Rapport du Travail Pratique 1

IFT 2035

Nom des membres de l'équipe :

Naromba Condé - 20251772
Qiyun Ou - 20264284

Description du Devoir

Ce projet consiste en l'implémentation d'un langage de programmation de type Lisp. Le devoir consistait à implémenter les fonctions `s2l` et `eval`. Le langage doit prendre en charge les fonctionnalités suivantes :

- Analyseur lexical pour traiter les entrées du langage.
- Analyseur syntaxique pour valider la structure des programmes.
- Mécanisme d'évaluation pour exécuter les programmes et retourner les résultats.

Le travail inclut également la gestion des objets fonctionnels et des fixations récursives, permettant l'évaluation de fonctions mutuellement récursives.

1 Difficultés Rencontrées

Au cours de la réalisation de ce projet, nous avons rencontré plusieurs défis. On a passé du temps à lire et à comprendre le squelette du code fourni dans l'énoncé. Il était nécessaire de maîtriser le mécanisme des fonctions des types `Sexp` et `Lexp` pour réussir l'implémentation.

1.1 Difficultés avec les Objets Fonctionnels (`fob`)

Lors de l'implémentation des objets fonctionnels, on a rencontré des erreurs à cause de la mauvaise traduction des expressions. Au début, nos expressions étaient mal traduites, et du coup, à l'exécution, on obtenait des `Lsend` au lieu de `Lfob`. Ça causait des soucis dans le traitement des paramètres et l'évaluation des fonctions.

Pour déboguer tout ça, on a utilisé la notion de trace. On a dû ajouter `import Debug.Trace (trace, traceShow)` pour voir chaque étape de notre exécution. Grâce à ces traces, on a vu qu'il y avait un message d'erreur disant que la variable `x` n'était pas connue. On a réalisé que le problème venait de la façon dont on avait créé l'objet `fob` dans la fonction `s2l`. On n'avait pas pensé à tester `s2l` pour `fob` avant de l'intégrer à l'évaluateur.

En nous basant sur les messages d'erreur obtenus lors du débogage, on a modifié la fonction `s2l` pour corriger la création des objets fonctionnels.

1.2 Difficultés avec la Gestion de `fix`

Lors de l'implémentation de `fix`, nous avons rencontré plusieurs difficultés.

1.2.1 Compréhension des Spécifications

Nous avons eu beaucoup de mal à comprendre les attentes pour la fonction `fix`, ce qui a rendu difficile la compréhension de la manière dont les fonctions mutuellement récursives devaient être définies et évaluées. Il a fallu plusieurs lectures de l'énoncé, du code fourni, ainsi que la lecture des réponses fournies par le professeur aux questions posées sur le forum de discussion pour comprendre.

1.2.2 Analyse des Déclarations

Il fallait analyser chaque déclaration de fonction et s'assurer qu'elles étaient correctement interprétées. Cependant, nous avons rencontré des difficultés à extraire les noms des fonctions et leurs paramètres à partir des nœuds `Snode`. Les erreurs de pattern matching sont apparues, notamment en raison d'une mauvaise compréhension de la structure des `Sexp`.

1.2.3 Erreurs de Traduction

Au début, nos expressions `fix` étaient mal traduites, ce qui faisait que, lors de l'exécution, on obtenait des `Lsend` au lieu de `Lfob`. Cela entraînait une mauvaise interprétation des appels récursifs et une évaluation incorrecte des résultats.

1.2.4 Débogage de la Fonction `s2l`

En tentant d'intégrer `fix` dans la fonction `s2l`, nous avons réalisé que le traitement des déclarations n'était pas adéquat. Nos premières tentatives ont souvent conduit à des résultats inattendus, tels que des évaluations incorrectes ou des références à des fonctions non définies. Par exemple, les appels de fonctions mutuellement récursives dans le corps de `fix` n'étaient pas gérés correctement.

En analysant les expressions simples pour `fix`, nous avons pu constater que la récursivité mutuelle entre `even` et `odd` fonctionnait correctement, avec l'environnement contenant bien les deux fonctions. Cependant, notre code ne semblait pas fonctionner pour l'expression `fix` que le professeur a donnée dans le fichier `exemple.slip`.

Pour illustrer ce problème, voici un exemple d'erreur que nous avons eu pour `fix` :

```
ghci> readSexp "(fix (((even x)  (if (= x 0) true  (odd  (- x 1)))
   ) ((odd x)   (if (= x 0) false (even (- x 1))))) (odd 42))"
Snode (Ssym "fix") [Snode (Snode (Snode (Ssym "even") [Ssym "x"])
   [Snode (Ssym "if") [Snode (Ssym "=") [Ssym "x",Snum 0],Ssym "
   true",Snode (Ssym "odd") [Snode (Ssym "-") [Ssym "x",Snum
   1]]]]) [Snode (Snode (Ssym "odd") [Ssym "x"]) [Snode (Ssym "if"
   ) [Snode (Ssym "=") [Ssym "x",Snum 0],Ssym "false",Snode (Ssym
   "even") [Snode (Ssym "-") [Ssym "x",Snum 1]]]]],Snode (Ssym "
   odd") [Snum 42]]
ghci> s2l (Snode (Ssym "fix") [Snode (Snode (Snode (Ssym "even") [
   Ssym "x"]) [Snode (Ssym "if") [Snode (Ssym "=") [Ssym "x",Snum
   0],Ssym "true",Snode (Ssym "odd") [Snode (Ssym "-") [Ssym "x",
   Snum 1]]]]) [Snode (Snode (Ssym "odd") [Ssym "x"]) [Snode (Ssym
    "if") [Snode (Ssym "=") [Ssym "x",Snum 0],Ssym "false",Snode (
   Ssym "even") [Snode (Ssym "-") [Ssym "x",Snum 1]]]]],Snode (
   Ssym "odd") [Snum 42]])
Lfix [] (Lsend (Lvar "odd") [Lnum 42])
ghci>
```

Listing 1 – Erreur rencontrée lors de l'utilisation de `fix`

Nous nous sommes rendu compte qu'à la fin de l'exécution, on se retrouvait avec une liste vide pour les déclarations dans `fix` (le premier argument de `Lfix` est `[]`).

Ce problème provenait du fait que notre fonction `s2l` ne gérait pas correctement la structure des `Sexp` lors de l'analyse des déclarations dans `fix`. Plus précisément, les déclarations étaient encapsulées dans des nœuds supplémentaires, ce qui empêchait leur extraction correcte.

En constatant ça, nous avons ajusté la fonction `s2l` pour gérer correctement les `Sexp` encapsulés et extraire les déclarations même lorsqu'elles sont enveloppées dans des nœuds supplémentaires.

Après cette modification, la fonction `s2l` a pu analyser correctement les déclarations dans `fix`, ce qui a permis de remplir la liste des déclarations dans `Lfix` et d'évaluer correctement les fonctions mutuellement récursives.

1.2.5 Tests et Validation

La validation des résultats a également nécessité plusieurs tests. Nous avons d'abord utilisé des exemples simples de déclarations de fonctions pour vérifier que l'implémenta-

tion de `fix` fonctionnait comme prévu, puis nous avons finalement testé notre code sur les exemples fournis par le professeur dans le fichier exemples.slip .

2 Choix de Programmation

Nos choix de programmation ont été guidés par le besoin d'avoir un code propre. J'ai choisi une approche plus structurée, où chaque partie du langage est gérée par des fonctions séparées, ce qui facilite le débogage et la maintenance.

Voici les choix de programmation que j'ai utilisés :

— **Création de fonctions auxiliaires** : Nous avons créé des fonctions comme `sexpToList`, `parseParamsFromSexp` et `parseDecl` pour gérer des tâches spécifiques, comme convertir un `Sexp` en liste ou extraire les paramètres et déclarations. Ça a simplifié le code principal et l'a rendu plus lisible.

— **Approche structurée** : En séparant les différentes fonctionnalités du langage en fonctions distinctes, le code est devenu plus organisé. Chaque composant, comme l'analyse syntaxique ou l'évaluation des expressions, est traité indépendamment, ce qui facilite les modifications et les ajouts futurs.

— **Utilisation d'outils de débogage** : On a utilisé les fonctions `trace` et `traceShow` du module `Debug.Trace` pour suivre l'exécution du programme pas à pas. Ça nous a aidé à identifier et corriger les erreurs plus efficacement en visualisant le flux d'exécution et les valeurs intermédiaires.

— **Amélioration de la fonction `s2l`** : Nous avons ajusté la fonction `s2l` pour qu'elle gère correctement les différentes structures de `Sexp`, surtout pour les expressions `fix` et `fob`. Ça a nécessité de bien comprendre les structures de données et de gérer les cas particuliers où les expressions sont encapsulées dans des nœuds supplémentaires.

— **Gestion des environnements pour la récursivité** : Pour permettre aux fonctions de se référencer mutuellement lors de leur évaluation, on a mis en place un environnement récursif. Ça a impliqué d'adapter la fonction `eval` pour gérer correctement cet environnement, notamment lors de l'évaluation des fixations récursives avec `fix`.

— **Tests et validations réguliers** : Nous avons fait des tests avec des exemples simples et ceux fournis par le professeur pour valider le fonctionnement de l'implémentation. Chaque test a aidé à identifier des problèmes supplémentaires et à améliorer le code.

Ces choix ont aidé à créer un interpréteur plus robuste et maintenable, tout en facilitant le développement et le débogage.

2.1 Débogage et Tests

Le processus de débogage a été important et on a passé une bonne partie de temps à identifier et corriger les erreurs. Nous avons utilisé des messages d'erreur explicites pour s'assurer que chaque étape du processus d'analyse et d'évaluation était claire.