Test et preuve pour des structures combinatoires : Coq et Prolog

Catherine Dubois[1], Alain Giorgetti[2], and Richard Genestier[2]

[1] Samovar (UMR CNRS 5157), ENSIIE, Évry, France
catherine.dubois@ensiie.fr
[2] FEMTO-ST institute (UMR CNRS 6174 - UBFC/UFC/ENSMM/UTBM)
Université de Franche-Comté, Besançon, France
alain.giorgetti@femto-st.fr, richard.genestier@femto-st.fr

Faire une preuve interactivement à l'aide d'un assistant à la preuve - quiconque s'y est essayé le dira - n'est pas chose facile. Mais le plus frustrant est sans doute d'essayer de faire une preuve d'un lemme incorrect. Il est donc intéressant de pouvoir *tester* ces conjectures avant de les prouver. Des outils plus ou moins aboutis traitent cette question pour la plupart des assistants à la preuve (Isabelle [1], Agda [4], PVS [5], FoCaLiZe [2] et plus récemment Coq [6]). Ces outils sont très souvent inspirés de QuickCheck [3]. Ils permettent d'acquérir une certaine confiance dans le lemme testé et dans les définitions utilisées dans son énoncé ou, en cas d'échec, d'obtenir un ou plusieurs contre-exemples. Dans le cadre de l'étude de structures combinatoires comme les cartes combinatoires [9] ou les λ-termes [7], les objectifs principaux concernent le comptage de ces structures et la mise en place de bijections entre différentes familles de structures. Dans ce cadre, il est souvent utile d'énumérer les structures jusqu'à une certaine taille et d'utiliser ces éléments pour tester une certaine propriété.

Nous proposons une méthodologie alliant test aléatoire et test exhaustif borné pour tester des propriétés écrites en Coq et portant sur des structures combinatoires. Plus précisément, nous utilisons conjointement le plugin QuickChick de Coq et Prolog (ainsi que la bibliothèque Prolog de validation développée par V. Senni) pour réaliser cette combinaison. La méthodologie est exposée sur deux exemples : les permutations et les cartes combinatoires enracinées.

Méthodologie et outils utilisés

Test aléatoire. QuickChick [3] est un plugin de test développé pour Coq [6]. Il permet de tester la validité de propriétés exécutables avec des données générées aléatoirement. QuickChick fournit différents combinateurs pour écrire des générateurs aléatoires et le code dédié au test. La propriété sous test doit être exécutable, ce qui demande en général de transformer un prédicat en une fonction booléenne équivalente. Dans certains cas, il est possible, pour ce faire, d'utiliser le plugin Relation Extraction [8].

Test exhaustif borné. Nous proposons d'utiliser Prolog pour énumérer les structures combinatoires jusqu'à une certaine taille, ce qui est en général très

3. https://github.com/QuickChick

facile à obtenir grâce au mécanisme de *backtracking* de Prolog. Chaque solution proposée par Prolog est traduite en un objet Coq avec lequel des lemmes Coq ou leur version exécutable sont instanciés. Dans le cas non exécutable, des tactiques appropriées peuvent être appliquées pour démontrer chaque instance des lemmes.

Cas d'étude

Nous illustrons ces différents modes de validation avec la mise au point de spécifications Coq pour les structures combinatoires des permutations et des cartes enracinées. Une permutation est définie comme une fonction injective sur un intervalle d'entiers naturels. Une telle permutation est isomorphe à une liste sans doublons contenant les éléments de l'intervalle de définition. Nous définissons ensuite la somme directe de deux permutations ainsi qu'une opération d'insertion. Dans un premier temps, l'objectif est de tester (puis démontrer) que ces deux opérations construisent bien des permutations lorsqu'elles sont appliquées à des permutations. Nous nous intéressons ensuite au cas des cartes combinatoires enracinées définies comme des paires transitives de permutations. Deux opérations spécifiques d'ajout d'une arête sont ensuite définies. Elles permettent de construire des cartes à partir de cartes plus petites. Ici la propriété que nous cherchons à valider concerne la préservation de la transitivité.

Références

1. Berghofer, S., Nipkow, T. : Random testing in Isabelle/HOL. In : Cuellar, J., Liu, Z. (eds.) Software Engineering and Formal Methods (SEFM 2004). pp. 230–239. IEEE Computer Society (2004)
2. Carlier, M., Dubois, C., Gotlieb, A. : Constraint Reasoning in FOCALTEST. In : Int. Conf. on Soft. and Data Tech. (ICSOFT'10). Athens (Jul 2010)
3. Claessen, K., Hughes, J. : QuickCheck : a lightweight tool for random testing of Haskell programs. In : Proceedings of Int. Conf. on Functional Programming (ICFP 2000). SIGPLAN Not., vol. 35, pp. 268–279. ACM, New York, NY, USA (2000)
4. Dybjer, P., Haiyan, Q., Takeyama, M. : Combining testing and proving in dependent type theory. In : Basin, D., Wolff, B. (eds.) Proceedings of Theorem Proving in Higher Order Logics. LNCS, vol. 2758, pp. 188–203. Springer (2003)
5. Owre, S. : Random testing in PVS. In : Workshop on Automated Formal Methods (AFM) (2006)
6. Paraskevopoulou, Z., Hritcu, C., Dénès, M., Lampropoulos, L., Pierce, B.C. : Foundational property-based testing. In : Interactive Theorem Proving - ITP 2015, Nanjing, China. LNCS, vol. 9236, pp. 325–343. Springer (2015)
7. Tarau, P. : Ranking/unranking of lambda terms with compressed de bruijn indices. In : Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA. LNCS, vol. 9150, pp. 118–133. Springer (2015)
8. Tollitte, P., Delahaye, D., Dubois, C. : Producing certified functional code from inductive specifications. In : Certified Programs and Proofs, CPP 2012, Kyoto, Japan, 2012. LNCS, vol. 7679, pp. 76–91. Springer (2012)
9. Tutte, W.T. : On the enumeration of planar maps. Bull. Amer. Math. Soc. 74, 64–74 (1968)