|  |  |
| --- | --- |
|  | Rapport informatique |
|  |  |
| 25/10/2016 | Programmation fonctionnelle et interprêteur Scheme : Livrable n°2 |
|  |  |

# TABLE DES MATIERES

I. Introduction 3

II. Spécifications 4

II.1 Fonction sfs\_eval et dérivées de aux\_eval.c 4

III. Implémentation 5

III.1 Etat du logiciel : ce qui fonctionne, ce qui ne fonctionne pas 5

III.3 Les optimisations et les extensions réalisées 5

IV. Suivi 6

IV.1 Problèmes rencontrés 6

IV.2 Planning effectif 6

V. Conclusion 7

# Introduction

L’objectif de cette seconde partie est de mettre en place, sur ce qui est déjà acquit, la gestion des environnements, l’ajout de l’évaluation des formes et des jeux de tests. Les améliorations du livrable I n'ont pas toutes été effectuées mais 23/26 tests évolués fonctionnent (les 3 restants concernent les infinis, les verylongchar et les lignes très longues).

Dans ce document, nous allons essayer de décrire au mieux notre livrable 2 : ce qui fonctionne, ne fonctionne pas et pourquoi. Nous verrons également brièvement les problèmes rencontrés, les solutions envisagées.

# Spécifications

Notre deuxième livrable est composé de trois fichiers principaux : eval.c, aux\_eval.c et is\_functions.c

### II.1 Fonction sfs\_eval et dérivées de aux\_eval.c

La fonction sfs\_eval récupère la sortie de sfs\_read, on a alors un arbre correctement formé. sfs\_eval gère les formes define, set!, if, and, or mais aussi les opérateurs de calcul et de comparaison simples. Grâce à la récursivité croisée, on évalue chaque élément de manière dépendante en effectuant une gestion d'erreur bien plus poussée que dans le livrable 1. Rien n'est encore parfait et certains cas ne sont pas traités. Dans aux\_eval.c, on retrouve toutes les fonctions nécessaires à la gestion des différents symboles du langage Scheme et certains atomes qui demandent une gestion particulière. On renvoie alors un object évalué dans le programme qui sera lu par sfs\_print.

### II.2 Fonctions environnementales et formes associées

Les formes define et set appellent des fonctions environnementales d'ajout de variable et de changement de valeur dans un environnement. L'environnement principal est créé comme un extern object et initialisé dans repl.c comme une paire avec un nil sur chaque branche. Cet objet est le toplevel et stocke toutes les variables du niveau d'initialisation.

La fonction d'ajout de variable stocke une nouvelle variable en tête de toplevel avec sa valeur. Celle qui change la valeur, recherche dans l'environnement la dite variable.

L'objet symbol n'est pas auto-évaluant donc on recherche la variable dans l'environnement et on renvoie sa "valeur" grâce à object eval\_symbol.

Une fonction non utilisée de création d'environnement par dessus le toplevel a été faite et sera utile dans l'incrément III.

# Implémentation

### III.1 Etat du logiciel : ce qui fonctionne, ce qui ne fonctionne pas

Tous les tests\_step1/simple/ fonctionnent de manière correctes et 18/26 des ../evolved/. Ce qui ne fonctionne pas relève de la poussée à l'extrême du programme (très grande string non gérée encore ou entier de type int alors qu'on lui envoie un long long int). Le but pour ses prochaines semaines est de réparer toutes ces erreurs et d'optimiser le programme déjà écrit.

### III.3 Les optimisations et les extensions réalisées

Par manque de temps pour de diverses raisons, aucune optimisation n’a été mise en place.

# Suivi

### IV.1 Problèmes rencontrés

Aucun réel problème n’a été rencontré lors de l’écriture des programmes (seulement des segmentation fault ou autres erreurs classiques qu’on aurait pu éviter en réfléchissant plus en amont). La difficulté était surtout de traiter tous les cas sans en oublier.

### IV.2 Planning effectif

Le planning a été peu respecté et le travail très mal réparti. Pour ce qui est des détails, la correction des erreurs nous a parfois pris plus de temps que prévu.

# Conclusion

Il reste à faire marcher tous les tests evolved puis se mettre au boulot pour l'implément 2 sans prendre de retard.