VERIFICATION OF A CONCURRENT GARBAGE COLLECTOR

le 19 décembre 2017 14h30

ENS Rennes, Salle du conseil

Soutenance de thèse de Yannick Zakowski (équipe Celtique - IRISA / Inria Rennes)
mention informatique

Les compilateurs modernes constituent des programmes complexes, réalisant de nombreuses optimisations afin d'améliorer la performance du code généré. Du fait de cette complexité, des bugs y sont régulièrement détecté, conduisant à l'introduction de nouveau comportement dans le programme compilé.

En réaction, il est aujourd'hui possible de prouver correct, dans des assistants de preuve tels que Coq, des compilateurs optimisants pour des langages tels que le C ou ML. Transporter un tel résultat pour des langages haut-niveau tels que Java est néanmoins encore hors de portée de l'état de l'art. Ceux-ci possèdent en effet deux caractéristiques essentielles: la concurrence et un environnement d'exécution particulièrement complexe.

Nous proposons dans cette thèse de réduire la distance vers la conception d'un tel compilateur vérifié en nous concentrant plus spécifiquement sur la preuve de correction d'un glaneur de cellules concurrent performant. Ce composant de l'environnement d'exécution prend soin de collecter de manière automatique la mémoire, en lieu et place du programmeur. Afin de ne pas générer un ralentissement trop élevé à l'exécution, le glaneur de cellules doit être extrêmement performant. Plus spécifiquement, l'algorithme considéré est dit «au vol»: grâce à l'usage de concurrence très fine, il ne cause jamais d'attente active au sein d'un fil utilisateur. La preuve de correction établit par conséquent que malgré l'intrication complexe des fils utilisateurs et du collecteur, ce dernier ne collecte jamais une cellule encore accessible par les premiers.

Nous présentons dans un premier temps l'algorithme considéré et sa formalisation en Coq dans une représentation intermédiaire conçue pour l'occasion. Nous introduisons le système de preuve que nous avons employé, une variante issue de la logique «Rely-Guarantee», et prouvons l'algorithme correct.

Raisonner simultanément sur l'algorithme de collection et sur l'implantation des structures de données concurrentes manipulées générerait une complexité additionnelle indésirable. Nous considérons donc durant la preuve des opérations abstraites: elles ont lieu instantanément. Pour légitimer cette simplification, nous introduisons une méthode inspirée par les travaux de Vafeiadis et permettant la preuve de raffinement de structures de données concurrentes dites «linéarisables». Nous formalisons l'approche en Coq et la dotons de solides fondations sémantiques. Cette méthode est finalement instanciée sur la principale structure de données

utilisée par le glaneur de cellules.

Abstract :

Modern compilers are complex programs, performing several heuristic-based optimisations. As such, and despite extensive testing, they may contain bugs leading to the introduction of new behaviours in the compiled program.
To address this issue, we are nowadays able to prove correct, in proof assistants such as Coq, optimising compilers for languages such as C or ML. To date, a similar result for high-level languages such as Java nonetheless remain out of reach. Such languages indeed possess two essential characteristics: concurrency and a particularly complex runtime.
This thesis aims at reducing the gap toward the implementation of such a verified compiler. To do so, we focus more specifically on a state-of-the-art concurrent garbage collector. This component of the runtime takes care of automatically reclaiming memory during the execution to remove this burden from the developer side. In order to keep the induced overhead as low as possible, the garbage collector needs to be extremely efficient. More specifically, the algorithm considered is said to be ``on the fly'': by relying on fine-grained concurrency, the user-threads are never caused to actively wait. The key property we establish is the functional correctness of this garbage collector, i.e. that a cell that a user thread may still access is never reclaimed.
We present in a first phase the algorithm considered and its formalisation in Coq by implementing it in a dedicated intermediate representation. We introduce the proof system we used to conduct the proof, a variant based on the well-established Rely-Guarantee logic, and prove the algorithm correct.
Reasoning simultaneously over both the garbage collection algorithm itself and the implementation of the concurrent data-structures it uses would entail an undesired additional complexity. The proof is therefore conducted with respect to abstract operations: they take place instantaneously. To justify this simplification, we introduce in a second phase a methodology inspired by the work of Vafeiadis and dedicated to the proof of observational refinement for so-called ``linearisable'' concurrent data-structures. We provide the approach with solid semantic foundations, formalised in Coq. This methodology is instantiated to soundly implement the main data-structure used in our garbage collector.

THÉMATIQUE(S)

Débouchés, Recherche - Valorisation, Vie de l'École

Mise à jour le 23 janvier 2018

JURY

Marieke Huisman / rapporteur
Professeure, Université de Twente (Pays-Bas)
Stephan Merz / rapporteur
Directeur de recherche, Inria Nancy
Daniel Hirschkoff/ examinateur
Maître de Conférences, ENS de Lyon
Francesco Zappa Nardelli / examinateur
Directeur de recherche, Inria Paris
Olivier Ridoux / examinateur
Professeur des universités, Université de Rennes 1
David Cachera / encadrant
Maître de conférences, ENS Rennes
David Pichardie / encadrant
Professeur des universités, ENS Rennes