Prédiction de branchement pour l'exécution symbolique Wasm

Léo Andrès
OCamlPro
leo@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 **algorithmes d'apprentissage** 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) [4] 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** [6] 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, 5].

Sujet

Ce projet s'inscrit dans le prolongement des travaux sur Owi, un cadre d'exécution symbolique développé en OCaml. L'objectif principal est de développer des techniques de **classification de la satisfiabilité des points de branchement** afin de réduire les appels coûteux aux solveurs SMT et d'améliorer l'efficacité de l'exécution symbolique.

Lors de l'exécution symbolique, chaque point de branchement génère deux requêtes vers un solveur SMT [7] pour vérifier si la condition et sa négation sont satisfiables. Ces appels permettent d'éliminer les branches impossibles, réduisant ainsi le temps d'exécution. Cependant, les appels aux solveurs SMT représentent la majorité du temps d'exécution de l'analyse et il est important de réduire leur utilisation autant que faire se peut.

Des travaux récents [9], [8], [10] ont montré que des algorithmes d'apprentissage pouvaient être utilisés pour prédire la satisfiabilité des conditions de branchement, offrant une alternative aux solveurs SMT. Ce projet propose de développer et d'évaluer deux approches pour cette classification : une méthode d'apprentissage « online » et une méthode supervisée « offline ».

La première méthode est celle de l'apprentissage « online » et de la prédiction dynamique des branches. Elle s'appuie sur l'apprentissage en continu pour guider l'exploration des branches pendant l'exécution symbolique. Dans cette approche, le système utilise les chemins parcourus (breadcrumbs) pour prédire dynamiquement si une branche conditionnelle est satisfiable ou non, sans nécessiter de phase d'entraînement préalable. Lorsqu'une nouvelle conditionnelle est rencontrée, une prédiction est effectuée en temps réel. Si cette prédiction est jugée peu fiable, le système sollicite un solveur SMT pour garantir l'exactitude de l'exploration symbolique. Cette méthode offre une adaptation immédiate aux caractéristiques spécifiques du programme analysé, tout en réduisant le besoin de préparation

[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.

initiale, puisqu'elle ne dépend d'aucun jeu de données préexistant. Toutefois, son efficacité repose sur un équilibre délicat entre l'exploration via le solveur SMT et l'exploitation des prédictions. De plus, elle nécessite un mécanisme robuste pour intégrer ces prédictions en continu tout en évitant les erreurs coûteuses.

La seconde méthode est celle de l'apprentissage supervisé « offline » basé sur les conditions de chemin. Elle repose sur l'entraînement préalable d'un modèle supervisé à partir des conditions de chemin symboliques accumulées lors de l'exécution symbolique. Ce modèle apprend à prédire la satisfiabilité des branches en utilisant ces formules logiques comme données d'entrée. Une fois entraîné, il peut être intégré directement à l'exécution symbolique, permettant ainsi de minimiser les appels au solveur SMT. Cette méthode se distingue par sa rapidité et son efficacité, notamment lorsqu'elle bénéficie de jeux de données riches et représentatifs. Elle exploite les avancées du machine learning pour capturer des patterns complexes dans les formules symboliques. Cependant, elle dépend fortement de la qualité des données utilisées pour l'entraînement et peut être moins performante si les conditions de chemin du programme analysé diffèrent de celles observées pendant la phase de formation.

Les objectifs du stage seront les suivants :

- développer une des deux approches (« online » ou « offline ») ;
- mesurer la précision de la classification des branches ;
- évaluer l'impact sur les performances globales, notamment le gain en temps d'exécution (le temps de prédiction est-il négligeable ? comment se compare-t-il au coût des solveurs SMT ?) et l'impact des erreurs de classification sur la détection de bugs.
- Comparer les performances des deux approches si le temps le permet.

En fonction de l'avancement, d'autres axes pourront être explorés, comme le développement de l'approche qui n'a pas été choisie initialement ou bien la conception d'algorithmes d'apprentissage pour d'autres aspects de l'exécution symbolique, par exemple la priorisation des chemins lors de l'exploration.

📚 Pré-requis

- niveau M1/M2 informatique ou équivalent
- algorithmes d'apprentissages
- notions de programation fonctionnelle

🏆 Compétences développées

- découverte approfondie de WebAssembly
- techniques d'exécution symbolique
- méthodes modernes de classification

📍 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 ;
- Basile Clément (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] 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

[5] 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

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

[7] 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

[8] Junye Wen, Mujahid Khan, Meiru Che, Yan Yan, et Guowei Yang. 2020. Constraint solving with deep learning for symbolic execution. *arXiv preprint arXiv:2003.08350* (2020).

[9] Junye Wen, Tarek Mahmud, Meiru Che, Yan Yan, et Guowei Yang. 2023. Intelligent Constraint Classification for Symbolic Execution. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2023. 144-154. https://doi.org/10.1109/SANER56733.2023.00023

[10] Mingyue Yang, David Lie, et Nicolas Papernot. 2024. Exploring Strategies for Guiding Symbolic Analysis with Machine Learning Prediction. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2024. 659-669.