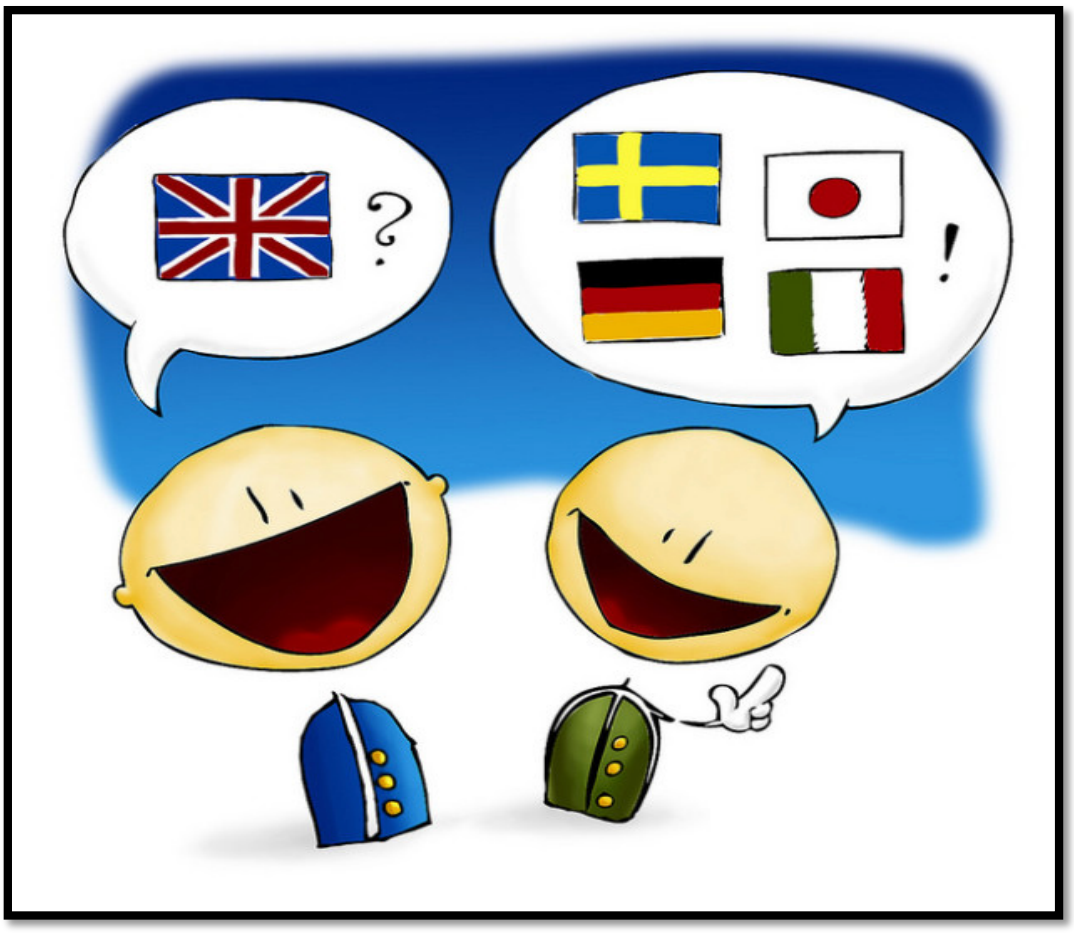
RAPPORT INFORMATIQUE



25/10/2016

Programmation fonctionnelle et interpréteur
Scheme : Livrable n°3

HRAMAN Jérémy & WALCH-GUINEBERT Romain

TABLE DES MATIERES

I. INTRODUCTION

L'objectif de cette troisième partie est de mettre en place, sur ce qui est déjà acquis, l'amélioration de gestion des environnements dans les fonctions, les primitives avec les pointeurs de fonctions, quelques adaptations des incréments précédents pour se mettre à jour avec nos nouvelles normes syntaxiques.

Dans ce document, nous allons essayer de décrire au mieux notre livrable 3 : ce qui fonctionne, ne fonctionne pas et pourquoi. Nous verrons également brièvement les problèmes rencontrés, les solutions envisagées.

II. SPECIFICATIONS

Notre troisième livrable est composé d'un fichier principal `primitive.c` et des changements fondamentaux dans le fichier : `aux_eval.c`.

II.1 Avant propos

Forme quote : La forme `quote` a été implémentée avec l'apostrophe. Elle fonctionne de manière optimale et a changé la manière de lire l'arbre grâce à `sfs_read`. Pour cela, on crée une pair directement en lecture avec en `car` la forme quote et en `cdr`, l'object à quoter.

Nombres réels : Avec l'implémentation de primitives tels que cos, sin, etc … l'ajout des nombres réels était devenu nécessaire. `type_input` (`aux_read.c`) et `read_atom_number` (`read_atom.c`) ont profondemment été modifiés pour accepter ces réels distinguant les deux cas. Une meilleure gestion des nombres a enfin été faite grâce à la fonction `strtod`.

SFS_NOTYPE : Pour les formes `define` et `set!`, nous retournions `NULL`, ce qui provoquait des `abort error` en mode script, elles retournent désormais un object de type `SFS_NOTYPE` qui permet de supprimer ces erreurs. Une condition dans `repl.c` est ainsi instaurée pour le passage dans la fonction `sfs_print: if ( output->type != SFS_NOTYPE )`

Environnements : Toutes les fonctions d'évaluations sont désormais munis de l'argument `object meta_environment` qui permet la gestion environnementale à plusieurs niveaux.

II.2 Structure de primitive

Les primitives sont implémentées dans la structure object comme indiquée dans le sujet :

```
struct {
   struct object_t * (*function)( struct object_t * );
   } primitive;
```

Toutes les primitives sont initalisées au démarrage de l'interpréteur grâce à `void init_primitive ( void )` qui lance toute la liste des primitives écrites. Elles sont toutes stockées dans l'environnement toplevel afin de faciliter leur recherche lors de l'évaluation.

`object eval_argument` évalue l'object à rentrer dans les primitives afin de donner tout son sens aux primitives.

II.3 Primitives

Primitives unaires de calcul :

```
object prim_sin ( object o );
object prim_cos ( object o );
object prim_tan ( object o );
object prim_abs ( object o );
object prim_exp ( object o );
object prim_sqrt( object o );
```

Basées sur les fonctions de `math.h`, ces primitives fonctionnent toutes de la même façon en faisant le bête calcul. On retourne simplement le résultat

Primitives unaires type? :

```
object prim_is_boolean ( object o );
object prim_is_char ( object o );
object prim_is_null ( object o );
```

```
object prim_is_number ( object o );
object prim_is_symbol ( object o );
object prim_is_pair ( object o );
object prim_is_string ( object o );
```

Grâce à `is_functions.c` créée auparavant, il y a juste une vérification du type inclus dans la forme de la fonction. Chacune renvoie le booléen `#f` ou `#t` selon l'exactitude du résultat. La doc R5RS n'indique pas la pluralité des arguments et l'interpréteur en ligne Scheme n'acceptait pas plus d'un argument d'où notre choix.

Primitives pluri-argumentaires de calcul :

```
object prim_plus ( object o );
object prim_minus ( object o );
object prim_times ( object o );
object prim_divide ( object o );
object prim_quotient ( object o );
object prim_modulo ( object o );
```

Les primitives +, -, x, / fonctionnent de la même manière à une exception près. Elles n'ont pas de limite d'entrée, faut-il que les entrées soient évaluables en nombre. La primitive va appliquer l'opérateur entre le car et le cadr jusqu'à ce qu'il n'y ait plus de cadr. Si ces primitives sont appelées avec un seul caractère, il est calculé avec pour cadr l'élément neutre de l'opérateur. La primitive / arrête son exécution et signale une erreur lorsque le cadr est égal à zéro.

Primitives type to type :

```
object prim_int_char ( object o );
object prim_char_int ( object o );
object prim_num_string ( object o );
object prim_string_num ( object o );
object prim_symbol_string ( object o );
object prim_string_symbol ( object o );
object prim_string_list ( object o );
```

Après gestion des erreurs sur le format des l'entrée ainsi que sa syntaxe, on convertit simplement l'object dans le type de sortie si cela est possible. On retourne l'object dans le nouveau type.

Autres primitives :

```
object prim_car ( object o );
object prim_cdr ( object o );
object prim_cons ( object o );
object prim_set_car ( object o );
object prim_set_cdr ( object o );
object prim_list ( object o );
```

Ces primitives agissent directement sur les objects et permettent d'en créer ou de les modifier. On ne gère pas l'ajout de "." entre les arguments d'une construction `cons`. `set-car!` et `set-cdr!` ne sont pas écrits.

III. IMPLEMENTATION

Etat du logiciel : ce qui fonctionne, ce qui ne fonctionne pas

La structure de primitive fonctionne ainsi que toutes les primitives mises en place.

Il manque encore certaines primitives utiles.

IV. SUIVI

IV.1 Répartition du travail

Implentation de la structure ensemble.

Blocs de primitives :

Romain : primitives pluri-argumentaires de calcul

Jérémy : primitives unaires de calcul, unaires type?, type to type et autres primitives

Améliorations et autres :

Romain : amélioration fonctions sur car et cdr de object.c, jeux de tests

Jérémy : meilleure gestion des erreurs, modification set! et define, évaluation d'arguments pour primitives, correction repl.c, forme quote, modification read_atom_number pour intégration float

IV.2 Problèmes rencontrés

Malgré la longueur moindre de cet incrément, il reste du travail.