Fondements sémantiques des représentations intermédiaires de programmes

Delphine Demange [1]

Lauréate du prix de thèse Gilles Kahn 2013

Delphine Demange a soutenu sa thèse en octobre 2012 à l'ÉNS Cachan, antenne de Bretagne, sous la direction de Thomas Jensen and David Pichardie, puis effectué un stage post-doctoral à l'université de Pennsylvanie (États-Unis). Elle est actuellement Maître de Conférences à l'université Rennes 1 et membre de l'équipe Celtique de l'IRISA.

Les programmes critiques sont des programmes dont une erreur à l'exécution aurait des conséquences désastreuses, en termes de vies humaines, de dégâts écologiques, ou financiers (systèmes bancaires, avionique, etc.). Ils requièrent donc de fortes garanties : leur exécution ne doit pas échouer, leur correction fonctionnelle doit être garantie. La vérification formelle de logiciels permet d'assurer qu'un logiciel remplit ces exigences. Mais la garantie ainsi apportée n'est réelle que si le vérifieur (lui aussi un programme) ne contient pas de bug, et si aucun bug n'est introduit dans le logiciel durant sa compilation (la vérification est le plus souvent faite au niveau source). Or, vue la complexité des compilateurs et vérifieurs actuels (langages sources haut-niveau, richesse

1. http://www.irisa.fr/celtique/demange/

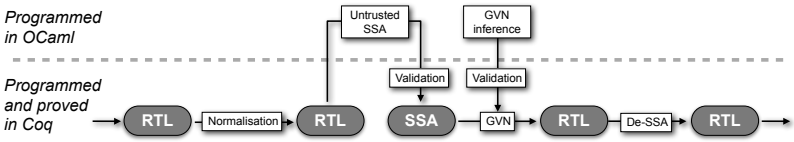

FIGURE 1. Middle-end SSA ajouté au compilateur CompCert, au niveau de la représentation RTL.

des propriétés, et besoins en performance), de tels risques existent. Ces logiciels demandent donc le même niveau de garantie que les programmes qu'ils analysent ou compilent. Le sujet de cette thèse est la vérification formelle de ces outils.

Pour simplifier l'analyse et la transformation de code, les compilateurs et vérifieurs utilisent des représentations intermédiaires (IR) de programmes. Nous soutenons que la preuve formelle du code des compilateurs/vérifieurs performants nécessite de considérer les IRs comme levier : utilisées pour leur propriétés sémantiques, les IRs devraient simplifier non seulement les algorithmes impliqués, mais aussi leur preuve de correction. Nous étudions d'un point de vue sémantique et formel les IRs, en choisissant trois problématiques récurrentes. Notre approche est de (1) formaliser la sémantique de l'IR de façon réaliste, en capturant l'intuition généralement admise, (2) d'en prouver un algorithme de génération conforme à l'état de l'art, (3) de pouvoir prouver des analyses ou optimisations sur l'IR (et donc d'identifier des lemmes sémantiques facilement applicables) et (4) de valider expérimentalement nos formalisations.

Nous étudions d'abord une IR basée registres pour le bytecode Java [1]. L'algorithme de génération, par évaluation symbolique, reconstruit des arbres d'expressions sans effet de bord, et décompile en une seule instruction la création d'objet (allocation, paramètres d'initialisation, et construction). Son théorème de correction sémantique explicite ce qui est préservé (initialisation des objets, ordre des exceptions) mais aussi ce qui est modifié et comment (ordre d'allocation). Nous implantons l'IR dans Sawja, un framework d'analyse statique pour Java, et évaluons empiriquement ses performances.

Puis, nous étudions Single Static Assignment (SSA) [3], une IR phare des compilateurs modernes, dont la formalisation sémantique reste encore peu développée. Dans [4], nous implantons et prouvons en Coq un middle-end SSA pour le compilateur C vérifié CompCert [2] : sémantique de SSA, preuve de la génération (par validation *a posteriori*), de la destruction, et d'une optimisation phare de SSA, CSE-GVN (voir Figure 1). Nous identifions et prouvons un invariant clé pour la preuve d'optimisations, permettant le raisonnement équationnel sur les programmes SSA. Enfin, nous évaluons les performances du middle-end.

Enfin, nous étudions, dans le cas de Java, un aspect difficile des langages modernes auquel les compilateurs et les vérifieurs doivent faire face, la concurrence. La définition du modèle mémoire (le JMM) autorise les optimisations agressives des compilateurs et des architectures multi-processeurs. Formalisée en 2005 [5], cette sémantique est trop complexe pour envisager, à ce jour, la preuve d'un compilateur vis à vis d'elle. Dans [6], nous proposons un sous-ensemble du JMM, spécialisé aux architectures TSO, défini axiomatiquement par ses réordonnancements mémoire. Pour conduire des preuves dans la lignée des travaux de Ševčík et al. [7], nous prouvons qu'il a une définition opérationnelle équivalente. Ces résultats peuvent s'appliquer à toutes les couches objet d'un compilateur, le modèle mémoire étant paramétré par la sémantique intra-thread des programmes. Une validation expérimentale montre qu'implanter ce modèle sur TSO a un coût raisonnable.

Références

[1] D. Demange, T. Jensen, D. Pichardie. A Provably Correct Stackless Intermediate Representation for Java Bytecode. Proc. 8th Asian conference on Programming languages and systems, APLAS'10, 97–113 (2010).

[2] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM* 52(7), 107-115 (2009).

[3] R. Cytron, J. Ferrante, B. K. Rosen, N. M. Wegman, F. K. Zadeck. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Trans. on Programming Languages and Systems (TOPLAS)* 13(4), 451–490 (1991).

[4] G. Barthe, D. Demange, D. Pichardie. Formal Verification of an SSA-Based Middle-End for CompCert. *ACM Trans. on Programming Languages and Systems (TOPLAS)* 36(1), Article no. 4 (2014).

[5] J. Manson, W. Pugh, S. V. Adve. The Java Memory Model. Proc. 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL'05, 378–391 (2005).

[6] D. Demange, V. Laporte, L. Zhao, S. Jagannathan, D. Pichardie, and J. Vitek. Plan B : A Buffered Memory Model for Java. Proc. 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL'13, 329–342 (2013).

[7] J. Ševčík, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, P. Sewell. CompCertTSO : A Verified Compiler for Relaxed-Memory Concurrency. *Journal of the ACM* 60(3), Article no. 22 (2013).