Invariants de classe et systèmes d'ownership

Romain Bardou, Claude Marché, INRIA Futurs, équipe ProVal

Mars – Juillet 2007

Le contexte général

Il existe de nombreux outils dédiés à la preuve de programme. L'INRIA Futurs développe la plateforme Why. Elle permet d'annoter des programmes C ou Java par une spécification décrivant leur comportement. Ensuite, ces programmes sont traduits en Jessie, langage intermédiaire, puis en Why. Enfin, ces programmes Why sont traduits en obligations de preuve lisibles par des outils de démonstration automatiques comme Simplify ou des assistants de preuve comme Coq. La validité des obligations de preuve implique la correction du programme vis-à-vis de sa spécification.

Le problème étudié

J'ai étudié la notion d'invariant de classe. Lorsqu'on crée une structure de donnée, on a souvent en tête un invariant pour cette structure. Par exemple, une structure représentant un cercle peut posséder un entier pour décrire ce rayon ; un invariant possible serait que cet entier est toujours positif.

Si l'invariant est rompu, la structure de donnée n'a plus de sens, et donc le programme a de grandes chances de ne plus fonctionner correctement. Il est donc intéressant de permettre au programmeur de spécifier les invariants de ses structures. Ceci lui permet de mieux structurer les preuves de ses programmes, et d'éviter certaines erreurs dans l'implémentation des librairies, ce qui permet de gagner en modularité.

Ce n'est pas un nouveau problème et plusieurs solutions ont été proposées. Celles-ci sont souvent fondées sur des notions d'ownership, c'est-à-dire d'appartenance entre objets. Notament, Boogie fournit une notion d'invariant intéressante, mais qui ne peut pas être implémentée directement dans Jessie. En effet, la méthodologie de Boogie est adaptée lorsque la mémoire est modélisée sous forme de bloc unique en tout point de programme, ce qui n'est pas le cas dans Jessie. J'ai donc cherché une notion d'invariant adaptée à Jessie et je l'ai implémentée.

La contribution proposée

J'ai d'abord étudié différentes notions existantes, notament celle de Boogie et le système de type Universes de JML. Finalement, j'ai choisi d'adapter la méthode de Boogie. Puisque dans Jessie la mémoire ne se présente pas sous forme d'un seul bloc, j'ai cherché un moyen de relier les différents blocs entre eux en un point de programme donné.

Avant cela, j'ai imaginé un prédicat inductif exprimable en Coq et définissant les invariants. Ceci n'a pas donné lieu à une solution satisfaisante pour le problème des invariants dans

Jessie, mais a permis d'avoir une meilleure idée de l'expressivité de celle que j'ai finalement implémentée.

Sans rapport avec les invariants, j'ai aussi effectué une réalisation en Coq du modèle mémoire de Jessie. Ceci permet de se convaincre de la cohérence de celui-ci.

Les arguments en faveur de sa validité

Le système utilisé dans Boogie est cohérent, expressif et flexible. De plus, il possède de bonnes propriétés de modularité. Etant donné que ma solution est aussi expressive que celle de Boogie, elle présente aussi toutes ces propriétés, tout en conservant les avantages d'une mémoire séparée. On peut donc imaginer qu'elle soit généralisable à un ensemble de langages plus important.

Le bilan et les perspectives

La plateforme Why possède maintenant, par l'intermédiaire de Jessie, une notion d'invariant puissante. La prochaine étape est d'utiliser ces invariants au profit des autres fonctionalités de Jessie. De nombreuses notions de programmation sont fondées sur des invariants ; leur intégration dans Jessie, et donc dans la plateforme Why, en sera maintenant facilitée.

On peut aussi chercher à étendre la solution que j'ai implémentée. Elle est suffisament flexible pour que l'on puisse imaginer l'améliorer encore.

Pendant mon stage, j'ai rencontré un autre problème intéressant. On a souvent besoin de montrer que deux pointeurs sont différents, pour être sûr qu'en modifiant l'un des deux on n'a pas rompu les propriétés (par exemple les invariants) que l'on a établies concernant le deuxième. La notion d'ownership ne répond que partiellement à ce problème.