Owi4seacoral : exécution symbolique pour la génération de tests ciblant des labels

Léo Andrès
OCamlPro
leo@ocamlpro.com

Steven de Oliveira
OCamlPro
steven.de-oliveira@ocamlpro.com

Nicolas Berthier
OCamlPro
nicolas.berthier@ocamlpro.com

Ce stage offre à un·e étudiant·e curieux·se et motivé·e l'opportunité de participer à un projet de recherche combinant **programmation avancée** (en OCaml) et enjeux concrets d'analyse logicielle appliquée à des programmes C.

Contexte

Le contexte de ce stage est celui de l'analyse statique de programmes WebAssembly (Wasm) [8] et plus précisément de l'outil **Owi** [1] développé chez OCamlPro en collaboration avec l'INESC-ID/Instituto Superior Técnico de l'Université de Lisbonne. Owi permet d'effectuer de l'**exécution symbolique** [10] de programmes Wasm. C'est-à-dire que l'utilisateur peut fournir un programme et obtenir automatiquement les valeurs d'entrées pour lesquelles le programme lève une erreur. Il est également possible d'analyser des programmes écrits dans d'autres langages tels que Rust ou C en les compilant vers Wasm, ce que l'outil est capable de faire directement. L'exécution symbolique dans Owi est particulière en ce qu'elle permet d'analyser des programmes inter-langages[1], qu'elle est effectuée en parallèle[2] et qu'aucune approximation n'y est faite.[3] Une description détaillée d'Owi est disponible dans plusieurs publications [2, 3, 9].

OCamlPro développe par ailleurs **Seacoral**, un orchestrateur d'outils de génération automatique de tests. L'objectif principal de Seacoral est de configurer et orchestrer chaque outil de manière à générer une suite de tests qui satisfasse un **critère de couverture** donné ; un objectif secondaire est de minimiser la taille de cette suite de tests. Seacoral utilise Frama-C/LTest [4] pour insérer automatiquement dans le code C des « labels (de couverture) » qui encodent le critère de couverture désiré. Cette approche permet de cibler une grande variété de critères, qui vont des conditions (CC) et décisions (DC), à des critères plus avancés comme conditions multiples (MCC), limites, mutations, ou encore MC/DC [7]. Seacoral est d'ores et déjà capable de faire coopérer un ensemble d'outils qui emploient des techniques variées et complémentaires. Cet ensemble comprend notamment le model-checker CBMC, le moteur d'exécution symbolique Klee, ou encore le fuzzer libFuzzer.

Sujet

Ce projet de recherche s'inscrit dans le prolongement des travaux menés sur Owi et Seacoral, avec pour objectif principal de **permettre à Seacoral d'utiliser Owi comme générateur de tests**, en plus des outils déjà pris en charge. Pour atteindre cet objectif, plusieurs étapes sont nécessaires.

Tout d'abord, il est nécessaire d'**ajouter dans Owi une gestion des labels de couverture**. Contrairement à la solution en « boîte noire » qui a été conçue pour Klee [5], nous proposons d'ajouter un support des labels de couverture directement dans Owi. Cette approche permettra d'optimiser l'algorithme d'exécution symbolique de manière à ne rechercher qu'une couverture des labels, tout en ignorant certains comportements comme les erreurs (qui ne sont pas à couvrir par la suite de tests à

[1] Par exemple, un programme fait à la fois de code Rust et de code C.
[2] Grâce à OCaml 5.
[3] Si l'analyse termine, on a la garantie d'une absence de bogue.

produire). Il faudra donc concevoir et mettre en œuvre un mécanisme équivalent ou complémentaire pour Owi, afin de :

- interpréter les labels de couverture pendant l'exécution symbolique ;
- rendre compte de la couverture obtenue en termes de labels couverts (cela pourra se faire en fin d'exécution dans un premier temps, puis en cours d'exécution ensuite) ;
- optimiser l'algorithme d'exploration pour prendre en compte les informations déjà connues sur les labels ;
- assurer la cohérence entre Owi et les autres outils mobilisés par Seacoral.

Ensuite, on cherchera à **automatiser la production de tests** (harnais de test) exploitables par Owi. Ce travail inclut la gestion de données d'entrée potentiellement complexes (pointeurs, tableaux de taille variable, structures dynamiques) et doit assurer la compatibilité avec les routines déjà utilisées dans l'orchestrateur. Un soin particulier sera porté à la manière de formuler les hypothèses (`assume`, contraintes, etc.) nécessaires au moteur symbolique.

Enfin, il faut chercher à **reconstruire et valider les modèles générés**. En effet, une fois les chemins symboliques explorés, il est nécessaire de reconstruire les valeurs concrètes (modèles) permettant de rejouer ou de valider les tests. Owi doit donc fournir un format de sortie exploitable avec Seacoral, de sorte que les modèles générés (pouvant inclure des zones de mémoire symbolisées) soient convertibles en cas de tests concrets.

Si le temps le permet, des extensions pourraient être envisagées. Par exemple, il serait intéressant d'évaluer la performance et la complémentarité d'Owi par rapport aux autres outils actuellement intégrés à Seacoral, par exemple KLEE [6]. Cette analyse comparative permettrait de mieux cerner les forces et limites de chaque moteur, et d'envisager, si besoin, des évolutions futures pour améliorer l'efficacité de l'orchestration.

📚 Pré-requis

- niveau M1/M2 informatique ou équivalent
- bonne maîtrise d'un langage fonctionnel (idéalement OCaml)
- bonne maîtrise du langage C
- notions de compilation

🏆 Compétences développées

- découverte approfondie de WebAssembly
- programmation OCaml avancée
- techniques d'exécution symbolique
- techniques de génération automatique de tests

📍 Lieu du stage

L'étudiant·e effectuera les travaux dans les locaux d'OCamlPro, à l'adresse suivante :

```
OCamlPro SAS
21 rue de Châtillon
75014 Paris
```

📅 Dates

Le stage pourra avoir lieu entre février et septembre.

👩‍🏫 Encadrants

- Léo Andrès (Ph. D.), ingénieur R&D ;
- Steven de Oliveira (Ph. D.), ingénieur R&D.
- Nicolas Berthier (Ph. D.), ingénieur R&D.

Le télétravail est possible, généralement deux jours par semaine.

📜 Bibliographie

[1] Léo Andrès, Pierre Chambart, Filipe Marques, Eric Patrizio, Arthur Carcano, et Zhicheng Hui. 2021. Owi. Consulté à l'adresse https://github.com/ocamlpro/owi

[2] Léo Andrès, Filipe Marques, Arthur Carcano, Pierre Chambart, José Fragoso Femenin dos Santos, et Jean-Christophe Filliâtre. 2024. Owi: Performant Parallel Symbolic Execution Made Easy,

an Application to WebAssembly. *The Art, Science, and Engineering of Programming* 9, 2 (2024). https://doi.org/10.48550/arXiv.2412.06391

[3] Léo Andrès. 2024. Exécution symbolique pour tous ou Compilation d'OCaml vers WebAssembly.

[4] Sébastien Bardin, Nikolai Kosmatov, Michaël Marcozzi, et Mickaël Delahaye. 2021. Specify and Measure, Cover and Reveal: A Unified Framework for Automated Test Generation. *Sci. Comput. Program.* 207, (2021), 102641.

[5] Nicolas Berthier, Steven De Oliveira, Nikolai Kosmatov, Delphine Longuet, et Romain Soulat. 2023. An Efficient Black-Box Support of Advanced Coverage Criteria for Klee. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC~'23)*, mars 2023. ACM, 1706-1715. https://doi.org/10.1145/3555776.3577713

[6] Cristian Cadar, Daniel Dunbar, et Dawson Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *USENIX Conference on Operating Systems Design and Implementation*, 2008. https://doi.org/10.5555/1855741.1855756

[7] John Joseph Chilenski et Steven P. Miller. 1994. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal* 9, 5 (1994), 193-200. https://doi.org/10.1049/sej.1994.0025

[8] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, et JF Bastien. 2017. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017. 185-200. https://doi.org/10.1145/3062341.3062363

[9] Zhicheng Hui et Léo Andrès. 2025. Cross-Language Symbolic Runtime Annotation Checking. In *36es Journées Francophones des Langages Applicatifs (JFLA 2025)*, janvier 2025. Roiffé, France. Consulté à l'adresse https://inria.hal.science/hal-04798756

[10] James C. King. 1976. Symbolic Execution and Program Testing. *Communications of the ACM* (1976). https://doi.org/10.1145/360248.360252