Dompter mutabilité et aliasing dans le langage de spécification Gospel pour OCaml

Proposition de stage de recherche (niveau L3/M1)

2022

Encadrant

Armaël Guéneau, chargé de recherche, LMF et Inria Saclay (armael.gueneau@inria.fr)

Résumé

Le projet Gospel s'intéresse à la conception d'un langage de spécification pour OCaml, permettant aux programmeurs OCaml d'exprimer formellement le comportement attendu des fonctions d'une bibliothèque. Je participe actuellement au développement d'une extension de Gospel permettant de raisonner finement sur des structures de données combinant mutabilité et aliasing (structures mutables imbriquées par exemple). Dans ce stage, je propose de contribuer à cette extension en implémentant une traduction de notre « Gospel étendu » vers une logique plus élémentaire (la logique de séparation), afin de permettre à des outils de vérification existants d'exploiter ces spécifications plus expressives.

Situation du sujet

Le projet ANR Gospel [gosb, gosa], dont je fais partie, a l'objectif suivant : permettre à des programmeurs OCaml de facilement spécifier, tester, et même vérifier formellement des propriétés à propos de leur code, comme partie intégrante de leur pratique de développement quotidienne.

Gospel définit un langage de spécification pour OCaml. En premier temps, il s'agit de définir et développer un langage permettant d'*écrire* des spécification pour des fonctions OCaml, associées à leur déclaration dans un fichier `.mli`. En second temps, il s'agit de développer un écosystème d'outils sachant *lire* ces spécifications, dans le but d'établir la correction du programme associé à une spécification donnée : par exemple par le test [FP21, OP21] (pour chercher des bugs) ou la preuve [PR21] (pour prouver la correction du code).

Le langage Gospel, dans son état actuel, permet d'écrire des spécifications pour une classe déjà intéressante de programmes, mais son expressivité reste limitée. En particulier, il ne permet pas encore de spécifier le comportement de programmes s'appuyant sur des combinaisons non triviales d'*état mutable* et d'*« aliasing »*. Par exemple, on ne peut pas spécifier de façon satisfaisante le comportement de fonctions prenant en argument ou renvoyant des listes de tableaux, tableaux de

tableaux ou autres structures mutables imbriquées. Les structures mutables à base de pointeurs (listes chainées, etc.) sont également problématiques.

Je travaille actuellement, avec mes collègues Arthur Charguéraud (Inria Strasbourg), Mário Pereira (Universidade Nova de Lisboa, Portugal) et François Pottier (Inria Paris), à la conception d'une extension de Gospel répondant à ces besoins. Cette extension de Gospel permet d'exprimer des spécifications en s'appuyant sur une notion de « possession mémoire » s'inspirant notamment de la logique de séparation, mais aussi de systèmes de types sous-structurels comme celui de Rust [rus] ou (plus récemment) Spark Ada [JDM+20]. Nous avons pour l'instant défini le cœur d'une telle extension, mais les détails sont encore informels et décrits en prose ; un certain nombre d'aspects restent à développer et clarifier. Par ailleurs, pour pouvoir utiliser ces nouvelles spécifications Gospel, il est nécessaire de pouvoir les traduire vers une logique prise en charge par des outils de vérification existants.

Dans ce stage, je propose de rendre cette proposition d'extension plus concrète en implémentant une traduction depuis notre « Gospel étendu » vers la logique de séparation (une logique plus élémentaire et expressive). Autrement dit, il s'agit d'implémenter un petit compilateur du langage de spécification Gospel étendu vers un fragment de la logique de séparation.

Ce travail serait également l'occasion d'évaluer notre proposition d'extension et sa faisabilité. Fort de l'expérience d'implémentation, une possibilité sera de suggérer des fonctionnalités encore manquantes au « Gospel étendu », ou d'ajuster la définition de certaines aspects du langage si cela peut rendre leur compilation plus simple ou élégante.

Objectifs

Le programme est donc le suivant :

1. Se familiariser avec Gospel en écrivant des spécifications pour des exemples simples, d'abord dans le cadre du langage Gospel actuel, puis dans le cadre de notre proposition d'extension à Gospel.
2. Se familiariser avec la logique de séparation en écrivant une version équivalente de ces spécifications Gospel, mais en logique de séparation.
3. Écrire un petit compilateur capable d'effectuer cette traduction automatiquement, d'abord pour un sous-ensemble de Gospel, puis étendre le compilateur pour prendre en charge d'avantage de fonctionnalités de Gospel.

Pré-requis

De bases solides en algorithmique et en programmation sont indispensables. Une familiarité avec le langage OCaml est également indispensable. Déjà connaître la logique de séparation n'est cependant pas nécessaire.

Détails pratiques

Le stage se déroulera au LMF, bâtiment 650, sur le plateau de Saclay (Campus Universitaire, Rue Raimond Castaing, Bâtiment 650, 91190 Gif-sur-Yvette).

Références

[FP21] Jean-Christophe Filliâtre and Clément Pascutto. Ortac : Runtime assertion checking for OCaml (tool paper). In Lu Feng and Dana Fisman, editors, *Runtime Verification*, pages 244–253, Cham, 2021. Springer International Publishing.

[gosa] The Gospel github repository. https://github.com/ocaml-gospel/gospel.

[gosb] The Gospel website. https://ocaml-gospel.github.io/gospel/.

[JDM+20] Georges-Axel Jaloyan, Claire Dross, Maroua Maalej, Yannick Moy, and Andrei Paskevich. Verification of programs with pointers in spark. In Shang-Wei Lin, Zhe Hou, and Brendan Mahony, editors, *Formal Methods and Software Engineering*, pages 55–72, Cham, 2020. Springer International Publishing.

[OP21] Nicolas Osborne and Clément Pascutto. Leveraging Formal Specifications to Generate Fuzzing Suites. In *OCaml Users and Developers Workshop, co-located with the 26th ACM SIGPLAN International Conference on Functional Programming*, Virtual, United States, August 2021.

[PR21] Mário Pereira and António Ravara. Cameleer : a deductive verification tool for ocaml. *CoRR*, abs/2104.11050, 2021.

[rus] The Rust language website. https://rust-lang.org.