TP1 Mini-Haskell

Olga Fadeitcheva et Dereck McDuff

16 juin 2017

Introduction

Blabla ce projet fut une douleur dans les fesses (anglicisme de haut niveau)

Fonction eval

La fonction eval prend en paramètre un environnement (c'est-à-dire une liste de paires de symboles et de valeurs) et une expression (EInt, EVar, ELam, EApp), afin de sortir en output la valeur du résultat de l'évaluation de l'expression, qui doit être de type Value. La fonction eval devait être complétée pour les cas où l'expression à évaluer était soit une fonction Lambda (ELam) où une application de deux expression (EApp). Dans le cas où l'expression est une:

EApp : Il faut évaluer les deux expressions de l'application (e1 et e2). Il est important à noter que e1 ne peux pas retourner une valeur de type Int car cela donnera une application d'un Int (e2) sur un Int (e1) ce qui ne ferait pas de sens. Ainsi, e1 peut soit être une autre application (EApp), une fonction lambda (ELam) ou une variable (EVar).

- Si e1 est une variable, la fonction lookupVar va s'occuper à déterminer quelle est la valeur associée au symbole (ou quelle est la fonction primitive associée au symbole). Ainsi le type de l'évaluation de e1 sera une fonction primitive (VPrim) qui prendrai en paramètre la valeur de l'évaluation de e2, et retournera la valeur de l'évaluation de e1.

- Si e1 est une fonction lambda, la valeur retourné sera VLam qui prends en paramètre le symbole du paramètre formel passé à la fonction (sym), l'expression du corps de la fonction (ex) et l'environnement qui détermine quel symbole est associé à quelle valeur. Il faut ensuite associer le symbole passé en paramètre à la fonction lambda, à la valeur de e2 qui nous est donné. C'est-à-dire que la valeur de e2 est le paramètre actuel qu'on fournis à la fonction lambda (e1). Il faut lier ce paramètre actuel au paramètre formel (sous la forme d'un symbole) de la fonction lambda qu'on évalue, et l'enregistrer dans l'environnement qui nous est donné. Ensuite il faut évaluer l'expression du corps de la fonction lambda, avec

le nouvel environnement dans lequel on a rajouté un [Symbol, Value]. Ce processus est exécuté récursivement jusqu'à ce qu'une valeur autre qu'une fonction ou une application est retournée.

Fonction typeCheck

La fonction typeCheck prend en paramètre un environnement ainsi qu'une expression tout comme la fonction Eval. De façon similaire, nous avions à compléter le cas où typeCheck reçoit un lambda et le cas où deux expressions sont passées.

- Dans le cas du lambda, la fonction recevras en fait ceci (ELam sym t body) qui sont le symbole, le type de réception ainsi que le corps du lambda ELam. Nous associons donc à ce symbole le type passé et l'ajoutons à l'environnement. Le typeCheck redistribue le corps ainsi que l'environnement contenant ce symbol à lui même pour que l'expression constituant le corps soit analysé à son tour par le typeCheck ayant la signature appropriée. Le résultat de cette analyse sera donc soit une erreur ou un type et donc ce que la fonction typeCheck doit retourner selon sa signature. Le type de réception et retour sera passé à TArrow qui formattera ces informations pour formatter les types avec la flèche et ainsi le renvoyer à la console sous forme textuelle.

- Pour ce qui est de passer un objet EApp à typeCheck, les deux expressions constituant le EApp seront individuellement évalué pour leur type par typeCheck. TArrow se chargera d'écrire le type d'entrée et de retour.

Fonction sexp2Exp

La fonction sexp2Exp prend une Sexp et retourne soit une erreur ou une expression qui sera ensuite prête à être évaluée.

- Le sucre syntaxique de la fonction sexp2Exp avec un lambda en entré vas comme suit: sexp2Exp prends une liste contenant un symbol lambda qui lui-même contient une liste variable associées à leur type sous forme de liste et finalement un corps de fonction (Body) après la liste de variables. Pour cette implémentation, deux variables sont déclarées afin de permettre le sucre syntaxique. Cette approche fait en sorte que la fonction sexp2Exp pourra être rappelée de nouveau jusqu'à ce qu'il n'y ai plus de variables dans la liste à rajouter and enregistrer. À cette fin, la variable body (body') sera initialisé avec un appel récursif pour contenir l'entièreté des variables puisque ELam contenant les variables suivantes sera retourné. Le type de chaque variable sera mis dans t' après son évaluation par récursion. Un cas pour l'utilisation d'un lambdas sans paramètre a été ajouté pour qu'il retourne l'erreur appropriée.

- Pour ce qui est de recevoir une application avec plusieurs arguments par sucre syntaxique, deux arguments seront reçu dans celle que nous avons écrite. La logique d'évaluation suit celle ci-haut du cas lambda; deux arguments seront pris et donc passés par récursion jusqu'à ce que le dernier cas, qui ne prends qu'un seul argument, évalue et retourne le dernier argument. La variable func' prendra le résultat d'une récursion contenant une liste constituée de la fonction et des arguments restants. Lorsqu'un seul argument restera, cette récursion ira au cas final qui était procuré initialement (sexp2Exp à un arg).

- sexp2Exp reçevant ELet est très similaire aux autre dans la mesure où la même récursion générale est employée. Cependant, ELet doit passer deux arguments à chaque appel récursif, soit le type de chaque le type de chaque variable et l'expression y étant reliée. La variable body' contient le résultat de la récursion. La récursion se finit lorsque qu'une seule liste composée d'un symbol de variable, d'un type et d'une expression est passée; à ce moment, le sexp2Exp attendant une seule liste avec ces types après le ELet s'approprie l'appel et retourne le dernier élément.

Datatype et Case

Par faute de ne pas avoir un Data ou Case fonctionnel, nous avons contourné le problème en modifiant la sortie des unit tests en console. Nous avons aussi créer une Exp de type EDate contenant un array de symboles qui représenterait les types de data crées par l'expression.