Réutiliser les spécifications et preuves

Gilles Dowek, Catherine Dubois

8 décembre 2014

1 Contexte

On constate que l'utilisation de méthodes déductives pour produire des logiciels sûrs progresse mais reste encore à diffusion limitée. Elle progresse car de nombreux formalismes, langages et outils sont proposés mais elle reste encore trop limitée et réservée à certaines niches. L'investissement est souvent jugé lourd, en particulier pour les nouveaux domaines, et le passage à l'échelle reste difficile. On constate également qu'il manque de passerelles ente les outils de preuve (qu'ils soient automatiques ou non) même si la coopération entre outils a fait des progrès récents.

2 Problématique

Une façon de progresser est de pouvoir réutiliser les spécifications et preuves formelles d'un contexte à un autre, mais aussi d'un outil à un autre.

3 Challenges identifiés

Les challenges identifiés concernent la proposition de techniques de réutilisation intra (*in the small*) et inter-outils (*in the large*).

La plupart des assistants à la preuve offre des techniques architecturales, pour aider à la réutilisation : modules, type classes, paramètres, foncteurs, contextes, héritage, redéfinition. La preuve est en général modulaire. Néanmoins ces techniques, efficaces dans certaines situations, conduisent dans d'autres cas à des formulations peu naturelles (par exemple il faut introduire trop de paramètres) et demandent parfois de reprendre le code (spécifications et/ou preuves). Le challenge identifié ici est de proposer des techniques de réutilisation *in the small* qui permettent une réutilisation plus sémantique et mettent en œuvre des techniques d'adaptation des spécifications et preuves formelles.

Le deuxième challenge concerne la réutilisation *in the large*, c'est-à-dire la possibilité de réutiliser un développement formel (spécifications et preuves) réalisé dans un formalisme donné dans un autre développement réalisé dans un autre formalisme. Quelques tentatives existent entre des systèmes de la même famille (par exemple, Open Theory pour les assistants de la famille HOL [4]) ; des traductions pair à pair ont également été proposées (par exemple HOL Light vers Coq [5], SMT vers Coq [1]). Il s'agit ici de proposer un standard pour les spécifications et preuves, comme il existe des standards dans d'autres domaines (réseaux, web, etc).

4 Jalons

Pour la réutilisation *in the small*, il est intéressant d'étudier les techniques provenant du monde de la programmation orientée objet et des lignes de produits. Des travaux embryonnaires existent dans cette voie : introduction de l'héritage et de la redéfinition dans Focalize, utilisation des techniques issues des lignes de produit pour définir la sémantique des langages de programmation (Meta-theory à la carte) [3].

Nous proposons de lever le verrou de la réutilisation *in the large*, en mettant en place un formalisme qui permettrait d'exprimer les spécifications et les preuves provenant de formalismes divers (logique du premier ordre, théorie des types, théorie des ensembles, logique d'ordre supérieur par exemple). Il s'agirait ensuite de fournir des outils de traduction des formalismes existants vers ce standard de formalisation. Une première piste concerne l'étude du $\lambda\Pi$-calcul modulo, λ-calcul typé avec types dépendants paramétré par un système de réécriture. Ce formalisme, utilisé avec un système de réécriture particulier, peut encoder d'autres formalismes [2]. Il existe actuellement des outils de traduction : Coq vers $\lambda\Pi$-calcul modulo, HOL vers $\lambda\Pi$-calcul modulo, Focalize vers $\lambda\Pi$-calcul modulo (voir `http://dedukti.gforge.inria.fr/`). Un premier jalon est d'étudier le $\lambda\Pi$-calcul modulo, de l'étendre, de l'évaluer comme standard de formalisation. De nombreux problèmes théoriques sont liés à cette approche, en particulier des problèmes de cohérence, de confluence, de terminaison des systèmes de réécriture nécessaires. Le chemin est cependant long jusqu'à la proposition d'un standard reconnu par la communauté.

Références

[1] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of SAT/SMT solvers to coq through proof witnesses. In *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2011.

[2] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In *TLCA*, pages 102–117, 2007.

[3] Benjamin Delaware, Bruno C. d. S. Oliveira, and Tom Schrijvers. Meta-theory à la carte. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 207–218. ACM, 2013.

[4] Joe Hurd. The opentheory standard theory library. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, volume 6617 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2011.

[5] Chantal Keller and Benjamin Werner. Importing HOL Light into Coq. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP - Interactive Theorem Proving, First International Conference - 2010*, volume 6172 of *Lecture Notes in Computer Science*, pages 307–322, Edimbourg, Royaume-Uni, 2010. Springer.