Programmation fonctionnelle et interpréteur Scheme

Lucien Dos Santos
Mohamed Hage Hassan

October 17, 2016

Table des matières

1 Introduction **2**

2 Implémentation du mode d'évaluation **2**

3 Environnements **3**

1 Introduction

Ce rapport porte sur les travaux effectués pour la mise en place de la deuxième partie de l'interpréteur scheme. Cette partie consiste à programmer l'ensemble des fonctions qui vont remplacer la représentation tout simple par une évaluation, ainsi que des différents environnements (locaux et global) qui les différents variables, et surtout les formes essentielles du language comme *quote, define, set! et if.*

On rappelle de nouveau que le code qu'on programme **ne dépend pas** de l'embryon du code fourni dans la partie 1. Notre démarche de travail va être de conséquence bien différente de celle qui est recommendée.

Il faut aussi noter que le code est programmé d'une façon incrémentatle, par effet, le code fourni dans le livrable 2 va être rattaché à celui de la première partie.

2 Implémentation du mode d'évaluation

On va parler tout d'abord du process pour le remplacement de la représentation par une partie dévalutation des fonction.
Comme on l'a indiqué précedement, le code est très modulaire et réparti en **parseurs**, chaqu'un sert pour une fonction donnée (parseur de chaîne de charactères, des entiers, des charatères...), qui sont ainsi groupés dans des fichiers séparés. En premier temps, le code était dépendant de la fonction `represent()` (présente dans tous les parseurs).

Les fonctions `evaluate()` et `evaluate_on()` étaient déjà existantes dans le code de la partie 1, mais non implémentées. Prenons par exemple le parseur **MgIdentifier.c**

```
1 static MgStatus* not_implemented_too_late(void) {
    return Mg_error_not_implemented;
3 }
```

Le travail de cette partie était réparti pour la mise en place correcte de ces 2 fonctions **pour tout les parseurs**, reprenons l'exemple du parseur de symbols (**MgIdentifier**) :

```
1 static MgStatus* evaluate(MgIdentifier* self, MgObject** output, MgInterpreter*
      interpreter, MgEnv* env){
    // Code
3 }

5 static MgStatus* evaluate_on(MgIdentifier* self, MgIdentifier** output){
    // Code
7 }
```

Les structures de données associées sont aussi modifiées (*MgObjectType type* = {}).

3 Environnements

La méthodologie utilisée globalement dans la production du code est de implémenter toutes les fonctionnalités possibles d'une manière modulaire (et en fichier séparé) . Ce travail est reparti par binôme, et ça sert à prendre en compte facilement des futures implémentation ou expantion de nouvel code.
Prenons par exemple le cas de l'implémentation des environnements : celle-ci ce présente par le fichier `MgEnvironment.c` present dans le répertoire `src/core/`. Cette procédure crée les fonctions `MgEnv_create(), MgEnv_destroy(), MgEnv_add_identifier()..)` qui ne sont d'autre que des wrappeurs pour **les manipulations de listes**.

La technique de programmation consiste à programmer le même fichier (ou même fonctionnalité) simultanément, de façon que si un des binôme bloque sur un problème, on peut voir tous les deux les moyens pour le résoudre (ou pour la proposition d'idées).
L'idée des environements nestés ce fait principalement avec la fonction `MgEnv_create()` :

```
1 MgStatus* MgEnv_create(MgEnv** env, MgEnv* parent_env)
```

On rappelle que le type de structure MgEnv n'est que **MgList** (manipulation de listes). Les autres fonctions de l'environnement peuvent accéder à cette fonctionnalité de nesting des environnements avec `MgEnv** env` et `int scope_limited`. Le fichier `.h` est associé pour les définitions de prototypes.
La manipulation directe des environnements se fait notamment dans le parseur **MgIdentifier**, surtout la fonction `evaluate()` qui fait appelle vers des fonctions de manipulation d'environnement.

Pour une prise en compte correcte de l'addition des environnements, on a modifié les fichiers `MgObject.c, MgIdentifier.c, MgList.c, MgString.c, Mgpair.c`.

Il faut aussi noté que pour améliorer le code de façon générale, on a ajouté le module **MgInterpreter.c**, ce qui nétoie la fonctionnalité de **Interactive.c** (On déplace la plupart des fonctions inclus dans ce fichier dans **MgInterpreter.c** en récrivant quelque fonctions).