Proposition de sujet de thèse 2013–2016

Obtention de programmes corrects par raffinement dans un langage de haut niveau

Correctness-by-refinement in a high-level programming language

Lieu de la thèse :
Équipe Toccata (LRI / Inria Saclay)
bât. 650 Université Paris Sud
91405 Orsay Cedex
`http://toccata.lri.fr/`

Période :
septembre 2013 – août 2016

Directeurs de thèse :
Jean-Christophe Filliâtre (CNRS)
Andrei Paskevich (Université Paris Sud)

Sujet

La vérification déductive de programmes consiste à ramener la correction d'un programme informatique à un ensemble d'énoncés mathématiques [9]. Si l'idée n'est pas neuve — elle date de la fin des années 60 — elle connaît aujourd'hui un essor important, grâce notamment aux progrès fulgurants de la démonstration automatique. Parmi les nombreux systèmes de vérification déductive existants, on peut citer entre autres KeY, SPEC#, VCC, Dafny, Frama-C, Krakatoa, VeriFast, GNATprove [4, 8, 12, 3, 13, 11, 7]. Ces systèmes visent à la vérification de programmes écrits dans des langages de programmation existants, tels que C, Java, C# ou Ada. Ils s'appuient parfois sur des langages intermédiaires dédiés à la vérification déductive, tels que Boogie [2] ou Why3 [10].

Une autre approche de la vérification déductive consiste à obtenir des programmes *corrects par construction*. Ainsi l'assistant de preuve Coq [16] propose un mécanisme d'extraction de code OCaml et Haskell à partir de preuves constructives. Des systèmes comme la méthode B [1] ou KIV [15] mettent en œuvre la technique par raffinements successifs d'une spécification haut niveau jusqu'à un code exécutable. Toutes les étapes de raffinement se font dans un unique langage, qui est à la fois un langage de spécification et de programmation. Une fois le code exécutable atteint, il peut être traduit dans un langage de programmation existant (typiquement C, C++ ou Ada).

Cette thèse propose d'explorer l'idée de la construction de programmes par raffinement en réutilisant un grand nombre de concepts issus de la programmation fonctionnelle : polymorphisme, types algébriques, filtrage, types abstraits, ordre supérieur, structures mutables — on entend ici « programmation fonctionnelle » au sens large, c'est-à-dire ML et non pas « programmation purement applicative ». Un certain nombre de ces concepts sont déjà à l'œuvre dans l'outil de vérification déductive Why3 [6] et son langage de programmation WhyML [10], mais ce dernier ne propose pour l'instant aucune notion d'encapsulation ou de raffinement. Plus précisément, les objectifs de cette thèse sont les suivants :

1. Rapprocher les langages de spécification et de programmation de Why3, notamment pour permettre ensuite un raffinement graduel de spécifications en des programmes exécutables.
2. Concevoir un système de modules pour le langage WhyML, permettant à la fois l'encapsulation et le raffinement.
3. Identifier le fragment exécutable de WhyML et proposer une traduction vers un langage existant. On pourra commencer par le langage OCaml, dont WhyML est proche, mais aussi étudier la traduction vers un langage comme C (éventuellement en se limitant à un fragment de WhyML).
4. Réaliser un prototype, par extension de l'outil Why3 [5]. En particulier, on conservera les capacités de l'outil Why3 à décharger les obligations de preuve à l'aide d'un grand nombre de démonstrateurs existants.
5. Évaluer expérimentalement cet outil, notamment en réutilisant des exemples de la méthode B. Ces exemples pourront être traités de deux façons différents : soit en utilisant une formalisation de la théorie des ensembles de B (par exemple reprise des travaux traduisant les obligations de preuve B vers Why3 [14]), soit en réécrivant ces exemples de manière idiomatique dans le langage WhyML, notamment pour mesurer l'impact du système de types de WhyML par rapport à la théorie des ensembles.

Les enjeux scientifiques de cette thèse se situent dans la combinaison, inédite à ce jour, de la théorie du raffinement et des systèmes de types riches avec polymorphisme, ordre supérieur et effets. Un défi particulier est celui du raffinement de structures de données complexes contenant notamment des champs fantômes ou mutables et portant des invariants.

Références

[1] Jean-Raymond Abrial. *The B-Book, assigning programs to meaning.* Cambridge University Press, 1996.

[2] Mike Barnett, Robert DeLine, Bart Jacobs, Bor-Yuh Evan Chang, and K. Rustan M. Leino. Boogie : A Modular Reusable Verifier for Object-Oriented Programs. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects : 4th International Symposium*, volume 4111 of *Lecture Notes in Computer Science*, pages 364–387, 2005.

[3] Patrick Baudin, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto. *ACSL : ANSI/ISO C Specification Language, version 1.4*, 2009. `http://frama-c.cea.fr/acsl.html`.

[4] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software : The KeY Approach*, volume 4334 of *Lecture Notes in Computer Science*. Springer, 2007.

[5] François Bobot, Jean-Christophe Filliâtre, Claude Marché, Guillaume Melquiond, and Andrei Paskevich. *The Why3 platform, version 0.80.* LRI, CNRS & Univ. Paris-Sud & INRIA Saclay, version 0.80 edition, October 2012. `https://gforge.inria.fr/docman/view.php/2990/8186/manual-0.80.pdf`.

[6] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3 : Shepherd your herd of provers. In *Boogie 2011 : First International Workshop on Intermediate Verification Languages*, pages 53–64, Wrocław, Poland, August 2011.

[7] Cyrille Comar, Johannes Kanig, and Yannick Moy. Integrating formal program verification with testing. In *Proceedings of the Embedded Real Time Software and Systems conference*, ERTS^2 2012, February 2012.

[8] Markus Dahlweid, Michal Moskal, Thomas Santen, Stephan Tobies, and Wolfram Schulte. VCC : Contract-based modular verification of concurrent C. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Companion Volume*, pages 429–430. IEEE Comp. Soc. Press, 2009.

[9] Jean-Christophe Filliâtre. Deductive software verification. *International Journal on Software Tools for Technology Transfer (STTT)*, 13(5) :397–403, August 2011.

[10] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 — where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Proceedings of the 22nd European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, March 2013.

[11] Bart Jacobs and Frank Piessens. The VeriFast program verifier. CW Reports CW520, Department of Computer Science, K.U.Leuven, August 2008.

[12] K. Rustan M. Leino. Dafny : An Automatic Program Verifier for Functional Correctness. In Springer, editor, *LPAR-16*, volume 6355, pages 348–370, 2010.

[13] Claude Marché. The Krakatoa tool for deductive verification of Java programs. Winter School on Object-Oriented Verification, Viinistu, Estonia, January 2009. `http://krakatoa.lri.fr/ws/`.

[14] David Mentré, Claude Marché, Jean-Christophe Filliâtre, and Masashi Asuka. Discharging proof obligations from Atelier B using multiple automated provers. In Steve Reeves and Elvinia Riccobene, editors, *ABZ'2012 - 3rd International Conference on Abstract State Machines, Alloy, B and Z*, volume 7316 of *Lecture Notes in Computer Science*, pages 238–251, Pisa, Italy, June 2012. Springer. `http://hal.inria.fr/hal-00681781/en/`.

[15] W. Reif, G. Schnellhorn, and K. Stenzel. Proving system correctness with KIV. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 859–862, Lille, France, April 1997. Springer.

[16] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.3*, 2010. `http://coq.inria.fr`.