Rapport Niveau 3 – PS5 BRAINF-CK

MIAOU – MIAOU Is An Open Umbrella
Nassim BOUNOUAS - Guillaume CASAGRANDE
Julien LEMAIRE - Pierre-Emmanuel NOVAC

26 novembre 2016

1 Introduction

Nous vous présentons à travers ce rapport notre version de l'interpréteur du langage de programmation exotique Brainf*ck développée à ce jour jusqu'à la fin du niveau 3, suivant le découpage du sujet d'origine. Nous avons donc repris les fonctionnalités dont le développement a été achevé dans les deux précédents niveaux, à savoir :

- La mise en place d'une `Machine` virtuelle et d'une mémoire (`Memory`).
- Le support des différentes `Instruction`s proposée par le langage Brainf*ck (`INCR`, `DECR`, `LEFT`, `RIGHT`, `IN`, `OUT`, `JUMP`, `BACK`).
- La redirection des flux d'entrée et de sortie pour que les instructions `IN` et `OUT` puissent respectivement lire et écrire dans des fichiers plutôt que sur l'entrée/la sortie standard.
- Le support des sauts conditionnels dans une implémentation naïve (décrite plus tard) et de la vérification de leur bonne utilisation dans un programme donnée.
- Le support des différentes syntaxes décrites dans le sujet, à savoir la syntaxe image (les instructions étant alors codées sous la forme de couleurs de 3 x 3 pixels), la syntaxe texte dite "longue" (les instructions sont alors écrites ligne par ligne, sous la forme de leur nom suscité) et la syntaxe texte dite "courte" (la syntaxe la plus connue, agrémentée de symboles).
- Le module de traduction (`Translator`) afin de pouvoir passer d'une syntaxe à une autre.

Ces fonctionnalités étant implémentées, nous devions alors y rajouter celles du niveau 3. Celles-ci étaient, entre autres, composées des `Metrics` qui, à chaque lancement d'un programme Brainf*ck, propose des données sur son exécution, ou encore du `Logger` qui, à la demande de l'utilisateur, devait fournir un fichier log, lié au programme exécuté, et détaillant les différentes instructions effectuées avec leur conséquence propre sur la `Machine` et la `Memory`. Le support des commentaires et de l'indentation permettaient de rentre un programme Brainf*ck aussi libre et compréhensible que l'auteur l'eusse souhaité.

En dehors de ces outils, plus de l'ordre de la maintenance d'un programme Brainf*ck et de sa clarté, l'ajout du support des `Macros` était réellement une vraie fonctionnalité supplémentaire pour le développeur Brainf*ck, qui pourrait alors sauvegarder un morceau de code sous l'appellation de son choix. Chaque appel de cette même appellation serait alors remplacé par le code lui-même lors de la lecture du fichier. Enfin, la `JumpTable` est apparue comme une amélioration potentiellement considérable de notre implémentation naïve des sauts conditionnels, permettant de lier chaque instruction `JUMP` à l'instruction `BACK` associée et inversement.

Proposer une implémentation de ces différentes fonctionnalités était donc notre objectif pour terminer ce niveau 3. Cet objectif étant à présent atteint, nous vous présenterons dans ce rapport nos différents choix d'implémentations pour le réaliser.

2 Nos choix d'implémentations

2.1 Schéma d'implémentation / Diagramme de classes

Nous avons implémenté les différentes classes de cette façon :

2.2 Metrics

Comme précisé en introduction, les `Metrics` sont des données fournies à chaque interprétation d'un programme Brainf*ck à propos de celui-ci et de son exécution. Ces `Metrics` sont composés de 6 valeurs, à savoir :

- `PROG_SIZE`, qui contient le nombre d'instructions écrites dans le programme interprété.
- `EXEC_TIME`, qui donne le temps d'exécution du programme en ms.
- `EXEC_MOVE`, qui donne le nombre de fois que le pointeur d'exécution (entendons par là, le pointeur sur les instructions à exécuter) a été déplacé.
- `DATA_MOVE`, qui donne le nombre de déplacements du pointeur sur la mémoire lors de l'exécution du programme.
- `DATA_READ`, qui donne le nombre d'opérations de lecture de la mémoire effectuées par le programme.
- `DATA_WRITE`, qui donne le nombre d'opérations d'écriture dans la mémoire effectuées par le programme.