TP1 IFT2035-A A2017

Yammine, Jad 1067212
Coulombe, Annie-pier 20000419

October 28, 2017

Abstract

Ceci est TP1 fait pour le cours de "Concepts de language de programmation". Il consiste à implementer un langage fonctionnel dont la syntaxe est inspirée du langage Lisp. . De l'enoncer du devoir, on tire que

> La syntaxe de style Psil comme toujours avec celle de Lisp, les parenthèses sont signicatives. Tout comme Lisp, et au contraire de Haskell, c'est un langage typé dynamiquement. Aussi Psil a 3 types de données : les entiers,les fonctions, et les structures qui peuvent comporter un nombre arbitraire de champs et viennent avec un tag qui permet de les distinguer. Les booléens ne sont pas un type spécial : les valeurs booléennes prédénies true et false ne sont en fait que des structures avec 0 champs et avec comme tag true et false respectivement. La fonction prédénie cons construit une structure, et on peut tester le tag d'une structure ainsi qu'en extraire ses champs, avec l'opération case, qui fonctionne un peu comme celle de Haskell. Les formes slet et dlet sont utilisées pour donner des noms à des dénitions locales. Il y en a deux, car Psil, tout comme Lisp, ore autant la portée dynamique que la portée statique : les variables dénies avec slet obéissent aux règles de la portée statique/lexicale, alors que celles dénies avec dlet utilisent la portée dynamique.

**Cette partie est prise de l'énonce du "Travail Pratique 1"

1 Introduction

C'est un travail d'equipe fait par Jad Yammine et Annie-pier Coulombe. On a commencé par essayer de comprendre comment utilier le read et le show pour comprendre comment le parser converti le code Psil en Sexp. Cette tache a pris une grande partie du travaille mais ca a rendu les taches d'implementation beacoup plus facile. Une fois toute la comprehension du code est terminé on a commencer par essayer d'implementer petit a petit le s2l et le eval. A Chaque fois on verifie que le s2l et le eval fonctionne pour chaque syntaxe implementer avant de passer a la suivante.

Pour ce qui est de l'implementation, on a rencontre des difficultés avec le Lcons quand on essayer d implementer le Lcase et le Lapp, et surtout avec l implementation de dlet et slet qui nous a causer trop de confusion.

Mais apres plusieurs heures on a reussi a acomplir un travail qu'on espere etre parfait. On a surtout consacre notre temps sur notre code puis on espere pas etre jugee severement sur le rapport.

Dans ce qui suit on vas expliquer pour commencer l implementation de s2l en premiers lieux et celle de eval en second lieux.

2 Implementation

2.1 Pour s2l :

Ordre d'implementation: On a suivie l'ordre suivant pour l'implementation de s2l:

1. Lcons
2. Llambda
3. if

4. case
5. Lapp
6. Llet (dlet puis slet)

Pour Lcons: On a commencer a cree le cas de base qui est de la forme

(Scons (Scons Snil (Ssym "cons")) (Ssym tag))

et qui doit juste associe un tag a une liste vide

Ensuite on a fait le cas generale qui vas ajouter a la liste vide tout les elements de l'argument a droite apres les avoir transformer en Lexp.

Pour Llambda: Le lambda etait assez facile a implementer. On a juste du cree une fonction pour parser les arguement de lambda apple "parseArgs" qui retourne une liste de Var comme [Var] Pour le reste il suffit juste d'appele Llambda avec la liste qu'on a obtenu et le body une fois converti en Lexp

Pour if: if est une Lcase avec juste 2 case "true" ou "false". Son impmentation est assez simple pas besoin de plus de commentaire.

Pour Lcase: On a commencer par implementer le case base dont la representation en Sexp est:

(Scons (Scons Snil (Ssym "case")) x)

il l'initialise avec une liste vide sur laquel on vas ajouter les elements de du cas generale. ici on a trouver des problemes ou c'etait interpreter comme des Lcons c'est pour ca qu on'a decider de modifier Lcons a voir plusieur cas qui vont etre interpreter selon leur structure une fois transformer en Lexp.

Pour cette le Lcase on a du cree une fonction appele "parseCases" qui prends la Sexp qui constitue le body du Lcase puis le transforme en une liste de tuple de pattern et de Lexp

De la forme:

parseCases::Sexp-> [(Pat,Lexp)]

Pour Lapp: On a commencer par implementer le case base dont la representation en Sexp est:

(Scons Snil x)

il l'initialise avec le nom et une liste vide sur laquel on vas ajoutera tout les appel que la fonction nom fait

Pour Llet: Les cas de dlet et slet dans le s2l on besoin d'etre placer avant d'autre definition pour ne pas essayer d interpreter leur continu qui sera traiter comme des Lcons.

Pour les 2 type de let, on as eu besoin de cree une fonction appele "simplifie" qui a comme role de suprimer le sucre syntaxique. Et pour la fonction simplifie, on a du cree 2 fonction pour trouver les arguments et leur nom associer avant de pouvoir appeler Llet avec les 2 type de binding type.

2.2 Pour le eval:

on a du cree 2 fontion: lookupenv qui cherche dans n'importe quel environement de type Env la var fourni et retourne sa valeur associer joinVarVal qui prend 2 liste de var et val et qui associe a chaque var sa val respective pour respecter le type d'env type Env = [(Var, Value)]