Gestionnaire mémoire dédié à la vérification à l'exécution de programmes C: résumé du stage

Romain Maliach-Auguste

automne 2020

Résumé

Le langage ACSL[1] permet d'annoter un programme C99[2] avec des propriétés logiques spécifiant son comportement attendu. E-ACSL[6], greffon de la plateforme d'analyse Frama-C[3], compile le programme C et ses annotations en un nouveau programme C, dont la correction vis-à-vis des propriétés est vérifiée dynamiquement (c'est-à-dire pendant son exécution). Au cours de ce stage, j'ai commencé une démarche d'optimisation du code généré par E-ACSL, notamment en ce qui concerne la mémoire. Pour ce faire, j'ai étudié le comportement d'E-ACSL pour déterminer des points d'optimisation possibles, écrit une bibliothèque C définissant un gestionnaire mémoire, puis j'ai adapté E-ACSL pour qu'il génère des appels à cette bibliothèque.

Étude du compilateur en boîte noire

La première étape du stage a consisté en l'étude du compilateur E-ACSL en boîte noire, c'est-à-dire que j'ai choisi certaines entrées, observées les sorties générées par E-ACSL, et essayé d'en déduire son fonctionnement interne. Cela m'a permis de poser des questions utiles pour ma formation et pertinentes pour l'équipe. J'ai pu faire plusieurs propositions d'amélioration, dont une correction de bug. Certaines propositions ont été retenues et implémentées immédiatement ; certaines propositions ont nécessité un travail de fond qui a constitué le reste de mon stage ; et les autres n'étaient pas pertinentes, ou nécessitaient d'autres travaux, de développement ou de recherche, qui dépassaient le cadre de mon stage.

Tâche principale du stage

Le problème auquel je me suis attaqué durant la suite de mon stage est l'utilisation de la mémoire par E-ACSL, pour les calculs logiques et arithmétiques. E-ACSL génère un code C, et utilise des structures de données fournies par C. La génération de ce code était largement sous-optimale : par exemple, elle faisait appel à la création de nombreuses variables intermédiaires, dont l'occupation mémoire est non négligeable, car E-ACSL « surdimensionne » ses types pour garantir l'exactitude de ses calculs de vérification. Cela était dû à un schéma de compilation récursif, où le passage de l'évaluation d'un prédicat (ou terme)[1] à son opérateur supérieur se faisait quasiment systématiquement par la création d'une variable. Un autre problème était l'utilisation du type int pour stocker des booléens, très nombreux dans la compilation typique d'une instruction ACSL - il s'agit d'un gâchis de mémoire.

Solution proposée

Afin de mieux maîtriser l'occupation mémoire, nous avons fait le choix de compiler les instructions ACSL vers un langage inter-

médiaire que j'ai conçu. En effet, des travaux théoriques récents au LSL ont montré qu'il était possible de définir l'instrumentation indépendamment de la façon dont la mémoire dont elle a besoin est gérée.[4]. Ce langage intermédiaire est composé de primitives simples, dans le style d'un langage assembleur. On peut décomposer l'évaluation d'une expression logique ou arithmétique d'ACSL en une succession de ces primitives. La structure de données retenue est celle de la pile, qui est assez « naturelle ». Les primitives correspondent au calcul de la valeur des différents noeuds que constituent les opérateurs arithmétiques et logiques. Une bibliothèque C, que j'ai conçue et écrite, est en charge d'effectivement manipuler la pile et d'effectuer les opérations.

Tâches réalisées

Spécification du langage assembleur intermédiaire, de la structure de pile et du schéma de compilation. Écriture des opérateurs, du compilateur ACSL vers assembleur, du compilateur assembleur vers C. Interfaçage avec le code ACSL non traduit en assembleur. Tests unitaires. Écriture de quelques optimisations au niveau du code assembleur.

Apports à la fin du stage

Documentation de certains fonctionnements d'E-ACSL, contributions au la base de code de Frama-C, preuve de concept pour une compilation par un langage intermédiaire. L'application d'une telle contribution serait l'amélioration des performances de l'instrumentation. Rapport.[5]

Pistes ouvertes

Écriture d'optimisations au niveau du code assembleur. Traduction de termes et prédicats[1] plus complexes. Meilleur intefaçage avec les nombres de précision arbitraire (GMP).

Références

[1] Patrick Baudin, Pascal Cuoq, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy et Virgile Prevosto. *ACSL : ANSI/ISO C Specification Language*. 1.16. CEA LIST, Software Security Laboratory et INRIA. 2020. URL : `https://github.com/acsl-language/acsl`.

[2] *International Standard ISO/IEC 9899*. Standard. Ultimate revision of the C99 standard. ISO/IEC JTC 1, sept. 2007.

[3] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles et Boris Yakobowski. « Frama-C : A Software Analysis Perspective ». English. In : *Formal Aspects of Computing* (jan. 2015), p. 1-37. URL : `http://julien.signoles.free.fr/publis/2015_fac.pdf`.

[4] Dara Ly, Nikolai Kosmatov, Frédéric Loulergue et Julien Signoles. « Verified Runtime Assertion Checking for Memory Properties ». In : *International Conference on Tests and Proofs (TAP)*. Juin 2020. URL : `http://julien.signoles.free.fr/publis/2020_tap.pdf`.

[5] Romain Maliach-Auguste. *Gestionnaire mémoire dédié à la vérification à l'exécution de programmes C : rapport de stage*. Rapp. tech. CEA - UTC, 2021. URL : `https://romain.maliach.fr/publications/gestionnaire-memoire-calcul-predicats-acsl.pdf`.

[6] Julien Signoles, Nikolai Kosmatov et Kostyantyn Vorobyov. « E-ACSL, a Runtime Verification Tool for Safety and Security of C Programs. Tool Paper ». In : *International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES)*. Sept. 2017. URL : `https://julien-signoles.fr/publis/2017_rvcubes_tool.pdf`.