Bruno PINTO
Maxence BRUNET

Enseignant : Pascal MANOURY

UE APS

COMPTE-RENDU

Mai 2020

SCIENCE ET TECHNOLOGIE DU LOGICIEL
SORBONNE UNIVERSITÉ
ANNÉE 2019/2020

Introduction

L'objectif de ce projet est d'implémenter un langage de programmation ayant à la fois un noyau impératif et fonctionel. L'implantation du lexer et parser ont été réalisé avec les technologies flex et bison basés sur le langage C. Nous avons opté pour ces technologies car elles nous étaient déjà famillières. L'implantation du typeur a elle, été réalisé en prolog conformément aux spécifications. APS0 et APS1 ont été réalisé complètement.

1 Arborescence des fichiers

Le projet est constiué d'un dossier source src qui contient toutes les sources à compiler pour tester notre langage. Un dossier test est également présent: ce dossier contient tous les tests liés à APS0 et APS1. On teste les programmes valides (c'est-à-dire qu'ils sont à la fois corrects en matière de syntaxe et de typage.) et les programmes censé être non valides (=ko). Les instructions d'exécution des tests sont présents dans le *Readme* de ce dossier.Enfin, nous avons les exécutables qui sont définis à la racine du dossier du projet:
-Eval: Evalue une source prolog.
-PrologTerm: Traduit un fichier *.aps* vers un terme en prolog.
-typrog: Teste le typage d'un terme prolog (à partir de son fichier .APS).

Analyse Syntaxique

```
typedef struct _ast{
        AstTag tag;
        int size;
        struct _ast** child;
        void* bonus;
}*AST;
```

Nous utilisons cette structure générique pour représenter l'AST. *AstTag* correspond à l'identifiant, est-ce que ce noeud correspond à un if, un bool ou une déclaration. *size* correspond au nombre de noeud fils et *child* un tableau de pointeur vers les fils. Le champ *bonus* correspond lui aux informations supplementaires qui ne correspond pas aux fils. Par exemple la valeur d'un int, ou le nom d'un ident, par default, il vaut *NULL*.
Nous avons opté pour cette représentation générique car nous la trouvions plus simple à mettre en place que le squelette proposé.

Evaluation

```
typedef struct{
        AST body; // 8 octets
        struct _env *env; // 8 octets
        String* args; // 8 octets
}Closure;
```

Nous avons basiquement implanté les Closures pour son code *body*, son environnement d'exécution *env* (qui sera detaillé juste après) , ainsi qu'un tableau contenant l'identificateur des arguments *String étant un synonyme char**. Notez que l'on utilisera aussi cette structure pour représenter les fonctions impératives (la distinction étant assuré par le typeur).
Lorsque l'on crée une closure on fait une copie de l'environnement.

```
typedef struct _f{
        struct _f *next;
        Value value;
        char *name;
}Cell, *List;
```

Nous utilisons une liste chainée pour faire office de table d'association avec comme clé l'identificateur *name* et sa valeur (fonctionnelle ou entière) par *Value* qui est un type union entre un long int et une Closure.

```
typedef struct _env{
        List global_env;
        List local_env;
        Closure_list list_to_pg;
        Value rax; // pas encore utilisé
}*Env;
```

L'environnement contient lui deux map correspondant à l'environnement global et local. Le champ rax était là en prévision des *return* (qui au final n'auront pas été implantés), la Closure_list qui est une simple liste chainée de closure est juste gardé pour clear la mémoire en fin de programme/

Typage

Le typeur va permettre de vérifier que les informations entrées par l'utilisateur sont cohérentes avec la spécification du langage. Pour cela, on met en place un prédicat pour chacun des éléments présents dans notre AST :

-typeProg : Il vérifie le programme entré par l'utilisateur.
-typeCMD : Il vérifie si le programme est un State ou une Dec.
-typeDec: Il vérifie les constantes, les fonctions classiques et recursives, les variables et les procédures classiques et récurisives.
-typeState: Il vérifie les états tel qu'écho, set, les blocs if (qui renvoient un bloc d'instructions), while et call.
-typeExpr: Il vérifie les expressions tel que les closures, les applications, les primitives, les opérations booléennes et arithmétique, les identifiants e les if.
-typeArgs: Il récupère le type des arguments.
-verifArgs: Il vérifie le type des arguments qui lui sont données, et vérifie qu'ils sont correct.