Proposition de stage niveau M1 ou M2 : optimisation de code WebAssembly pour l'exécution symbolique

Léo Andrès, Pierre Chambart, Arthur Carcano

1 Modalités

Lieu du stage Le stage aura lieu à l'adresse suivante :
Équipe WebAssembly
OCamlPro SAS
21 rue de Châtillon
75014 Paris

Dates Le stage durera entre trois et six mois.

Encadrants

- Léo Andrès, ingénieur R&D et doctorant - OCamlPro et Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles.
- Arthur Carcano (PhD), ingénieur R&D - OCamlPro.
- Pierre Chambart (PhD), CTO et ingénieur R&D sénior - OCamlPro.

2 Contexte

Le contexte de ce stage est celui de l'analyse statique de programmes WebAssembly (Wasm) [6] et plus précisément de l'outil Owi [1] développé chez OCamlPro dans l'équipe Wasm en collaboration avec l'INESC-ID/Instituto Superior Técnico de l'Université de Lisbonne. Avec cet outil, l'utilisateur peut exécuter symboliquement des programmes Wasm et obtenir un modèle donnant les valeurs des entrées pour lesquelles le programme lève une erreur. Il est également

possible d'exécuter symboliquement du code d'autres langages tels que Rust ou C que l'outil se chargera de compiler vers Wasm. L'implémentation de l'exécution symbolique dans Owi est décrite dans [2].

3 Sujet

Un optimiseur de code Wasm, `owi opt`, est développé dans le cadre d'Owi. Il ne propose pour l'instant que des optimisations simples (propagation de constantes, élimination de code mort) implémentées de façon naïves. L'objectif de ce stage est d'identifier ou de concevoir des optimisations pouvant être bénéfiques à l'exécution symbolique.

En lançant Owi sur Test-Comp (un ensemble de benchmarks comprenant plus d'un millier de fichiers C avec chacun un bug à trouver), nous avons constaté qu'optimiser le code C au moment de la compilation vers Wasm (avec `-O3`) permet d'obtenir de bien meilleurs résultats - notamment parce que certaines optimisations réduisent le nombre de branchements, ce qui est bénéfique pour l'exécution symbolique puisque cela permet de limiter l'explosion combinatoire.

Une première étape du stage sera d'expérimenter avec les différentes optimisations disponibles dans `clang` et d'observer leur impact sur l'exécution symbolique. Pour l'instant, nous utilisons `-O3` qui contient de nombreuses optimisations dont on ne sait pas si elles sont toutes bénéfiques.

Il sera ensuite possible d'implémenter des optimisations dédiées à Wasm. Elles pourront par exemple :

- s'inspirer de langages à pile : Lisp [8], Forth [3] ou STAL [9] ;
- passer par des représentations plus générales telles que SSA [5, 10] (en commençant par s'intéresser au graphe de flot de contrôle [7]) ou Sea of Nodes [4] ;
- être tirées des optimisations implémentées dans Binaryen (un optimiseur de code Wasm) [11].

Il sera également possible de prouver que certaines optimisations préservent la sémantique du programme source ou bien de montrer des propriétés de confluence sur l'ordre d'application de ces optimisations.

4 Prérequis

- programmation OCaml

— techniques de compilation
— algorithmique
— sémantique opérationnelle

Références

[1] Léo Andrès, Pierre Chambart et Filipe Marques. *Owi*. 2021. url : `https://github.com/ocamlpro/owi`.

[2] Léo Andrès et al. « Owi : Performant Parallel Symbolic Execution Made Easy, an Application to WebAssembly ». working paper or preprint. Juin 2024. url : `https://hal.science/hal-04627413`.

[3] Christopher Bailey. « Optimisation Techniques for Stack Based Architectures ». Thèse de doct. University of Teesside, 1996.

[4] Cliff Click et Keith D Cooper. « Combining analyses, combining optimizations ». In : *ACM Transactions on Programming Languages and Systems (TOPLAS)* 17.2 (1995), p. 181-196.

[5] Ron Cytron et al. « An efficient method of computing static single assignment form ». In : *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 1989, p. 25-35.

[6] Andreas Haas et al. « Bringing the web up to speed with WebAssembly ». In : *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2017, p. 185-200.

[7] Daniel Lehmann et al. « That'sa Tough Call : Studying the Challenges of Call Graph Construction for WebAssembly ». In : *Symposium on Software Testing and Analysis (ISSTA'23)*. 2023.

[8] Larry M Masinter et L Peter Deutsch. « Local optimization in a compiler for stack-based Lisp machines ». In : *Proceedings of the 1980 ACM conference on LISP and functional programming*. 1980, p. 223-230.

[9] Greg Morrisett et al. « Stack-based typed assembly language ». In : *International Workshop on Types in Compilation*. Springer. 1998, p. 28-52.

[10] Barry K Rosen, Mark N Wegman et F Kenneth Zadeck. « Global value numbers and redundant computations ». In : *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 1988, p. 12-27.

[11] WebAssembly Community Group participants. *Binaryen*. 2015. URL : https://github.com/WebAssembly/binaryen.