Proposition de Sujet de Thèse

Un langage unique pour à la fois développer des programmes et les prouver

Lieu

Équipe VALS du LRI & Équipe Toccata de l'Inria Saclay
http://toccata.lri.fr/
Batiment 650
Rue Noetzlin
Université Paris-Sud
Orsay, France

Période

De septembre 2013 à août 2016

Directeur de thèse

Claude Marché
Tél : 01 72 92 59 69
Email : Claude.Marche@inria.fr

Contexte de la thèse

La preuve de programme (encore appelée *vérification déductive*) est une idée ancienne, remontant à la fin des années 1960. Elle repose sur le concept de *contrat* : une spécification du comportement attendu d'un programme est exprimée dans un langage formel, par exemple la logique du premier ordre. On peut alors ramener la preuve de la correction d'un programme, vis-à-vis d'une telle spécification, à la preuve de certaines formules logiques, appelées *obligations de preuves*. C'est seulement depuis la fin des années 1990 que cette approche a commencé à être utilisable en pratique, grâce aux progrès simultanés des performances des ordinateurs d'une part et des outils de démonstration automatique d'autre part. C'est ainsi qu'une partie du code critique embarqué de la ligne 14 du métro parisien a été développée et prouvée dans l'Atelier B. D'autres environnements de preuve ont vu le jour au cours des années 2000 : KeY [6], Spec# [4], VCC [10], Dafny [14], Frama-C [11], VeriFast [13], GNATprove [9], etc. L'utilisation croissante de tels outils, en particulier dans un contexte industriel, soulève un problème nouveau : la confiance que l'on peut accorder aux outils eux-mêmes.

Une direction de recherche qui prend de l'importance de nos jours concerne donc la vérification de tels outils, par des techniques déductives. La nature de ces outils, qui effectuent essentiellement des calculs symboliques, au lieu des codes embarqués qui font essentiellement du calcul numérique, demande de concevoir de nouvelles approches. Des projets récents ont avancé dans cette direction. Ainsi, le projet CompCert [15] propose un compilateur certifié, dont la preuve est basée sur une

formalisation de la sémantique du langage C et de l'assembleur. De telles formalisations exigent des langages de spécification significativement plus expressifs que la logique du premier ordre : par exemple CompCert est développé et prouvé au sein de l'environnement Coq [7]. Si l'environnement Coq possède un langage logique très riche, il fournit par contre un faible degré d'automatisation des preuves. De plus, son langage de programmation a des limitations qui rendent son utilisation laborieuse : les effets de bord sont interdits, toutes les fonctions doivent être définies de façon totale (pas de levée d'exception, et ces fonctions doivent être prouvées terminantes pour tous paramètres d'entrée).

Objectifs de la thèse

L'environnement Why3 (`http://why3.lri.fr` [8]) développé dans l'équipe Toccata permet de spécifier des programmes dans une logique qui se veut intermédiaire entre la logique du premier ordre et les logiques riches comme celle de Coq. Why3 permet d'appeler de nombreux prouveurs automatiques, offrant ainsi un haut niveau d'automatisation. Le langage de programmation de Why3 est proche d'Objective Caml, il permet les effets de bord, les exceptions en particulier. Why3 est à l'heure actuelle utilisé par plusieurs autres outils de vérification (GnatProve, EasyCrypt, etc) mais n'est pas utilisé pour directement produire du code exécutable.

L'objectif général de cette thèse est ainsi d'étudier la possibilité d'utiliser Why3 comme un outil de programmation, permettant de développer des programmes, de les compiler vers des exécutables, tout en permettant simultanément de les prouver. Le travail d'étude bibliographique s'appuiera sur quelques projets récents qui poursuivent un objectif similaire : Plaid [1] [2, 1], Trellys [2], ATS [3], Guru [18], Fstar [4]. On peut énumérer quelques défis qu'il faudra résoudre :

- La spécification des outils eux-mêmes peut nécessiter d'étendre la logique de Why3, par exemple vers plus d'ordre supérieur. Il faudra alors envisager de mettre en place des techniques d'automatisation des preuves, par exemple par transformation.
- Le mécanisme de production d'un véritable code exécutable à partir d'une programme Why3 est actuellement dans un état embryonnaire (par traduction vers OCaml) est devra être considérablement enrichi. L'obtention d'un code final exécutable qui soit non seulement prouvé correct mais raisonnablement efficace est un objectif important de cette thèse.
- L'objectif de prouver un outil d'analyse de façon complète est très ambitieux. On étudiera les méthodes permettant de développer des analyses avec des *validateurs* [5], qui permettent de ne devoir prouver que ces validateurs. On étudiera par ailleurs la combinaison des vérifications par preuve et par tests [9].
- Des études de cas significatives devront être développées pour valider l'approche proposée. On envisagera d'une part le développement d'un prouveur automatique certifié, en s'inspirant des travaux récents [16, 3] ; d'autre part le développement d'un générateur certifié d'obligations de preuve [12, 17].

Références

[1] Jonathan Aldrich. Resource-based programming in Plaid. Fun Ideas and Thoughts, June 2010.

1. `http://www.cs.cmu.edu/~aldrich/plaid.html`
2. `http://code.google.com/p/trellys/`
3. `http://www.ats-lang.org/`
4. `http://research.microsoft.com/en-us/projects/fstar/`

[2] Jonathan Aldrich, Joshua Sunshine, Darpan Saini, and Zachary Sparks. Typestate-oriented programming. In *OOPSLA*, pages 1015–1022, October 2009.

[3] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Thery, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP - Certified Programs and Proofs - First International Conference - 2011*, volume 7086 of *Lecture notes in computer science - LNCS*, pages 135–150, Kenting, Taïwan, Province De Chine, December 2011. Springer.

[4] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# Programming System : An Overview. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices (CASSIS'04)*, volume 3362 of *Lecture Notes in Computer Science*, pages 49–69. Springer, 2004.

[5] Gilles Barthe, Delphine Demange, and David Pichardie. A formally verified SSA-based middle-end - static single assignment meets compcert. In Helmut Seidl, editor, *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 47–66. Springer, 2012.

[6] 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.

[7] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer-Verlag, 2004.

[8] 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.

[9] 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.

[10] 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.

[11] The Frama-C platform for static analysis of C programs, 2008. http://www.frama-c.cea.fr/.

[12] Paolo Herms. *Certification of a Tool Chain for Deductive Program Verification*. Thèse de doctorat, Université Paris-Sud, January 2013.

[13] Bart Jacobs, Jan Smans, and Frank Piessens. VeriFast : Imperative programs as proofs. In *VSTTE workshop on Tools & Experiments*, August 2010.

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

[15] Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4) :363–446, 2009.

[16] Stéphane Lescuyer. *Formalisation et développement d'une tactique réflexive pour la démonstration automatique en Coq*. Thèse de doctorat, Université Paris-Sud, January 2011.

[17] Claude Marché and Asma Tafat. Weakest precondition calculus, revisited using Why3. Research Report RR-8185, INRIA, December 2012.

[18] Aaron Stump, Morgan Deters, Adam Petcher, Todd Schiller, and Timothy Simpson. Verified programming in Guru. In *Proceedings of the 3rd workshop on Programming languages meets program verification*, PLPV '09, pages 49–58, New York, NY, USA, 2008. ACM.