Sorbonne Université

PSTL
Rapport

Un Langage "Pur" pour Web Assembly

Élève :
Lucas Fumard
Lauryn Pierre
Saïd Mohammad ZUHAIR

Enseignant :
Frédéric Peschanski

24 mars 2023

Table des matières

1 Introduction

WebAssembly ou Wasm [5] est un nouveau langage bytecode pris en charge par les principaux navigateurs Web, conçu principalement pour être une cible de compilation efficace pour les langages de bas niveau tels que C/C++ et Rust[1]. WebAssembly a un format texte(.wat) et un format binaire(.wasm). Dans le cas de notre projet on utilise wabt[4] pour compiler et on exécute le compilé à l'aide de Node.js [2]. Le but de notre projet est de concevoir un langage 100 % fonctionnel et "pur" pour WebAssembly en se basant sur le langage défini dans cet article[3]. L'article défini un langage fonctionnel dont la gestion de la mémoire se fait par un mécanisme de comptage de références.

2 Concepts fondamentaux

Dans cette partie, nous allons présentons les différentes notions nécessaire à la compréhension du projet.

2.1 WebAsssembly

2.2 Présentation de l'article

3 Cahier des charges

Les tâches que nous avons identifié sont les suivantes :
- Analyser le fonctionnement de WASM
- Programmer un parseur qui puisse lire le langage pur tel que défini dans l'article[3]
- Programmer un interpréteur en Rust du langage selon les sémantiques du langage pur
- Définir quelques tests unitaires couvrant les sémantiques définies dans l'article
- Ajouter la gestion des instructions `inc`, `dec`, `reset`, `reuse`
- Programmer un compilateur du langage agrandi vers WASM

4 Tâches Réalisées

Afin de créer le langage pur pour WASM, nous avons tout d'abord lu l'article[3] et analysé la structure du langage pur à implémenter telle que définie dans la section 3 de l'article. Cette structure nous a permis d'écrire un parseur en Rust capable de créer un AST du langage.

Par la suite, nous avons implémenté les sémantiques définies dans la section 4 sur la figure 1 en programmant un interpréteur. Nous avons testé cet interpréteur en créant plusieurs tests unitaires sur les sémantique mais aussi quelques programmes simples, tel que le calcul de fibonacci.

5 Tâches Restantes

Il nous faut choisir un schéma de mémoire pour les objets dans la mémoire. Il nous faut implémenter `inc`, `dec`, `reset`, `reuse`. Il nous faut faire le compilateur en WASM

WASM n'a qu'un seul type de variable : les nombres (entiers ou flottant, 32 bit ou 64 bit). Ainsi, il faut que l'on interprète différement ces nombres selon le contexte dans lequel on les prend. Dans la mémoire, nos constructeurs et applications partielles seront stoqués sous la forme `<type> <nombre de références> <arguments>`. Par exemple, une application partielle est sous la forme `6 1 7 2 12 13` où 6 est le type "application partielle", 1 est le nombre de références à cette application partielle, 7 est l'identifiant de la fonction à appeler (dans l'interpréteur, on utilise des `string`), 2 est le nombre d'arguments fixés et 12 et 13 sont les arguments fixés.

Dû à la difficulté d'une telle tâche, nous laissons l'implémentation des applications partielles pour plus tard.

Références

[1] 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*. PLDI 2017. New York, NY, USA : Association for Computing Machinery, 14 juin 2017, p. 185-200. ISBN : 978-1-4503-4988-8. DOI : 10.1145/3062341.3062363. URL : https://dl.acm.org/doi/10.1145/3062341.3062363 (visité le 16/03/2023).

[2] *Node.js Et WebAssembly*. Node.js Et WebAssembly. Section : Node.js. URL : https://nodejs.dev/fr/learn/nodejs-with-webassembly/ (visité le 16/03/2023).

[3] Sebastian Ullrich et Leonardo de Moura. *Counting Immutable Beans : Reference Counting Optimized for Purely Functional Programming*. 5 mars 2020. DOI : 10.48550/arXiv.1908.05647. arXiv : 1908.05647[cs]. URL : http://arxiv.org/abs/1908.05647 (visité le 08/03/2023).

[4] *WABT : The WebAssembly Binary Toolkit*. original-date : 2015-09-14T18 :14 :23Z. 16 mars 2023. URL : https://github.com/WebAssembly/wabt (visité le 16/03/2023).

[5] *WebAssembly*. URL : https://webassembly.org/ (visité le 12/02/2023).