Travail pratique #2

IFT-2035

3 juin 2024

1 Survol

Ce TP vise à améliorer la compréhension des langages fonctionnels en utilisant un langage de programmation fonctionnel (Haskell) et en écrivant une partie d'un interpréteur d'un langage de programmation fonctionnel (en l'occurence une sorte de Lisp). Les étapes de ce travail sont les suivantes :

1. Parfaire sa connaissance de Haskell.
2. Lire et comprendre cette donnée. Cela prendra probablement une partie importante du temps total.
3. Lire, trouver, et comprendre les parties importantes du code fourni.
4. Compléter le code fourni.
5. Écrire un rapport. Il doit décrire **votre** expérience pendant les points précédents : problèmes rencontrés, surprises, choix que vous avez dû faire, options que vous avez sciemment rejetées, etc... Le rapport ne doit pas excéder 5 pages.

Ce travail est à faire en groupes de 2 étudiants. Le rapport, au format LaTeX exclusivement (compilable sur `ens.iro`) et le code sont à remettre par remise électronique avant la date indiquée. Aucun retard ne sera accepté. Indiquez clairement votre nom au début de chaque fichier.

Ceux qui veulent faire ce travail seul(e)s doivent d'abord en obtenir l'autorisation, et l'évaluation de leur travail n'en tiendra pas compte. Des groupes de 3 ou plus sont **exclus**.

$e ::= n$	Un entier signé en décimal
$\mid x$	Une variable
$\mid (e_0\ e_1\ ...\ e_n)$	Un appel de fonction (*curried*)
$\mid (\mathsf{proc}\ (x_1\ ...\ x_n)\ es)$	Une fonction ; les arguments sont *curried*
$\mid (\mathsf{def}\ (d_1\ ...\ d_n)\ es)$	Ajout de déclarations locales
$\mid + \mid - \mid * \mid / \mid \mathsf{print}$	Opérations prédéfinies
$\mid (\mathsf{node}\ e_1\ e_2) \mid \mathsf{null} \mid (\mathsf{seq}\ e_1\ ...\ e_n)$	Construction de listes
$\mid (\mathsf{case}\ e\ b_1\ ...\ b_n)$	Filtrage sur les listes
$es ::= \epsilon \mid es\ e$	Séquence d'instructions
$b ::= (\mathsf{null}\ es) \mid ((\mathsf{node}\ x_1\ x_2)\ es)$	Branche de filtrage
$d ::= (x\ e)$	Déclaration de variable
$\mid ((x\ x_1\ ...\ x_n)\ es)$	Déclaration de fonction

FIGURE 1 – Syntaxe de Psil

2 Psil avec effets de bord

Vous allez travailler sur l'ajout d'effets de bord à Psil. La Figure 1 montre la nouvelle syntaxe du langage. Il y a fondamentalement deux changements :

— L'ajout de la nouvelle opération primitive print.

— L'ajout de la notion de *séquence* d'instructions.

Les constructions proc et def ainsi que les branches du case ont été ajustées pour permettre les séquences d'instructions au lieu de n'avoir qu'une seule expression dans leur corps.

Un changement non visible mais non moins important concerne def : dorénavant def n'autorise plus toujours la récursion : si toutes les définitions du def sont des fonctions, alors elles peuvent être (mutuellement) récursives et devraient toutes avoir des noms différents, mais sinon, chaque définition ne peut faire référence qu'aux définitions précédentes mais pas à elle même ni aux définitions qui suivent. Par exemple :

```
(def ((x 3))
  (def ((x (+ 5 x))
        (y 7))
    (+ x y)))
```

Devrait renvoyer 15 : le x de (+ 5 x) ne peut faire référence qu'au x extérieur (et on ne pourrait pas utiliser y à cet endroit, contrairement à ce qui était le cas avec la version de Psil du TP1).

3 Implantation

Une petite partie du travail a déjà été faite pour vous :

— Les type de Vop et de eval ont été mis à jour pour renvoyer un résultat de type IO Value au lieu de Value.
— Le type Lexp a été ajusté pour refléter les changements dans la syntaxe.
— L'environnement initial env0 a été complété avec la nouvelle primitive print.
— Le code de run, valOf, et evalSexp a été mis à jour pour tenir compte du nouveau type de eval.
— Les 4 premiers cas de eval ont été ajustés pour tenir compte du nouveau type de eval.

4 À faire

Il vous reste donc à ajuster s2l à la nouvelle syntaxe ainsi que terminer de mettre à jour eval. Pour cette raison, le code fourni a encore beaucoup d'erreurs de typage.

Tout comme pour le TP1, vous devez aussi fournir un fichier `tests.psil`, similaire à `exemples.psil`, mais qui contient au moins 5 tests que *vous* avez écrits (avec en commentaire la valeur de retour que vous pensez devrait être renvoyée ainsi que le texte qui devrait être imprimé comme effet de bord) et qui exercent les nouveaux éléments du langage. Les tests sont évalués sur les critères suivants :

— Ils doivent bien sûr être corrects.
— Ils doivent être suffisamment différents les uns des autres (et des exemples fournis) pour détecter des erreurs différentes.
— Votre implantation de Psil doit les exécuter correctement.

4.1 Remise

Pour la remise, vous devez remettre 3 fichiers (`psil.hs`, `tests.psil`, et `rapport.tex`) par la page Moodle (aussi nommé StudiUM) du cours. Assurez-vous que le rapport compile correctement sur `ens.iro` (auquel vous pouvez vous connecter par SSH).

5 Détails

— La note sera divisée comme suit : 30% pour *s2l*, 25% pour le rapport, 15% pour les tests et 30% pour *eval*.
— Tout usage de matériel (code, texte, ...) emprunté à quelqu'un d'autre (trouvé sur le web, ...) doit être dûment mentionné, sans quoi cela sera considéré comme du plagiat.
— Le code ne doit en aucun cas dépasser 80 colonnes.
— Vérifiez la page web du cours, pour d'éventuels errata, et d'autres indications supplémentaires.
— La note est basée d'une part sur des tests automatiques, d'autre part sur la lecture du code, ainsi que sur le rapport. Le critère le plus important

est que votre code doit se comporter de manière correcte. Ensuite, vient la qualité du code : plus c'est simple, mieux c'est. S'il y a beaucoup de commentaires, c'est généralement un symptôme que le code n'est pas clair ; mais bien sûr, sans commentaires le code (même simple) est souvent incompréhensible. L'efficacité de votre code est sans importance, sauf si votre code utilise un algorithme vraiment particulièrement ridiculement inefficace.