Annexe à la proposition Concert

Cristal, Foc-Cnam, Lemme, Mimosa, Miró, Oasis

9 décembre 2002

Résumé

Dans cette annexe, nous joignons une étude des avancées récentes de la vérification formelle sur des thèmes proches du domaine des compilateurs. Cette étude montre que les conditions propices à l'aboutissement de ce thème sont maintenant réalisées.

Les avancées récentes décrites ici viennent s'ajouter à nos travaux plus anciens sur l'étude des langages de programmation et la compilation [7, 8, 6, 3].

1 Avancées récentes sur l'étude des langages de programmation

Le projet européen VérifiCard a permis de faire un progrès notable sur la description du langage JavaCard, un sous-ensemble significatif du langage Java. Ainsi, L'équipe Lemme a fourni une description formelle des instructions de la machine virtuelle [2, 4]. Une description similaire, incluant également des aspects d'analyse statique (vérification de bytecode), a été développée par la société Trusted Logic [14]. L'équipe *Bali* (T. Nipkow et ses collègues) à l'université technologique de Munich a fourni une description sémantique d'un fragment significatif de Java et a produit une *démonstration de correction* pour un compilateur non optimisant [17].

La possibilité d'utiliser des prouveurs de théorèmes pour la description sémantique de langages réalistes a également progressé de façon majeure au cours des dernières années. En particulier, le travail de Michael Norrish sur la description dans HOL du langage C [16] est particulièrement proche et pourra nous servir pour atteindre notre objectif.

Les travaux de B. Grégoire dans l'équipe Cristal sur la compilation du test de convertibilité dans le système Coq ont également débouché sur une preuve de correction d'un compilateur non optimisant vers une machine abstraite, le langage source étant ici les termes des Constructions [11]. Un apport de ce travail récent est une avancée sur la possibilité d'exécuter rapidement les fonctions définies dans le système de preuve, avec un impact prévisible sur les démonstrations automatiques. Sur les langages fonctionnels nous pouvons également citer les travaux sur l'inférence de types [9].

Un autre projet focalisé sur la vérification de compilateurs est le projet Verifix qui regroupe trois universités allemandes, en utilisant la technique des machines à états abstraites de Gurevitch, et le système PVS pour mécaniser une partie des vérifications [10]. Ce projet a également considéré des compilateurs optimisants et représente l'un des concurrents les plus avancés de notre proposition. Nous pensons que notre proposition présentera l'avantage d'une meilleure intégration dans le système de preuve, ce qui permettra une extraction directe du code du compilateur de la formalisation, ce qui permettra d'éviter un codage manuel qui risque de réintroduire des erreurs.

Depuis une dizaine d'années, plusieurs descriptions formelles et démonstration de correction ont été proposées sans vérification mécanique. L'un des plus impressionants est le compilateur VLisp décrit par Guttman, Farmer, Wand, et leurs collègues à Mitre Corporation et l'université North-Eastern de Boston [12]. L'absence de vérification mécanique repose sur deux raisons majeures. D'une part, ces auteurs veulent garder un contrôle humain sur l'implémentation de la machine, suivant l'argument que le programme de vérification de preuve peut lui-même contenir des erreurs. D'autre part, l'effort de formalisation est perçu comme un coût excessif. Le premier argument est difficile à débattre, mais nous pensons que le second devient moins significatif.

2 Avancées dans la vérification d'algorithmes

Les algorithmes d'optimisation sont souvent plus difficile que les algorithmes de traduction. D'une part, ils peuvent travailler sur des données plus complexes, comme des graphes, d'autre part il peuvent faire intervenir des arguments de terminaison distincts des arguments structurels usuels.

Nous pensons que la compétence générale pour traiter des algorithmes complexes a fait des progrès notables ces dernières années. En particulier, les algorithmes de vérification symbolique de formules booléennes (comme l'algorithme de Stålmarck [15] ou les algorithmes de manipulations de diagrammes de décision binaire [18] — BDD — ont été formalisés ces dernières années, bien qu'ils comportent des complexités algorithmiques majeures). De même, les algorithmes de vérification de bytecode Java étudiés au projet Lemme, dans l'équipe Bali, et chez Trusted Logic, sont autant d'exemples réalistes d'analyses *dataflow* effectuées sur du code machine non structuré (présentés sous forme de graphes de contrôle); leur formalisation n'a pas posé de difficultés majeures.

D'autre part, les techniques de description d'algorithme font maintenant apparaitre une séparation nette entre les arguments de calcul et les arguments de terminaison. Ainsi, il est possible d'aborder le raisonnement sur le résultat d'un programme *lorsqu'il termine* séparément du raisonnement qui assure sa terminaison. De façon grossière, il nous serait même possible d'assurer une correction partielle du compilateur, qui assure seulement la validité des résultats, sans assurer que ces résultats sont fournis[5, 1].

Références

[1] Antonia Balaa and Yves Bertot. Fix-point equations for well-founded recursion in type theory. In Harrison and Aagaard [13], pages 1–16.

[2] G. Barthe, G. Dufay, L. Jakubiec, B. Serpette, and S. Melo de Sousa. A Formal Executable Semantics of the JavaCard Platform. In D. Sands, editor, *Proceedings of ESOP'01*, volume 2028 of *Lecture Notes in Computer Science*, pages 302–319. Springer-Verlag, 2001.

[3] Y. Bertot. A certified compiler for an imperative language. rapport technique RR-3488, INRIA, 1998.

[4] Yves Bertot. Formalizing a jvml verifier for initialization in a t heorem prover. In *Computer Aided Verification (CAV'01)*, number 2102 in Lecture Notes in Computer Science. Springer-Verlag, July 2001.

[5] Ana Bove and Venanzio Capretta. Nested general recursion and partiality in type theory. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 121–135. Springer-Verlag, 2001.

[6] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order Abstract Syntax in Coq. In *Proc. of TLCA*, volume 902, pages 124–138. Springer-Verlag, 1995.

[7] Joëlle Despeyroux. Proof of translation in natural semantics. In *Symposium on Logic in Computer Science*, pages 193–205. IEEE Computer Society, June 1986.

[8] Joëlle Despeyroux and André Hirschowitz. Higher-order syntax and induction in Coq. In F. Pfenning, editor, *proceedings of the fifth Int. Conf. on Logic Programming and Automated Reasoning (LPAR 94)*, volume 822, pages 159–173. Springer-Verlag LNAI, July 1994. Also appears as INRIA Research Report RR-2292 (June 1994).

[9] Catherine Dubois and Valérie Ménissier-Morain. Certification of a type inference tool for ml: Damas-milner within coq. In *Journal of Automated Reasoning,*, pages 319–346. Kluwer Academic Publishers, 1999.

[10] Gerhard Goos and Wolf Zimmermann. Verifying compilers and asms. In Yuri Gurevich, Philipp W. Kutter, Martin Odersky, and Lothar Thiele, editors, *Abstract State Machines, Theory and Applications*, volume 1912 of *Lecture Notes in Computer Science*, pages 177–202. Springer-Verlag, 2000.

[11] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *International Conference on Functional Programming 2002*, pages 235–246. ACM Press, 2002.

[12] Joshua D. Guttman and Mitchell Wand (eds.). Special issue on VLISP. *Lisp and Symbolic Computation*, 8(1/2), 1995.

[13] J. Harrison and M. Aagaard, editors. *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[14] Xavier Leroy. Java bytecode verification: an overview. In *Computer Aided Verification (CAV'01)*, number 2102 in Lecture Notes in Computer Science. Springer-Verlag, July 2001.

[15] Pierre Letouzey and Laurent Théry. Formalizing Stålmarck's algorithm in Coq. In Harrison and Aagaard [13], pages 387–404.

[16] Michael Norrish. C formalised in HOL. Technical Report UCAM-CL-TR-453, University of Cambridge, December 1998.

[17] Martin Strecker. Formal verification of a Java compiler in Isabelle. In *Proc. Conference on Automated Deduction (CADE)*, volume 2392 of *Lecture Notes in Computer Science*, pages 63–77. Springer-Verlag, 2002.

[18] Kumar Neeraj Verma, Jean Goubault-Larrecq, Sanjiva Prasad, and S. Arun-Kumar. Reflecting bdds in coq. In *Proc. 6th Asian Computing Science Conference (ASIAN'2000)*, volume 1961 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.