Approche multi-solveur pour l'exécution symbolique Wasm

Léo Andrès
OCamlPro
leo@ocamlpro.com

Hichem Rami Ait El Hara
OCamlPro & Université Paris-Saclay, CEA, List
hra@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 appliqués à des programmes WebAssembly, C et Rust.

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].

Les différents moteurs d'exécution symbolique tels qu'Owi utilisent des solveurs SMT [13] pour tester la satisfiabilité des conditions correspondant aux différentes branches ainsi que pour générer un modèle correspondant aux valeurs d'entrées. Les solveurs SMT sont des prouveurs automatiques de formules mathématiques exprimées en une combinaison de différentes théories en logique du premier ordre [5]. Parmi ces théories, on trouve : l'arithmétique des entiers ou des réels, linéaire ou non linéaire, les vecteurs de bits, les nombres flottants, les tableaux fonctionnels et les chaînes de caractères. Grâce à leur expressivité et leur efficacité, différents outils de vérification logicielle les utilisent. OCamlPro développe son propre solveur SMT, Alt-Ergo, qui est utilisable par Owi en plus d'autres solveurs.

Sujet

Ce projet de recherche s'inscrit dans le prolongement des travaux menés sur Owi, avec pour objectif principal **l'intégration d'un support avancé pour plusieurs solveurs SMT** afin d'améliorer l'efficacité et la flexibilité de l'exécution symbolique. Les moteurs d'exécution symbolique s'appuient généralement sur des solveurs SMT tels que Z3 [12], Alt-Ergo [7], Colibri2 [18], Bitwuzla [14] ou CVC5 [4]. Owi ne communique pas directement avec ces solveurs, mais repose sur la bibliothèque Smt.ml [11, 16], qui offre une abstraction unifiée et transparente pour leur utilisation. Actuellement, toutes les expérimentations dans Owi se font avec Z3 comme solveur par défaut.

Cependant, des travaux antérieurs ont montré qu'une sélection contextuelle de solveurs, adaptée au programme analysé, permet d'obtenir des gains significatifs en performance [6, 15, 17, 19]. Ces résultats soulignent l'importance d'une stratégie fine de sélection de solveurs, adaptée aux caractéristiques du programme en cours d'analyse.

Dans cette optique, nous proposons de développer et d'intégrer une **passe d'analyse pour le classement des solveurs SMT**. Cette passe vise à prédire le solveur offrant les meilleures performances pour

[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.

un programme donné. Étant donné que Wasm est un langage bas niveau dont les instructions s'alignent souvent étroitement avec les concepts manipulés par les solveurs SMT, nous faisons l'hypothèse qu'une analyse statique du programme Wasm en amont de l'exécution symbolique pourrait suffire à guider efficacement le choix des solveurs, réduisant ainsi le besoin d'examiner directement les formules SMT générées.

Pour atteindre cet objectif, les étapes proposées sont les suivantes :

- **Validation des solveurs existants :** tester les solveurs SMT exposés par Smt.ml afin de s'assurer qu'ils prennent correctement en charge les requêtes générées par l'exécution symbolique.
- **Extension des solveurs supportés :** en fonction des besoins, intégrer de nouveaux solveurs dans Smt.ml pour enrichir les possibilités d'analyse.
- **Développement de la passe de classement :** implémenter une passe qui analyse le programme Wasm pour attribuer un score à chaque solveur au moyen d'heuristiques (par exemple en fonction des instructions qu'il contient).
- **Évaluation et benchmarks :** mesurer les performances de la solution développée en s'appuyant sur les benchmarks existants dans Owi.

Si le temps le permet, des explorations plus avancées pourraient être envisagées :

- **Stratégies dynamiques :** étudier des approches où le solveur peut être changé dynamiquement au cours de l'exécution symbolique, en fonction des branches explorées ou des types de requêtes.
- **Heuristiques selon le langage source :** développer des heuristiques rapides basées sur le langage source ayant produit le programme Wasm analysé.
- **Optimisations ciblées :** identifier les requêtes SMT les plus fréquentes à partir de statistiques sur les exécutions, et proposer des optimisations dédiées dans Smt.ml, Alt-Ergo ou Colibri2 pour ces cas spécifiques.
- **Cas particulier de la génération de modèles :** explorer la conception d'heuristiques afin de fournir un classement alternatif pour sélectionner le solveur utilisé à la fin de l'exécution pour la génération de modèle.

📚 Pré-requis

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

🏆 Compétences développées

- découverte approfondie de WebAssembly
- programmation OCaml avancée
- techniques d'exécution symbolique
- connaissance des solveurs SMT

📍 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 ;
- Hichem Rami Ait El Hara, 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] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et others. 2022. cvc5: A versatile and industrial-strength SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2022. 415-442.

[5] Clark Barrett, Pascal Fontaine, et Cesare Tinelli. 2016. The Satisfiability Modulo Theories Library (SMT-LIB).

[6] Luohui Chen, Yong Tang, Min Zhou, Shuning Wei, et Wenchuan Sun. 2023. Optimize value-flow analysis based static vulnerability detection by solver rating. In *International Conference on Computer Network Security and Software Engineering (CNSSE 2023)*, 2023. SPIE, 127140. https://doi.org/10.1117/12.2683173

[7] Sylvain Conchon, Albin Coquereau, Mohamed Iguernlala, et Alain Mebsout. 2018. Alt-Ergo 2.2. In *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, 2018.

[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

[11] Filipe Marques. 2023. Smt.ml. Consulté à l'adresse https://github.com/formalsec/smtml

[12] Leonardo De Moura et Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[13] Leonardo De Moura et Nikolaj Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (septembre 2011), 69-77. https://doi.org/10.1145/1995376.1995394

[14] Aina Niemetz et Mathias Preiner. 2023. Bitwuzla. In *International Conference on Computer Aided Verification*, 2023. 3-17.

[15] Hristina Palikareva et Cristian Cadar. 2013. Multi-solver support in symbolic execution. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*, 2013. 53-68.

[16] João Madeira Pereira, Filipe Marques, Pedro Adão, Hichem Rami Ait El Hara, Léo Andrès, Arthur Carcano, Pierre Chambart, Nuno Santos, et José Fragoso Santos. 2024. Smt. ml: A Multi-Backend Frontend for SMT Solvers in OCaml. (2024).

[17] Heinz Riener, Finn Haedicke, Stefan Frehse, Mathias Soeken, Daniel Große, Rolf Drechsler, et Goerschwin Fey. 2017. metaSMT: focus on your application and not on solver integration. *International Journal on Software Tools for Technology Transfer* 19, 5 (2017), 605-621.

[18] Colibri2 Team. Colibri2. Consulté à l'adresse https://colibri.frama-c.com/

[19] Sheng-Han Wen, Wei-Loon Mow, Wei-Ning Chen, Chien-Yuan Wang, et Hsu-Chun Hsiao. 2019. Enhancing symbolic execution by machine learning based solver selection. In *Proceedings of the NDSS Workshop on Binary Analysis Research*, 2019.