IFT 2035: Rapport TP1

Jaydan Aladro (20152077)
David Telles (75030101)

Pour le 30-05-2021 à 23:59

1 Problèmes rencontrés

Pendant l'écriture du code manquant, nous avons rencontré plusieurs problèmes qui étaient prévisibles, mais relativement durs à régler. Tout d'abord, lors de la compilation, les erreurs les plus communes étaient celles de typage. Effectivement, en raison de notre manque d'expérience en Haskell (généralement au début), c'était parfois difficile de prévoire le type d'une valeur dans les parties de code où on retrouvait une multitude de symboles. De plus, nous avons fait face à d'autres erreurs de compilation comme les erreurs de parenthésage.

En ajout à cela, nous avons également rencontré des problèmes qui ne sont pas liés à la compilation. Premièrement, c'était plutôt difficile de trouver des solutions récursives, notamment dans les fonctions qui convertissent les expressions. En effet, on pouvait parfois passer un temps énorme à trouver une solution qui fonctionne, ce qui a beaucoup influencé la gestion de notre temps pour faire le travail. Deuxièmement, lié au problème mentionné précédemment, nous avons dû apporter une attention particulière à la simplicité de nos fonctions, ce qui a également prit du temps. Évidemment, une bonne lisibilité était un élément important, afin d'éviter les erreurs de typage.

Ensuite, il y avait d'autres problèmes moins fréquents, mais aussi problématiques que ceux qu'on a mentionné. Tout d'abord, on devait s'assurer que le code fonctionnait et était conforme aux consignes de l'énoncé, ce qui était difficile à cause de plusieurs raisons. Premièrement, dans le but de débugger notre code, nous devions faire appel aux diverses fonctions prédéfinies dans le code source, ce qui demandait beaucoup de compréhension et de relecture de l'énoncé. De plus, il fallait bien comprendre ce qu'une fonction devait faire et pas faire (par exemple, la gestion de cas particuliers ou de sucre synthaxique), ce qui demandais également une grande compréhension du code source et de l'énoncé.

Quelques problèmes plus spécifique étaient les suivants :
- l'évaluation du "Llet" a pris le plus de temps, pour comprendre et ensuite surtout pour faire la fonction avec l'évaluation paresseuse de Haskell. Avec l'aide de Maxime et beacoup de patience, nous avons fini par réussir.
- la construction du "let" était aussi quelque chose très melangeant. Les multiples récursions imbriqués donnaient un mal de tete. Cependant on a trouvé cela moins laborieux que l'évaluation du "Llet". Au début, nous avions fais une fonction basique qui ne traitait que le cas de 'd' pour une déclaration de variable. Nous n'avions pas vu les 3 cas de 'd'.
Mais nous avons réalisé en faisant, justement, l'évaluation du "let" nos erreurs.
-lors d'une construction "call", nos arguments étaient inversé. On a réglé cela avec un simple "reverse". (Merci pour l'aide de Karim Boumghar qui avait un probléme similaire) -pas trop un probléme, mais nous avions réalisé un peu trop tard l'utilité de la fonction "++" en Haskell. On croyait que cela s'appliquait seulement aux strings et non au listes aussi ! (merci Karim Boumghar aussi pour cette découverte)

2 Surprises

Dans le travail pratique, nous avons fait face à beaucoup de surprises (autant positives que négatives). Pour commencer, lorsque le compilateur nous informait qu'il y avait des erreurs de typage, nous avions généralement l'instinct d'aller modifier nos lignes de codes, afin de les régler. Cependant, on a été surpis de constater que, dans certains cas, ces erreurs sont liés à l'ordre des lignes. En addition à cela, on a été plutôt surpris par notre incapacité à gérer notre temps et planifier notre travail, en raison de la multitude de problèmes qui peuvent survenir en Haskell. Contrairement aux languages de programmation (comme Java, C, etc), il est difficile d'estimer le temps qu'on peut

passer sur un morceau de code, à cause qu'une grande partie des fonctions doivent être récursives, ce qui fait qu'une fonction peut avoir l'air plus simple qu'elle est en réalité.

En plus de cela, nous avons été positivement surpris dans des cas. Tout d'abord, en avançant dans le travail, nous étions surpris de constater la différence de notre niveau en Haskell avec celui lorsqu'on venait de commencer. Par exemple, quelque chose qui paraissait difficile à comprendre au départ pouvait facilement être comprise vers la fin. De plus, à plusieurs reprises, nous avons modifié nos lignes de code écrites au début du travail pratique, afin d'éliminer des parties redondantes. Ensuite, nous avons été plutôt surpris par la petite taille de nos fonctions, malgré leur complexité.

Durant la fin, les tests nous ont particulièrement surpris comme étant pas si simple que nous l'avions imaginé. Par exemple, nous avons beaucoup douté de notre code durant les exécutions car nous recevions des erreurs. Nous avons eu du mal avec le type des "Tuple" pour un "hastype" et au final cela n'était que du mauvais syntaxe Psil. En revanche, nous avons pu remarqué une erreur de typage "check" pour "fetch". L'environement n'était pas mis à jour après l'inférence. (vous pouvez constater l'usage de "++" sur des listes que nous venions de découvrir)

3 Choix que nous avons dû faire

Au cours du travail pratique, nous voulions à tout prix éviter les erreurs, puisque les parties suivantes du travail se basent généralement sur les parties précédentes (donc les erreurs au début pouvaient être catastrophiques pour les résultats à la fin). Pour cela, nous avons dû très souvent valider nos lignes de code en posant des questions (en plus des nombreuses relectures de l'énoncé).

Pour poursuivre, notre fonction s2l devait être en mesure de traiter des déclarations "let" sous plusieurs formes (par exemple (x e), (x t e), etc). Dans le code qui gère cela, nous étions obligé de faire des conversions de listes de Sexp en Lexp. En raison de cela, nous avons décidé de faire usage de fonctions externes.

Ce coucher très tard peut-etre considéré un choix que nous avons dû faire. (Mais c'est amusant de faire du 'problem solving')

4 Options que nous avons sciemment rejetées

Nous avons rejeté plusieurs options qui s'offraient à nous, lors de notre travail pratique. En premier lieu, il était clair, selon nous, que les importations de modules étaient des choses à éviter pour plusieurs raisons. D'abord, on savait qu'en utilisant des importations, notre solution aurait été éloignée de celle du corriger (puisque les importations ne sont pas mentionnés dans l'énoncé). En plus de cela, le code serait plus lourd que nécessaire.

Ensuite nous avons aussi évité le plus possible les fonctions externes. Nous trouvions que cela pouvait rendre le code trop confus.

Finalement, avant tout, nous avons évité de modifier le code source pour plusieurs raisons encore. Premièrement, en raison du fait que nous étions pas énormément familier avec le code, il y avait des possibilités que nous brisons le code suite à une modification. En effet, le code source contient beaucoup d'éléments qui dépendent de d'autres éléments, ce qui rend la possibilité d'oublie relativement grande. Deuxièmement, nous savions qu'en faisant une telle chose, notre solution deviendrait différente de celle prévue, ce qui apportait davantage de problèmes. D'abord, demande de l'aide aurait été plus difficile, car il aurait possiblement fallu détailler les autres changements. En plus de cela, on risquerait que notre solution soit moins efficace que prévu.

5 Ce que nous avons retiré de ce travail

Les problèmes de récursions ce font plus vite et saute à l'oeil plus rapidement. Ce travail nous a fait beaucoup apprécier Haskell, ce language effrayant à premier coup d'oeil.

Les utilisations d'environement sont largement mieux compris maintenant et mieux manipulé.