Paradigmes et Langages de Programmation

Haute École d'Ingénierie et de Gestion du Canton de Vaud

Devoir 2

2022

1 Introduction

Ce devoir va constituer pour vous en l'implémentation Haskell d'un interpréteur d'un langage de programmation fonctionnelle. Cette implémentation nécessite d'être familier avec le fonctionnement des interpréteurs. Ce sujet sera introduit lors d'une séance ultérieure du cours. Il n'en demeure pas moins que vous pouvez d'ores et déjà commencer l'implémentation des autres tâches du devoir et qu'il vous est possible de les tester sans cette partie. Vous êtes toutefois libres de prendre de l'avance sur le cours et de lire les chapitres correspondants de l'ouvrage Programming Language Concepts.

2 Langage fonctionnel

La définition et l'implémentation d'un langage de programmation fonctionnelle vous sont proposées dans ce devoir, que vous êtes libres de réaliser seul ou en binôme. Il s'agit d'écrire en Haskell un interpréteur d'un langage de programmation caractérisé par des constructions de programmation typiques du paradigme fonctionnel. Afin de vous guider dans votre démarche, ce devoir est divisé en tâches bien précises. La description de chaque tâche se veut être délibérément concise et exige de votre part un certain degré d'investissement quant à l'élaboration de vos solutions et la recherche voire demande de compléments d'information. Notez que tout comportement non-spécifié dans l'énoncé signifie que vous pouvez l'adresser comme bon vous semble, pour autant que cela soit pertinent dans le contexte d'interprétation d'un langage de programmation. En cas de doute, utilisez sans autre le forum de discussions pour poser vos questions et demander des clarifications.

2.1 Fonctionnalités

Votre langage de programmation fonctionnelle doit fournir deux catégories de constructions de programmation : des *définitions* et des *expressions*. Une définition ne retourne pas de valeur lors de l'évaluation mais augmente l'environnement de l'interpréteur (global) avec une nouvelle association alors qu'une expression retourne une valeur d'exécution durant l'évaluation. *Cette distinction est similaire à celle du langage Haskell ; dans l'interpréteur ghci, une définition ne retourne pas de valeur, tandis qu'une expression retourne une valeur.*

En outre, votre langage de programmation fonctionnelle doit supporter au minimum les fonctionnalités suivantes :

Définition de fonctions (non-récursives) ; définition de variables ; littéraux entiers, booléens, tuples ; occurrences de variable ; applications de fonction ; expressions let-in avec plusieurs définitions ; expressions case-of sans gardes avec motifs universel, variable, littéraux ; opérations unaires, binaires.

Vous êtes libres d'inclure des fonctionnalités additionnelles qui peuvent éventuellement vous rapporter des points supplémentaires. Néanmoins, celles-ci seront considérées selon leur pertinence et leur difficulté à mettre en place, et à condition seulement que celles qui vous sont demandées de base soient bien supportées.

2.2 Grammaire

Décrivez une grammaire EBNF, **grammar.txt**, d'un langage de programmation fonctionnelle permettant d'exprimer les fonctionnalités énoncées à l'aide de constructions de programmation. Votre grammaire doit rendre possible l'écriture de définitions et d'expressions en vue de les évaluer au sein d'une boucle de lecture-évaluation-impression (*REPL*), et ce, à la manière de l'interpréteur Haskell. La syntaxe du langage ne vous est pas imposée. Libre à vous de créer une syntaxe originale ou de vous inspirer de syntaxes existantes.

2.3 Analyse lexicale

Écrivez un module Haskell, **lexer.hs**, qui expose une fonction permettant d'effectuer l'analyse lexicale de termes de votre langage fonctionnel. Pour ce faire, vous utiliserez la librairie Alex en vous appuyant sur l'alphabet de la grammaire décrite au préalable. Vous devrez également définir une représentation intermédiaire décrivant les unités lexicales (*tokens*) d'un terme que vous exposerez et utiliserez lors des tâches suivantes.

2.4 Analyse syntaxique

Écrivez un module Haskell, **parser.hs**, qui expose une fonction permettant d'effectuer l'analyse syntaxique de termes de votre langage fonctionnel. Pour ce faire, vous utiliserez la librairie Happy en vous appuyant sur la grammaire décrite au préalable. Vous devrez également définir une représentation intermédiaire décrivant l'arbre syntaxique abstrait d'un terme que vous exposerez et utiliserez lors des tâches suivantes.

2.5 Analyse sémantique

Écrivez un module Haskell, **semantics.hs**, qui expose une fonction permettant de calculer le type d'un terme de votre langage fonctionnel, autrement dit, le type d'une définition ainsi que d'une expression. De plus, votre fonction devra effectuer toutes les vérifications de type nécessaires à l'exactitude sémantique d'un terme. Les règles de typage que vous devrez implémenter pour la vérification des types sont laissées à votre bon jugement.

2.6 Interpréteur

Écrivez un module Haskell, **eval.hs**, qui expose une fonction permettant d'évaluer des termes de votre langage fonctionnel. Cette fonction doit partir du principe que tout terme devant être évalué est correctement typé à ce stade. Elle devra toutefois veiller à ce que les symboles évalués soient bien définis. Les seules erreurs susceptibles de se produire lors d'une évaluation sont donc des erreurs d'exécution dues à des erreurs de programmation de l'utilisateur. À vous d'identifier lesquelles et d'adopter un comportement approprié lorsque cela est amené à se produire.

2.7 REPL

Écrivez un programme Haskell, **repl.hs**, qui implémente une boucle de lecture-évaluation-impression (*REPL*). Cette boucle doit permettre d'interpréter votre langage de programmation fonctionnelle. Pour ce faire, vous devez combiner les différentes fonctions d'analyse implémentées précédemment. Une entrée utilisateur donnée doit valider toutes les phases d'analyse avant de pouvoir évaluer le terme qui en découle. Votre boucle de lecture-évaluation-impression doit maintenir un environnement global, mis à jour lors de chaque nouvelle définition à la manière de l'interpréteur Haskell :

```
Prelude> x = 5
Prelude> x
5
Prelude> f y = y + 1
Prelude> f x
6
```

Votre programme doit également implémenter un interpréteur de commandes qui s'exécute à chaque tour de boucle, pour autant que l'utilisateur souhaite poursuivre l'interprétation, et qui supporte les commandes suivantes

:{	activer l'édition multi-ligne (:} pour la désactiver)
:r	réinitialiser l'état de l'interpréteur
:t <expr>	afficher le type d'une expression
:e	afficher l'environnement
:h	afficher l'aide
:q	quitter le programme

Votre interpréteur de commande devra reporter de manière conviviale à l'utilisateur toutes les erreurs que vous seriez amenés à générer dans les différents composants de votre interpréteur.

3 Évaluation

La grammaire comptera pour une note indépendante et fera l'objet d'une évaluation ternaire : *fait*, *partiellement fait*, *pas fait*. En revanche, l'évaluation de votre implémentation pour chacune des tâches proposées dans ce devoir s'appuiera sur des critères bien précis, qui sont :

■ Exactitude	Le code est correct au sens des exigences et de l'exécution
■ Complexité	Le code est compréhensible à la première lecture
■ Performance	Le code est dépourvu de surcharges inutiles de performance
■ Modularité	Le code est organisé de façon pertinente
■ Style	Le code est écrit dans un style fonctionnel

Chaque critère aura le même poids sur le nombre de points obtenus pour une tâche donnée. De manière similaire, chaque tâche vous rapportera au maximum un point sur le total de la note du devoir. Autrement dit, la note du devoir sera calculée selon la formule :

$$N = 1 + \sum_{x=1}^{5} E_x + C_x + P_x + M_x + S_x$$

où E_x, C_x, P_x, M_x *et* S_x valent chacun un cinquième de point.

4 Rendu

Le rendu du devoir comprend au minimum les six fichiers mentionnés explicitement dans ce document. Chaque fichier doit inclure une entête, un commentaire Haskell, laquelle indique le(s) auteur(s) du code. Le code rendu doit être le résultat de votre propre production. Le plagiat de code, de quelque façon que ce soit et quelle qu'en soit la source, sera considéré comme de la tricherie et dénoncé. La date de rendu est fixée au **vendredi 17 juin 2022 à 16h30**. Aucun retard ne sera admis et la note de 2 vous sera automatiquement attribuée si cela devait se produire.

5 Conclusion

Ce devoir est exigeant dans la mesure où il vous est demandé de développer un langage de programmation à partir de zéro. Il n'en demeure pas moins excitant puisqu'il connecte ensemble différents sujets du cours et vous fait construire ce qu'on appelle un frontend dans le jargon des compilateurs. À vous d'y investir le temps que vous jugerez nécessaire. Le forum de discussions est à votre entière disposition pour poser des questions et demander des conseils sur les différentes tâches du devoir. Toutefois, ce forum ne doit pas devenir le lieu pour déboguer votre code. En cas de bug dans vos implémentations, il est de votre responsabilité d'identifier la source du problème et de la corriger. À cet égard, les outils de debugging sont vos meilleurs alliés.

Bon travail !