Modèle de fuite structurée & applications à la compilation sécurisée

Gilles Barthe[1,2], Benjamin Gregoire[3], Vincent Laporte*[4], and Swarn Priya[3]

[1]MPI-SP – Allemagne
[2]IMDEA Software Institute – Espagne
[3]Inria Sophia Antipolis – Université Côte d'Azur, Inria – France
[4]LORIA – CNRS, Inria, Université de Lorraine – F-54500 Nancy, France

Résumé

Un compilateur correct permet d'étudier les comportements du programme cible - résultat de la compilation - en observant uniquement le programme source : nul besoin d'observer le programme généré ni de comprendre l'implémentation du compilateur pour savoir que le programme cible calcule la même fonction que le programme source correspondant.

Cependant, cette faculté est limitée aux propriétés fonctionnelles des comportements du programme cible : on sait ce que le programme cible calcule - la même chose que le programme source - mais on ne sait pas comment. Pourtant, connaître des propriétés non fonctionnelles du programme cible est souvent instructif : quel est le coût (p.ex. : le temps) d'une exécution du programme ? le programme fait-il fuir des données sensibles par des canaux auxiliaires ? etc.

La sémantique des langages de programmation peut être étendue (on dit aussi " instrumentée ") et rendue plus précise pour permettre de décrire de telles propriétés (ou hyper-propriétés) non-fonctionnelles ; cependant, la correction des compilateurs ne peut pas, en général, être étendue à ces sémantiques instrumentées : les optimisations peuvent modifier les effets non-fonctionnels rendus visibles par l'instrumentation (que l'on nomme la " fuite " d'une exécution).

Dans ce travail nous présentons un modèle de fuite structurée comme un moyen d'instrumenter les sémantiques des langages de programmation pour permettre de décrire un éventail de propriétés (et hyper-propriétés) non-fonctionnelles d'intérêt comme le coût à l'exécution ou la politique de sécurité " constant-time ". Étant entendu qu'un compilateur ne préserve pas, en général, la fuite associée à l'exécution d'un programme, le caractère structuré de cette fuite permet toutefois de décrire simplement l'effet de la compilation sur celle-ci. Plus précisément, en plus d'un programme cible, le compilateur produit un " transformateur de fuites " qui décrit comment calculer la fuite d'une exécution cible à partir de la fuite de l'exécution correspondante dans le programme source. Ces transformateurs de fuite sont exprimés dans un langage dédié, ce qui permet de les employer pour justifier la préservation de propriétés non-fonctionnelles mais aussi pour décrire comment ces propriétés sont modifiées.

*Intervenant

L'application de cette méthodologie au compilateur Jasmin (dont la correction est formellement vérifiée en Coq) a ainsi permis de prouver que ce compilateur préserve la contre-mesure dite " constant-time " et de transférer au niveau assembleur des résultats d'une analyse de coût effectuée au niveau source. Cette analyse peut donc s'appuyer sur les abstractions et structures de haut niveau qui sont explicites au niveau source et ainsi être bien plus précise qu'une analyse similaire effectuée directement sur un programme assembleur.