Interopérabilité vérifiée entre OCaml et C

Proposition de stage de recherche (niveau M2)

2022

Encadrant

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

Résumé

Il est courant pour des bibliothèques OCaml de s'interfacer à des bibliothèques plus bas niveau écrites en C. Ceci est rendu possible via l'implémentation de code « glue » écrit en C mais soumis à diverses restrictions dictées par l'implémentation du langage OCaml. Il n'existe aujourd'hui pas de spécification formelle de ces restrictions, qui permettraient de vérifier rigoureusement la sûreté d'un tel code glue. Je propose dans ce stage de s'intéresser à la conception de telles règles de raisonnement pour l'interopérabilité entre OCaml et C.

Situation du sujet

Il existe aujourd'hui de multiples approches permettant de vérifier formellement la sûreté ou correction de programmes, chaque outil étant typiquement spécifique pour un langage de programmation particulier.

Cependant, en pratique, un nombre non négligeable de bibliothèques sont implémentées non pas dans un unique langage mais dans une combinaison de plusieurs langages. C'est typiquement le cas de bibliothèques pour un langage de haut niveau (par exemple OCaml) dont une partie est implémentée à l'aide de code bas niveau (typiquement écrit en C). Les raisons sont multiples : efficacité supplémentaire, accès à des primitives non disponibles dans le langage, interaction avec le système d'exploitation, ou encore interopérabilité avec des bibliothèques tierces implémentées dans un autre langage.

Les détails pratiques de la « FFI » (*Foreign Function Interface*), qui définit comment lier du code de haut niveau avec du code plus bas niveau, varient d'un langage à un autre. Dans le cas d'OCaml, un ensemble d'interfaces C est fourni à la programmeuse ou au programmeur, qui doit alors les utiliser avec soin pour écrire du code C servant de « glue » entre son code OCaml et le code C externe. Le programmeur a un contrôle fin lui permettant d'implémenter un code de FFI le plus efficace possible ; mais en retour, il doit obéir à de nombreuses restrictions lors de son interaction avec le langage de haut niveau. Pour OCaml, ces restrictions ne sont documentées que partiellement et informellement [man]. La programmation de code d'interfaçage entre OCaml et C reste un art réservé aux initiés, et le débogage est difficile.

Mon projet de recherche principal (mené en collaboration avec des chercheurs de l'université d'Aarhus et du MPI-SWS) s'attaque à la spécification et *vérification formelle* de ces règles d'interopérabilité et de leur bonne utilisation. Je travaille à développer des règles basées sur la *logique de séparation* permettant de vérifier la correction de code « glue » interopérant entre OCaml et C ; ces règles étant elle mêmes justifiées vis-à-vis d'une sémantique opérationnelle plus élémentaire. Le tout est formalisé en Coq en utilisant le framework de logique de séparation Iris [JKJ+18, iri].

Afin de justifier les fondements théoriques de l'approche, ce travail ne considère pour le moment qu'un sous-ensemble minimal des interactions possibles entre un mini-Caml et mini-C. Une bonne partie des opérations utilisées en pratique par du code C s'interfaçant avec OCaml n'est pas encore modélisée.

En se basant sur les règles minimales existantes, au cours de ce stage, je propose comme point de départ d'étoffer ce panel de règles de raisonnement, en concevant de nouvelles règles permettant de vérifier des exemples intéressants de code C utilisant la FFI OCaml. Afin de valider ces règles, une possibilité sera alors de montrer qu'elles sont *utiles*, en les intégrant à un outil de vérification de code C [SLK+21, JP08], dans le but d'appliquer celui-ci au code C inclus dans des bibliothèques OCaml « de la vraie vie ». Alternativement, on pourra chercher à montrer rigoureusement que les règles sont *correctes*, en étendant la formalisation Coq pour prouver formellement le bien fondé de ces nouvelles règles.

Objectifs

Le programme de travail est donc le suivant.

1. Identifier un ensemble de bibliothèques OCaml utilisant du code C via la FFI qui semblent importantes. Des points de départ possibles sont la bibliothèque standard, la bibliothèque zarith [zar], ou encore des bibliothèques exposant une interface OCaml pour des bibliothèques C (tsdl [tsd], etc). Identifier quel sous-ensemble de la FFI OCaml ces bibliothèques utilisent.
2. Écrire des règles en logique de séparation permettant de raisonner sur le code C « glue » trouvé dans ces bibliothèques.
3. Valider ces règles vis-à-vis de ce code, par exemple en ajoutant ces règles à un outil de vérification existant, que l'on exécutera sur le code en question.

Pré-requis

Avoir suivi un cours de preuve de programmes, bonus si celui-ci inclut une partie sur la logique de séparation (comme par exemple le cours « Preuves de programmes » du M2 MPRI). Souhaitable : de bases solides en programmation, et une familiarité avec les langages C et OCaml.

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

[iri] The Iris website. https://iris-project.org.

[JKJ+18] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up : A modular foundation for higher-order concurrent separation logic. 2018.

[JP08] Bart Jacobs and Frank Piessens. The VeriFast program verifier. Technical report, Technical Report CW-520, Department of Computer Science, Katholieke ..., 2008.

[man] The OCaml manual : Interfacing C with OCaml. https://v2.ocaml.org/manual/intfc.html.

[SLK+21] Michael Sammler, Rodolphe Lepigre, Robbert Krebbers, Kayvan Memarian, Derek Dreyer, and Deepak Garg. RefinedC : Automating the foundational verification of C code with refined ownership types. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 158–174, New York, NY, USA, 2021. Association for Computing Machinery.

[tsd] The tsdl library. https://github.com/dbuenzli/tsdl.

[zar] The zarith library. https://github.com/ocaml/Zarith.