HE VD
IG

Paradigmes et Langages de Programmation

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

Devoir 6

2023

1 Introduction

Ce devoir va consister pour vous en la deuxième étape de l'implémentation Haskell d'un interpréteur d'un langage de programmation. À cet égard, il vous est demandé d'implémenter un analyseur sémantique lequel est constitué à son tour de sous-phases, à savoir l'analyse de noms et la vérification de types.

2 Analyse sémantique

Dans ce devoir, on vous propose de concevoir un langage de programmation caractérisé par des constructions de programmation typiques du paradigme fonctionnel. Celui-ci a pour but de vous faire pratiquer la conception des langages de programmation telle qu'étudiée jusqu'à maintenant. La donnée du problème se veut être délibérément concise et exige de votre part un certain degré d'investissement quant à l'élaboration de votre solution 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 n'affecte en rien la robustesse, la pertinence et la cohérence de votre solution. En cas de doute, utilisez sans autre le forum de discussions pour poser vos questions.

2.1 Analyse de noms

Écrivez un module Haskell, **name.hs**, implémentant et exposant une fonction qui permet d'effectuer une analyse de noms sur des termes de votre langage fonctionnel. Cette fonction doit partir du principe que tout terme devant être analysé est syntaxiquement correct à ce stade du processus d'interprétation. Dit autrement, elle recevra en argument un arbre syntaxique abstrait généré par la phase d'analyse syntaxique implémentée précédemment. La fonction ne devra pas construire de table des symboles mais se contentera de détecter des problèmes autour de la déclaration et l'utilisation de symboles. On admettra que votre langage est caractérisé par une portée lexicale et permet aux déclarations de symboles de faire de l'ombre aux symboles déclarés dans la portée externe (*shadowing*), pour autant que cela ne crée ni conflit ni ambiguïté.

2.2 Vérification de types

Écrivez un module Haskell, **type.hs**, implémentant et exposant une fonction qui permet d'effectuer une vérification de types sur des termes de votre langage fonctionnel. Cette fonction doit partir du principe que tout terme devant être analysé est non seulement syntaxiquement correct mais également que tous les symboles qu'il contient peuvent être résolus à ce stade du processus d'interprétation. Dit autrement, elle recevra en argument un arbre syntaxique abstrait pour lequel l'analyse de noms n'a détecté aucune erreur. Votre langage étant statiquement et explicitement typé, la vérification de types devra s'appuyer sur les annotations de type des symboles déclarés de part et d'autre pour s'assurer qu'un terme est correctement typé. De plus, vous implémenterez les règles de typage du langage Haskell qui s'imposent selon les constructions de programmation de votre langage. Par exemple, la condition d'une expression conditionnelle doit être de type booléen. À vous d'identifier les autres règles de typage à appliquer.

2.3 Gestion des erreurs

La gestion des erreurs durant les différentes vérifications de l'analyse sémantique doit adopter un style purement fonctionnel. Dit autrement, vous ne pouvez pas lever simplement une erreur d'exécution au moyen de la primitive *error*. À la place, les fonctions d'analyse de noms et de vérification de types retourneront l'ensemble des erreurs identifiées au cours de leurs processus respectifs. Ces erreurs doivent décrites de manière compréhensible en vue de les reporter à l'utilisateur lors d'une phase ultérieure. À vous de déterminer quelle(s) structure(s) de données adéquate(s) utiliser pour représenter de telles erreurs.

3 Évaluation

Pour ce devoir, l'évaluation de votre implémentation s'appuiera sur les critères suivants :

- Exactitude — Le code est correct au sens des exigences énoncées
- Fiabilité — Le code est robuste en toute circonstance
- Complexité — Le code est facile à lire et à comprendre
- Performance — Le code est dépourvu de surcharges de performance
- Style — Le code est écrit dans un style fonctionnel

Chaque critère aura le même poids sur le nombre de points obtenus pour ce devoir. Un critère vous rapportera au maximum un point sur le total de la note, qui sera calculée selon la formule :

$$N = 1 + E_x + F_x + C_x + P_x + S_x$$

où E_x, F_x, C_x, P_x *et* S_x correspondent à l'évaluation de votre code selon chaque critère.

4 Rendu

Le rendu du devoir comprend au minimum deux fichiers tels que mentionnés dans ce document, c'est-à-dire *name.hs* et *type.hs*. Les fichiers en question doivent inclure un entête - un commentaire Haskell - lequel indique le(s) auteur(s) du code. Libre à vous de travailler seul ou en binôme. 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 en conséquence. Sauf avis contraire, il ne vous est pas permis d'utiliser des fonctionnalités du langage Haskell qui n'ont pas été présentées en cours. La date de rendu pour ce devoir est fixée au **mardi 30 mai 2023 à 10h25**. Aucun retard ne sera admis et la note de 2 vous sera automatiquement attribuée si cela devait se produire.

5 Conclusion

Ce cours est votre première expérience avec la conception de langages de programmation. Il est donc important pour vous de commencer l'implémentation relativement tôt. 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 le devoir. Toutefois, ce forum ne doit pas devenir le lieu pour déboguer votre code. En cas de problème, il est de votre responsabilité d'identifier la source du bug et de le corriger. À cet égard, les outils de debugging sont vos meilleurs alliés.

Bon travail !