Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière Informatique et Systèmes de Communication
Orientation Informatique Logicielle

Cahier des charges
Travail de Bachelor
2025

Modèles numériques alternatifs pour WebAssembly par instrumentation de code

Luca Michel
luca.michel@edu.hefr.ch

Superviseur: Frédéric Bapst

Expert: Baptiste Wicht

Fribourg, 22 mai 2025
Version 0.1.0

Hes·so

Table des versions

Version	Date	Auteur(s)	Modifications
0.1.0	22/05/2025	LM	Version initiale

Table des matières

Chapitre 1
Contexte

WebAssembly (Wasm) est un langage bas niveau similaire à l'assembleur qui permet d'exécuter du code écrit depuis différents langages dans un navigateur web avec des performances similaires aux applications natives [1]. Cette technologie en plein essor est de plus en plus utilisée dans des contextes d'applications web grâce à sa performance et sa portabilité, mais il est également possible de faire fonctionner du code Wasm localement. Dans un contexte web, Wasm fonctionne de pair avec JavaScript (JS) qui permet de charger des modules Wasm au sein d'une application JS. Cela permet de donner accès à la flexibilité de JS et la performance de Wasm [1].

Le code Wasm fonctionne à la manière du code assembleur, c'est-à-dire qu'il s'appuie sur une machine virtuelle à pile capable d'exécuter du code binaire. L'environnement de cette machine virtuelle est décrit comme efficace, *memory safe*, *sandboxed*,[1] ouvert et *debuggable* [2].

Cependant ce langage, comme beaucoup d'autres, est sensible aux anomalies arithmétiques usuelles. Ces dernières peuvent avoir des conséquences catastrophiques comme par exemple l'*Ariane flight V88* [3] ou la *VM escape via VGA device* [4]. Ces conséquences sont souvent dûes à un manque de mesures de détection de ces anomalies.

COJAC [5] vise à détecter et à pallier aux anomalies arithmétiques pouvant être rencontrées durant l'exécution d'un programme Java. Cet utilitaire a inspiré un projet de semestre réalisé par M. Julmy visant à implémenter une *proof of concept* (*POC*) démontrant la possibilité d'implémenter les fonctionnalités de COJAC dans un contexte Web, à l'aide de Wasm.

Compte tenu de la présence grandissante de Wasm dans les nouvelles applications, il est intéressant de fournir un outil permettant de détecter et de pallier à ces anomalies. Ce projet se concentre donc sur l'extension de la *POC* de M. Julmy.

[1] Contraint à l'espace mémoire qui lui est attribué, sans possibilité d'interagir avec le reste du système.

Chapitre 2
Objectifs

La *POC* de M. Julmy permet d'instrumenter du code Wasm à la volée afin d'altérer le comportement des opérations instrumentées. Ces opérations sont substituées par des opérations de détection d'anomalies arithmétiques suivies des opérations initiales, ce qui permet de vérifier que les opérations se déroulent correctement sans altérer leur comportement.

Cette instrumentation est réalisée à l'aide de la librairie WAIL [6] et permet actuellement uniquement la détection des *integer overflow* et les résultats infinis sur les nombres à virgule flottante pour les opérations d'addition et de soustraction.

Afin de compléter cette fonctionnalité (appelée *numerical sniffer*), le support pour les opérations de multiplication et de division sera implémenté. De plus, les anomalies suivantes seront également détectées :

- *cancellation*
- *absorption*
- *questionable comparisons*
- *underflow*

La logique actuelle, illustrée par la Fig. 1, exécute la détection des anomalies en Wasm et appelle conditionnellement une fonction de signalisation JS.

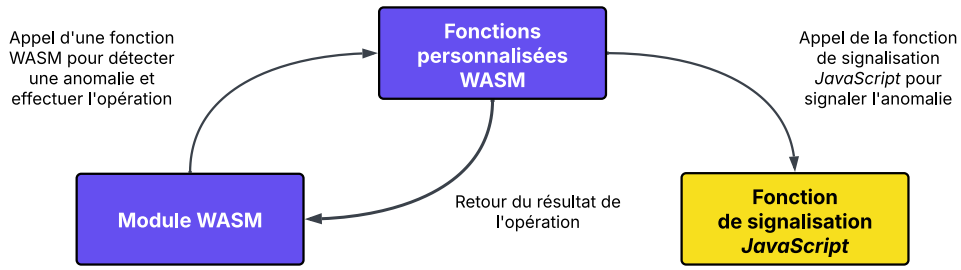


Fig. 1. – Déroulement de l'instrumentation et de la détection d'anomalies

Ce comportement était basé sur l'hypothèse que les opérations devaient être effectués dans le même langage que les opérations d'origine afin de garantir leur bon fonctionnement. Cependant, Wasm et JS suivent tous deux les mêmes normes IEEE, ce qui rend ce traitement superflu. Dans le but d'uniformiser le comportement du programme et d'augmenter sa maintenabilité, ce projet modifiera l'instrumentation des opérations ciblées afin qu'une seule fonction JS soit appelée et cette dernière effectuera les contrôles d'anomalies.

Les anomalies actuellement détectées sont signalées durant l'exécution du programme, sans indication quant à leur localisation dans le code. Afin de permettre au développeur de réagir à ces anomalies si c'est pertinent, le programme indiquera au moins le nom de la méthode concernée, dans les limites des capacités de Wasm.

Afin de potentiellement économiser de l'espace mémoire, COJAC propose une fonctionnalité *double-as-float* qui teste si des calculs entre *double* peuvent être faits avec des

floats sans perte de précision. Cette fonctionnalité sera également implémentée durant ce travail de bachelor.

Enfin, COJAC propose une fonctionnalité permettant de *wrap* les nombres dans des structures de nombres plus riches afin de garantir une plus grande robustesse. À l'heure de l'écriture de ce cahier des charges il n'est pas certain que cette fonctionnalité soit réalisable, cependant une étude de l'approche nécessaire à son implémentation sera relatée dans le rapport de ce travail de bachelor. Si la fonctionnalité est réalisable alors une *POC* de cette dernière se trouvera dans le livrable final.

Les objectifs de ce projet sont donc les suivants :

- Modifier la logique d'instrumentation afin de n'appeler qu'une fonction JS
- Signaler la localisation relative des anomalies dans le code source
- Instrumenter les multiplications et divisions en plus des additions et soustractions
- Compléter le *numerical sniffer* avec les anomalies suivantes :
  - *cancellation*
  - *absorption*
  - *questionable comparisons*
  - *underflow*
- Implémenter la fonctionnalité *double-as-float* de COJAC
- Étudier et implémenter (si possible) la fonctionnalité *wrapper* de COJAC

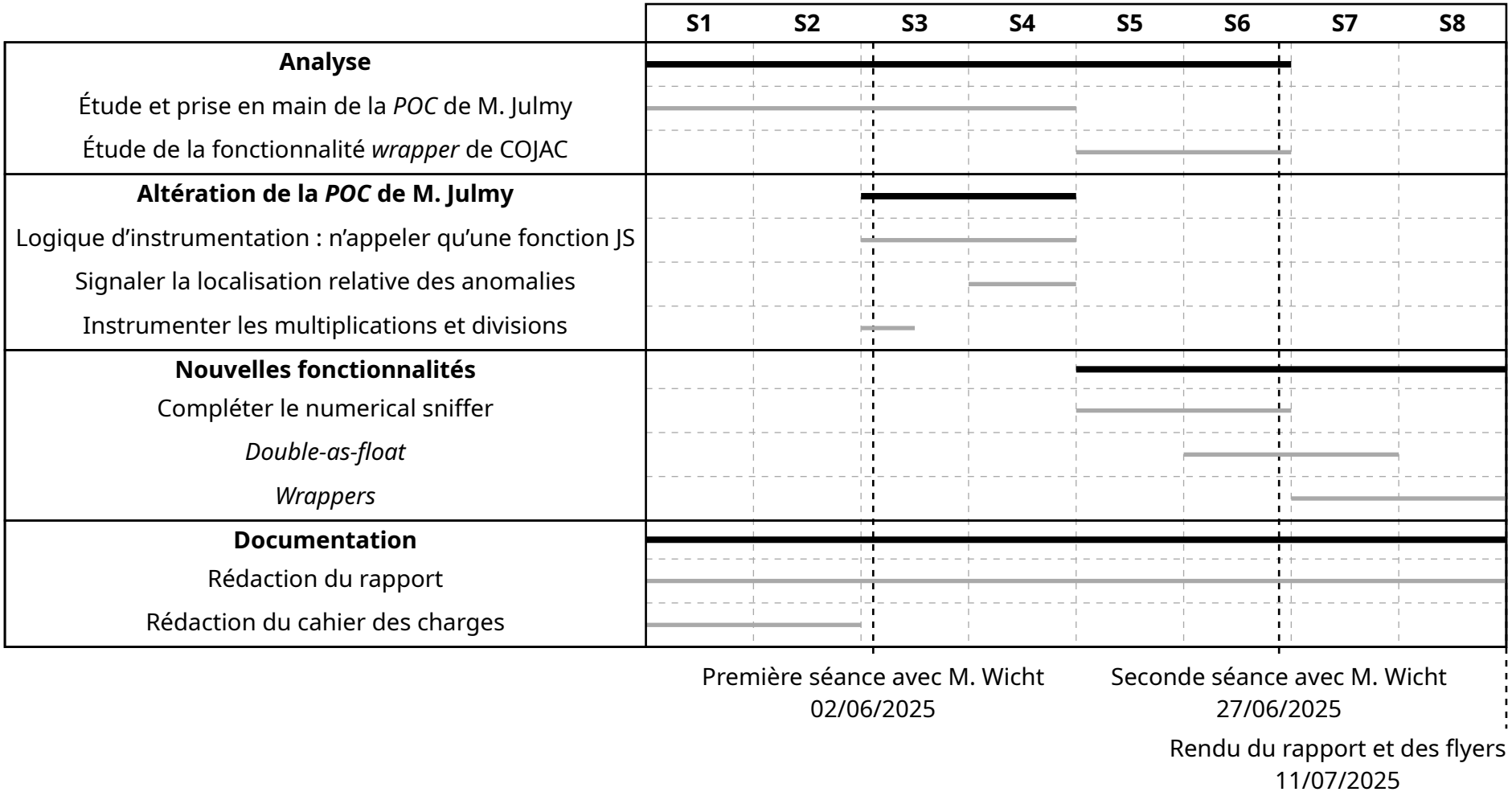
De plus un rapport documentant les processus d'implémentation, les notions abordées et l'état du projet à la fin du travail de bachelor sera rédigé.

Chapitre 3

Tâches

Chapitre 4

Planning


S1
S2
S3
S4
S5
S6
S7
S8
Analyse
Étude et prise en main de la POC de M. Julmy
Étude de la fonctionnalité wrapper de COJAC
Altération de la POC de M. Julmy
Logique d'instrumentation : n'appeler qu'une fonction JS
Signaler la localisation relative des anomalies
Instrumenter les multiplications et divisions
Nouvelles fonctionnalités
Compléter le numerical sniffer
Double-as-float
Wrappers
Documentation
Rédaction du rapport
Rédaction du cahier des charges
Première séance avec M. Wicht
02/06/2025
Seconde séance avec M. Wicht
27/06/2025
Rendu du rapport et des flyers
11/07/2025

Figures

Chapitre 5
Glossaire

absorption: Anomalie arithmétique survenant lorsque deux nombres à virgule flottante d'un ordre de grandeur excessivement différent sont additionnés ou soustraits sans effet. 2, 3

cancellation: Anomalie arithmétique survenant lorsque deux nombres à virgule flottante dont la valeur absolue est presque similaire sont additionnés ou soustraits. Cette anomalie a pour effet de rendre les bits de poids faible plus signifiants sur le résultat alors que ceux-cis sont souvent plutôt du bruit. 2, 3

COJAC: Utilitaire Java développé à l'HEIA-FR permettant de détecter et de pallier aux anomalies arithmétiques pouvant être rencontrées à l'exécution d'un programme Java. 1, 1, 2, 3, 3, 3, 6

integer overflow: Anomalie arithmétique survenant lorsqu'un nombre entier hors du domaine des valeurs permises est représenté. 2

JS – **JavaScript**: Langage de scripting interprété utilisé notamment dans le contexte de applications web. 1, 1, 1, 2, 2, 2, 3, 6

POC – ***proof of concept***: Utilitaire Java développé à l'HEIA-FR permettant de détecter et de pallier aux anomalies arithmétiques pouvant être rencontrées à l'exécution d'un programme Java. 1, 1, 2, 3, 6, 6

questionable comparisons: Anomalie arithmétique survenant lorsque deux nombres à virgule flottante très proches sont comparés. 2, 3

underflow: Anomalie arithmétique survenant lorsque le résultat de la division de deux nombres à virgule flottante est si petit qu'il est arrondi à zéro. 2, 3

Ariane flight V88 ***[3]***: Décolage de fusée ayant échoué à cause de plusieurs erreurs de conception du code, notamment un *overflow*. 1

VM escape via VGA device ***[4]***: Décolage de fusée ayant échoué à cause de plusieurs erreurs de conception du code, notamment un *overflow*. 1

WAIL: Librairie d'instrumentation de code Wasm. 2

Wasm – **WebAssembly**: Format d'instructions binaires basé sur une machine virtuelle à pile. 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2

Bibliographie

[1] 2025. Documentation de Mozilla concernant Wasm. Consulté à l'adresse https://developer.mozilla.org/fr/docs/WebAssembly

[2] 2025. Site officiel de Wasm. Consulté à l'adresse https://webassembly.org/

[3] 2025. Page wikipedia du vol Ariane V88. Consulté à l'adresse https://en.wikipedia.org/wiki/Ariane_flight_V88

[4] 2025. Rapport de sécurité concernant une vulnérabilité de Oracle VM VirtualBox permettant d'exécuter du code sur la machine d'hôte. Consulté à l'adresse https://en.wikipedia.org/wiki/Ariane_flight_V88

[5] 2025. Repository GitHub de COJAC. Consulté à l'adresse https://github.com/Cojac/Cojac?tab=readme-ov-file

[6] 2025. Repository GitHub de la librairie WAIL. Consulté à l'adresse https://github.com/Qwokka/WAIL

Déclaration d’honneur

Je, soussigné, Luca Michel, déclare sur l’honneur que le travail rendu est le fruit d’un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d’information utilisées et les citations d’auteur ont été clairement mentionnées.