IFT 2035
Travail pratique 1 - Interpréteur Slip
Rapport

Marie-Anne Prud'Homme-Maurice (1054064)
Olivier Guénette (20154866)

21 octobre 2021

Une sorte de Lisp

Dans le cadre du cours IFT 2035. Il nous a été demandé de concevoir une sorte d'interpréteur Lisp en utilisant le language fonctionnel Haskell. Le travail a pour but d'implanter une fonction qui finalise l'analyse syntaxique de l'expression fournie ainsi que de la fonction eval qui permet d'évaluer celle-ci.

Ce rapport décrit notre processus d'analyse, les problèmes rencontrés, les décisions prises et notre expérience durant la création de cet interpréteur.

Analyse et compréhension de l'énnoncé

Comme dans tout travail la première étape consistait à comprendre la tâche à réaliser. Sans le cacher, Haskell et Lisp sont des nouveaux langages pour nous. Juste ce fait rend la tâche du projet plus complexe.

À la suite de plusieurs lectures, nous avons commencé à reconnaitre des similitudes entre la structure de Slip et Haskell.

Problèmes rencontrés

Problèmes de compréhension

Dans le fichier fourni "exemples.slip", certains exemples contenaient la syntaxe de Lisp donc nous pensions devoir aussi la gérer en plus de celle du Lisp. Après consultation avec un auxiliaire d'enseignement, nous avons appris que ce n'était pas le cas et nous devions transformer ces expressions en syntaxe Lisp. Par exemple, l'expression (>3 4) devient (4 (3 >)).

Élimination dynamique du sucre syntaxique

Le premier problème que nous avons rencontré était le manque de dynamisme dans l'analyse des Scons. En effet, nous avions mis directement le nombre de Scons attendu dans le pattern matching du s2l, ce qui créait des problèmes pour les exemples qui demandaient un nombre différent. Nous avons réglé ce problème en plaçant ces Scons dans une variable qui est ensuite décomposée par s2l.

Évaluation des Lfn

L'évaluation des Lfn nous a causé problème dû à notre manque de connaissances du curring de haskell. Initialement, nous n'étions pas capables de créer une variable du type (Env ->Value ->Value). Ce qui nous à pousser à créer des fonctions auxiliaires inutiles dans le programme. L'évaluation des Lfn est quelque chose que nous avons résolu vers la fin du travail pratique.

Solutions rejetées et choisies

Implentation initial de eval slet et dlet

Lorsque nous étions rendus à l'évaluation des slet et des dlet dans la fonction eval, la première solution trouvée était de décortiquer l'information du let en plusieurs parties. Nous avions donc implanté des fonctions auxiliaires perme-

ttant de trouver toutes les variables définies dans le let ainsi que l'expression finale à évaluer.

Cette méthode fonctionnait particulièrement bien pour les expressions n'incluant pas de récursion, car cette solution remplaçait les variables du let par leur valeur. Ainsi, si une expression fessait référence à une variable du let, le système ne s'en souvenait pas.

Clairement cette méthode ne permettait pas l'évaluation de toutes les expressions de slet et de dlet. En corrigeant ce problème, nous nous somme rendu compte que la source du problème venait en effet du manque d'évaluation en Vfn des Lfn. À la suite d'une bonne évaluation des Lfn en Vfn, cela nous a permis d'ajouter les variables du let dans leurs environnements respectifs pour ensuite évaluer l'expression finale.

Implentation initial de eval pour Lfn

Comme mentionné plus haut, nous avions on problème avec l'évaluation des Lfn. La solution initiale était de décomposer le Lfn pour en extraire les variables inconnues ainsi que le corps de la fonction.

Cette méthode était pratique, car elle nous permettait de savoir exactement le nombre de variables inconnues. Ainsi, nous pouvions valider si ces variables existaient avant même d'évaluer la fonction. En contrepartie, cette méthode ne sauvegardait en aucun cas les fonctions dans l'environnement, car nous ne les transformions pas en Value. Lors de l'implantation de la fonction fact, il nous a fallu trouver une solution.

Nous avons trouvé la solution lorsque nous avons compris qu'une fonction lambda pouvait accepter plus qu'un argument à la fois. Nous nous étions basés sur les fonctions pour créer l'environnement initial des équations standard. Ainsi, nous avons pu convertire les Lexp de Lfn en Value Vfn et les insérer dans les environnements.

Conclusion

In fine, ce travail pratique nous a permis d'approfondir nos connaissances des langages fonctionnels. Certes, nous avons rencontré des problèmes. Cependant, ceux-ci étaient causés principalement par notre expérience très brève avec ces langages de programmation.