèse, la spécification formelle, la validation par test et la vérification déductive sont appliquées à des implémentations d'algorithmes connus de combinatoire, mais également à des implémentations de nouveaux algorithmes. L'accent est mis sur les algorithmes qui facilitent la génération ou le dénombrement des cartes.

Les travaux de vérification déductive concernant le domaine des cartes combinatoires sont peu nombreux. Dans [DM07], C. Dubois et J.-M. Mota proposent une formalisation des cartes générales utilisant le formalisme B, très proche de leur définition mathématique à base de permutations et d'involutions. J.-F. Dufourd et F. Puitg ont proposé une spécification en ML des cartes combinatoires pointées [DP00]. En utilisant des hypercartes combinatoires, J.-F. Dufourd a établi en Coq [Coq] une preuve formelle du théorème de Jordan [Duf09]. On trouve également une formalisation avancée des hypercartes combinatoires dans le vaste projet aboutissant à la preuve formelle du théorème des quatre couleurs en Coq [Gon05].

L'objectif d'effectuer des preuves formelles automatiques sur des objets complexes comme les cartes enracinées est assez ambitieux. Une approche par étapes a donc été adoptée, d'une part pour en évaluer sa faisabilité, et d'autre part pour identifier les limites des outils de vérification automatique utilisés. Les algorithmes de génération présentés sont programmés en C et spécifiés en ANSI C Specification Language (ACSL) [BCF+13]. ACSL est un langage dédié à l'analyse statique de programmes C qui permet de spécifier formellement des contrats qui doivent être vérifiés par le programme. L'outil principal utilisé est la plateforme d'analyse de programmes C Frama-C [KKP+15] développée par le CEA LIST et INRIA Saclay, qui utilise le langage de spécification ACSL. Pour la vérification déductive, Frama-C utilise le greffon WP, qui implante le calcul de plus faible précondition pour des programmes C annotés en ACSL, et les solveurs SMT Alt-Ergo [Alt], CVC3 [BT07] et CVC4 [CVC]. Ces solveurs sont utilisés par WP pour déterminer si une formule du premier ordre est satisfaisable.

Les cartes combinatoires étant définies à partir de permutations, il paraissait intéressant de considérer la génération des permutations et de diverses structures combinatoires proches, encodables en C par un tableau d'entiers dit *structuré*, car satisfaisant une contrainte structurelle exprimée en logique du premier ordre. Avec A. Giorgetti et G. Petiot, nous avons porté notre attention sur les algorithmes d'énumération combinatoire qui génèrent séquentiellement toutes les tableaux structurés de même longueur, selon un ordre total prédéfini. Un tel algorithme est nommé *générateur séquentiel* lorsqu'il est composé de deux fonctions : la première fonction construit la plus petite structure de longueur donnée selon l'ordre prédéfini, et la seconde fonction modifie une structure quelconque pour construire la structure suivante de même longueur selon cet ordre. Nous présentons une approche uniforme pour la mise en œuvre rationnelle de ces générateurs séquentiels de tableaux structurés, implantés par des fonctions C. Nous considérons trois propriétés comportementales pour ces fonctions de génération. La propriété de *correction* affirme que les deux fonctions génèrent des tableaux satisfaisant leur contrainte structurelle. La propriété de *progression* affirme que la deuxième fonction génère un tableau supérieur à son tableau d'entrée selon l'ordre prédéfini. Cette propriété implique la terminaison des appels répétés à la seconde fonction. La propriété d'*exhaustivité* affirme que le générateur n'omet aucune solution. Les propriétés de correction et de progression sont vérifiées statiquement. Dans ce but, nous spécifions formellement le comportement des fonctions de génération de tableaux. Nous les annotons ensuite avec des invariants et variants de boucle de manière à ce que leurs contrats formels soient automatiquement prouvés. La propriété d'exhaustivité se traduit par le fait qu'il n'existe pas de tableau structuré entre deux tableaux structurés successifs générés. Nous n'avons pas identifié de fragment décidable de la théorie des tableaux incluant la condition de vérification de cette propriété, qui comporte une quantification (existentielle) sur un tableau. De fait, elle n'est pas déchargée par les prouveurs automatiques actuels. En conséquence, nous validons la propriété d'exhaustivité par comptage grâce à une génération exhaustive bornée des structures. Cette approche a débouché sur

la conception de patrons généraux d'aide à l'écriture de ces programmes spécifiés, de générateurs séquentiels génériques de tableaux structurés, et d'une bibliothèque de générateurs séquentiels vérifiés [1]. Ce travail a fait l'objet de deux publications, correspondant à deux stades successifs de son avancement [GGP15a, GGP15b].

Il résulte de cette approche un certain nombre de constats. Afin de faciliter la démonstration automatique de la correction de ces fonctions C, ces dernières doivent utiliser un petit fragment du langage C, et leurs spécifications doivent pouvoir être exprimées en logique du premier ordre. Plus précisément, il convient de considérer au sein de ces fonctions des tableaux d'entiers alloués préalablement (pas de pointeur ni d'allocation dynamique), et uniquement des expressions d'arithmétique linéaire sur les entiers. Ensuite, les difficultés dans l'établissement des preuves sont liées à la correction et à la précision des invariants de boucles, qui ne doivent être ni trop faibles (ils ne permettent pas de prouver les postconditions attendues), ni trop forts (les prouveurs peinent alors à prouver leur préservation). De manière générale, lorsque la preuve de la correction d'une fonction présente une difficulté, ajouter des assertions ou décomposer cette fonction en sous-fonctions munies de contrats contenant des postconditions intermédiaires menant progressivement à l'établissement des postconditions finales aide grandement les prouveurs dans leur tâche.

Les cartes étiquetées étant des couples de permutations agissant transitivement sur un ensemble fini D, nous avons complété cette bibliothèque avec de nouveaux générateurs génériques de couples de structures transitives, puis nous les avons instanciés pour obtenir divers générateurs séquentiels de familles de cartes étiquetées. Cela permet de visualiser ces objets et de valider des conjectures sur ces cartes, par test exhaustif borné.

En complément de cette étude des cartes étiquetées, les cartes enracinées (non étiquetées) ont été considérées à travers l'une de leurs premières représentations par des mots. Avec J.-L. Baril, A. Giorgetti et A. Petrossian, nous nous sommes intéressés plus particulièrement aux cartes planaires, qui sont des cartes qu'on peut représenter par un graphe dessiné sur une sphère. Une carte planaire enracinée (non étiquetée) à $2n$ brins peut être représentée par un mot à $2n$ lettres issues d'un alphabet à 4 lettres. Pour tout mot π sur cet alphabet, appelé *motif*, nous définissons sur ces mots la relation d'équivalence suivante : deux mots de même longueur sont dits π*-équivalents* si et seulement s'ils contiennent le motif π aux mêmes positions. Pour quasiment tous les motifs de longueur au plus deux, nous avons déterminé dans [BGGP16] la fonction génératrice du nombre de classes de π-équivalence. La bijection entre mots et cartes planaires enracinées donne ainsi des résultats d'énumération pour les classes d'équivalence de cartes planaires enracinées. En suivant la méthodologie présentée dans [GGP15a, GGP15b], nous avons mis au point un générateur séquentiel de ces mots. Ce générateur a été utilisé pour compter le nombre de classes de π-équivalence des mots de petite longueur. Cette expérimentation a grandement facilité l'établissement des résultats de [BGGP16].

Avec A. Giorgetti et C. Dubois, nous nous sommes également intéressés à d'autres modes de génération, notamment des générations arborescentes, basées sur des constructions inductives de cartes combinatoires enracinées. Dans ce cadre, le choix d'une représentation formelle des cartes est primordial. En effet, selon ce choix, l'implémentation et la vérification des constructions inductives seront plus ou moins aisées et efficaces. Nous proposons à cet effet la notion de carte locale, codée par une seule permutation, ainsi que deux opérations sur les permutations adaptées au mode de construction inductive adopté. Nous avons formalisé cette notion, ces opérations et leur correction en Coq. Cette correction est validée par test exhaustif borné et aléatoire, avant d'être prouvée interactivement en Coq [DGG16].

1. Archive `enum.*.tar.gz` téléchargeable depuis la page `http://members.femto-st.fr/richard-genestier/en`.

Les deux opérations sur les permutations consistent à insérer un élément dans une permutation et à fusionner deux permutations. Nous les avons implémentées en C, nous avons spécifié leur correction en ACSL et avons prouvé cette correction automatiquement [GG16] avec Frama-C et son greffon WP.

Remerciements. L'auteur remercie O. Kouchnarenko, A. Giorgetti et les relecteurs pour leurs suggestions.

Références

[Alt] The Alt-Ergo SMT solver. `http://alt-ergo.lri.fr`.

[BCF+13] P. Baudin, P. Cuoq, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. *ACSL : ANSI/ISO C Specification Language*, 2013. `http://frama-c.com/acsl.html`.

[BGGP16] J.-L. Baril, R. Genestier, A. Giorgetti, and A. Petrossian. Rooted planar maps modulo some patterns. *Discrete Mathematics*, 339 :1199–1205, 2016.

[BT07] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *LNCS*, pages 298–302. Springer, 2007.

[Coq] The Coq Proof Assistant. `http://coq.inria.fr`.

[CVC] CVC4. `http://cvc4.cs.nyu.edu/web/`.

[DGG16] C. Dubois, A. Giorgetti, and R. Genestier. Tests and proofs for enumerative combinatorics. In *Tests and Proofs (TAP)*, 2016. To appear.

[DM07] C. Dubois and J.-M. Mota. Geometric modeling with B : formal specification of generalized maps. *Journal of Scientific & Practical Computing Journal*, 1(2) :9–24, 2007.

[DP00] J.-F. Dufourd and F. Puitg. Functional specification and prototyping with oriented combinatorial maps. *Computational Geometry*, 16(2) :129 – 156, 2000.

[Duf09] J.-F. Dufourd. An intuitionistic proof of a discrete form of the Jordan curve theorem formalized in Coq with combinatorial hypermaps. *J. Autom. Reason.*, 43(1) :19–51, June 2009.

[GG16] R. Genestier and A. Giorgetti. Spécification et vérification formelle d'opérations sur les permutations. In *Approches Formelles dans l'Assistance au Développement Logiciel (AFADL)*, 2016.

[GGP15a] R. Genestier, A. Giorgetti, and G. Petiot. Gagnez sur tous les tableaux. In *Journées Francophones des Langages Applicatifs (JFLA)*, 2015. `https://hal.inria.fr/hal-01099135`.

[GGP15b] 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, Heidelberg, 2015.

[Gon05] G. Gonthier. A computer checked proof of the Four Colour Theorem, 2005. `http://research.microsoft.com/gonthier/4colproof.pdf`.

[KKP+15] F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C : a Software Analysis Perspective. *Formal Aspects of Computing*, 27(3) :573–609, 2015.

[LZ04] S. K. Lando and A. K. Zvonkin. *Graphs on Surfaces and Their Applications*. Springer, 2004.

[Sta97] R. P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, Cambridge, England, 1997.