IFT2035 Rapport du TP1

Yuchen Hui, Yuyang Xiong

23 octobre 2021

1 Remerciement

Sans aide de la part du professeur et des démonstrateurs, nous n'arriverions jamais à compléter le travail. Veuillez accepter notre Remerciment par coeur !

2 Compréhension générale

Suite à la distribution de l'énoncé et du code de Cadeux, nous avons commencé la lecture et la compréhension plus tôt que possible. Notre expérience :

1. Nous avons déterminé nos travaux après la lecture de données : Cela cosiste à implémenter un interpréteur d'un langage Slip de la famille Lisp, dont la syntax est très similaire à la dernière. La première partie d'un interpréteur, soit l'analyse lexicale et syntaxique, a été implantée par le professeur. La sortie de cette analyse est une expression sous forme de la structure de donnée Sexp telle que définie dans le code slip.hs. La fonction 'readSexp' a synthétisé les deux analyses : Application de la fonction readSexp à une expression du langage Slip retournera une expression Sexp qui décrit l'arbre syntaxique de l'expression en entrée. Au fur et à mesure, nous avons réalisé que la connaissance exacte de la syntax de Slip est la base de nos travaux.
2. L'étape suivante : compréhension de la syntax. En lisant la grammaire BNF-like, dans la partie 'Survol', nous avons trouvé une erreur : La teminale 'b' est senseé être

   $$b ::= ((tag\ x_1\ x_2\ ...\ x_n)\ e)$$

   plutôt que

   $$b ::= ((tag\ x_1\ x_2)\ e).$$

   Une autre remarque importante a été faite plusieurs jours après, quand nous participions à la vidéoconférence de Maxim : Le filtrage s'applique uniquement aux structures générées par (cons ..).

3. La surprise est survenue lorsque nous analysons le sucre syntaxique des appels de fonctions. Les appels de fonctions, dans Slip, est organisés différement que dans Haskell : L'argument sera passé par être placé devant la fonction, soit

$$(argument\ fonction).$$

Et le sucre syntaxique lui-même est assez étonnant :

$$(e_0\ e_1\ e_2\ ...\ e_n) \Longleftrightarrow (..((e_0\ e_1)\ e_2)\ ...\ e_n)$$

L'argument e_0 est propagé de gauche à droit, de la même manière qu'un produit qui traverse la pipeline industrielle. C'est de l'ordre inverse que dans haskell !

4. La compréhension de la fonction s2l. Le nom s2l implique Sexp to Lexp. Lexp est une représentation intermédiaire de l'expression à interpréter. Ce que la fonction s2l fait, c'est de transformer l'arbre syntaxique en Lexp, qui ne contient pas les sucres syntaxiques. Par conséquent, nous devrions aussi éliminer les sucres syntaxiques par s2l.
5. La compréhension de la fonction eval. Eval, est une fonction prennant deux environnement (Statique, dynamique) et un Lexp en argument, qui retourne une 'Value'. Cette phase sert à évaluer la représentation intermédiaire Lexp et la réduire en 'Value'. Value est une structures de données qui permet trois formes de résultat de l'évaluation : un Int, une List et une Fonction.

3 Implémentation de la fonction s2l

L'analyse des expression Sexp est critique pour l'implémentation de s2l. Nous avons adopté la méthodologie d'analyser la structure en Sexp particulière de chaque type de Slip syntax puis les traiter séparément. Chaque type de Syntax corresponde à une branche de notre fonction s2l définie en utilisant le filtrage. Donc le premier pas sera de parfaire nos connaissances de Sexp. Voici des Obstacles que nous avons rencontré et finalement surmonté :

1. Appel de fonctions. Cette partie de s2l nous semble la plus simple. Cependant, nous n'avons aucune idée en termes des façons systématiques d'extraire informations stockées sous forme de Sexp. Au tout début, nous pensions que, puisque Sexp représente l'arbre syntaxique d'une expression, alors on pourrait le parcourir tout comme parcourir une structure de donnée arbre. Malheureusement Nous sommes bloqués dedans pour plusieurs heures (Peut-être à cause du fait que jusqu'à date le Cour IFT2015 n'a pas encore couvert les matières sur le parcours des arbres). Alors, nous avons changé d'aspect de voir le problème. Comme l'indiqué dans l'énoncé (ainsi que la vidéo du professeur), Sexp n'est rien qu'une liste simplement chainée. À partir de cette opinion, nous avons compris les

Sexp. Auparavant ce qu'on a considérés comme des sousarbres deviennent les sous-liste. Ensuite, inspirés par la fonction foldl pour les liste de haskell, nous avons implémenté la fonction 's2l_foldl', qui prend en argument une autre fonction auxiliaire 'pipeline' . Puisque la fonction 's2l_foldl' est d'ordre supérieur, il sera trop abstrait de décrire en mot. Je vous invite donc à lire le code si vous voulez en savoir plus.

2. Constructeur. Avec la fonction 's2l_foldl', on a qu'à implémenter une fonction auxiliaire 'consConcate' similaire à 'pipeline' et appeler 's2l_foldl' passant en argument 'consConcate'.
3. Case. Case utilise aussi le concept de foldl, cependant elle est plus complexe. Nous avons modifié 's2l_foldl' pour obtenir la nouvelle fonction 's2l_foldl_case'. De la même manière, nous avons implémenté des fonctiosn auxiliaires pour case.
4. Dlet,slet,lambda. Par analogie, nous avons trouvé que ces trois branches utilisent le concept de foldr. Lambda est la plus simple, un peu comme le rôle que joue l'appel des fonctions pour le concept de foldl, donc nous avons implémenté 's2l_foldr' pour expression lambda et puis la version adaptée aux slet et dlet : 's2l_foldr_let'. Aussi, nous avous créé queleques fonctions auxiliaires.
5. Nous avons remarqué que la syntax d'un appel de fonction et celle d'une déclaration de varible partage la même forme, toutefois cela ne causera pas de problème, car les déclarations exsitent seulement dans la branche pour $(case\ e\ b_1\ ...\ b_n)$ et toute autre ocurrence seront conséquemment considérées comme un appel de fonction.

4 Implémentation de la fonction eval

Avec les expériences des devoirs 1-3, nous sommes tous familiers avec ce que signifie eval. La partie la plus difficile sera la compréhension des comprtements de slet et dlet, ainsi que l'évaluation des 'Lfn'.

4.1 Problème avec le concept de la portée

Normalement, le concept 'portée' sera utilisé pour décrire une propriété d'un langage de programmation. La définition des deux portée dans le livre 'Programming Language Design Concepts' :

"A language is statically scoped if the body of a procedure is executed in the environnement of the procedure's definition."

"A language is dynamically scoped if the body of a procedure is executed in the environnement of the procedure call."

Cependant, dans l'énoncé du TP, on dit 'une variale utilise la portée statiques(dynamique).' Cette expression nous semble plutôt mélangante : on ne parle plus d'évaluation des fonctions mais d'évaluation des variables ! Heureusement, quand nous posons nos questions, le démonstrateur Maxim nous a tracé

la lighe et nous a guidé vers une compréhension correcte :

les varibale dans une fonction, si elles sont déclarées statiquement, nous les cherchons alors dans l'environnement où la fonction a été crée ; si elles sont déclarées dynamiquement, nous les cherchons dans l'environnement où la fonction est appelée !

Après deux jours, nous avons comfirmé notre stipulation, en voyant une réponse rédigée par Monsieur Monnier dans le forum de discussion qui nous guide aussi vers cette compréhension !

4.2 Choix à faire

Lorsque l'on était en train de coder la branche de la fonction eval qui évalue le "Lvar Var", on était bloqué par un problème – puisqu'on avait deux environnements(l'un est statique et l'autre dynamique), dans quel environnement devait-on chercher le "Var" ? On a posé cette question à l'auxiliaire Maxim Bernard pendant la vidéoconférence ayant lieu ce lundi, il croyait qu'on pourrait chercher d'abord dans l'environnement statique puis dans l'environnement dynamique. On s'est rappeleé que le prof avait dit, dans la vidéo promenade, qu'il existait plusieurs possibles et que c'était à nous de prendre la décision pour choisir l'environnement, dans lequel on cherche une variable. Finalement, On a décidé de respecter la suggestion de Maxim – chercher d'abord dans l'environnement statique puis l'environnement dynamique. Nous allons expliquer la raison par un exemple ici :
eval env0 [] (s2l (readSexp"(dlet (((curried+ x1) (slet ((x 9)) (lambda (y) (y (x +)))))(x 7))(6 (5 curried+)))")) ;⇝ 15.
Dans cette expression, on remarque que le corps de la fonction curried+ sera (slet ((x 9)) (lambda (y) (y (x +)))), où x est déclarée statiquement. En conséquence, quand on veut évaluer le corps, la valeur de x devrait être cherchée dans l'environnement où curried+ est définie, dans laquelle x vaut 9. Donc le résultat d'évaluation est supposé d'être 15 selon notre compréhension de slet et dlet discutée dans la section 4.1. Si on décide de chercher d'abord dans l'environnement statique, l'interpréteur trouve bien que x = 9, qui est compatible avec notre compréhension ; Par contre, si on cherche premièrement dans l'environnement dynamique, l'interpréteur assignera à variable x la valeur 7 trouvée dans l'environnement d'appel. Ce résultat n'est pas raisonnable car il est en conflit avec le fait que x est déclarée statiquement.
En conclusion : il vaut mieux de chercher d'abord dans l'environnement statique.

4.3 Évaluation de Lfn

Sur ce qui concerne l'évaluation de Lfn (Par exemple (Lfn "x" (Evar "x"))), Nous ne savions aussi pas trop comment transférer 'x' en Value, vu qu'il n'y a pas de place pour des string dans la structure de donnée Value. Encore une

fois, Maxim nous a donné un indice. Avec l'indice, on implémente rapidement le reste du TP1.

4.4 Autre

Les autres parties sont similaires aux devoirs 1-3, donc nous n'avons pas rencontré d'empêchements.

5 Conclusion

Grace à ce TP, nous comprenons mieux les concepts des langages de programmaiton vus en cours en écrivant une partie d'un interpéteur de Slip. Cela nous a pris 5 jours (et un jour pour le rapport et les tests). Ce sont 5 jours occupés mais aussi 5 jours remplis de surprise. Merci à tout le monde !