Rapport

SANTONI Thomas

November 17, 2024

1 Introduction

Toutes les parties ont des tests correspondants dans les fichiers testsEval.ml et testsInference.ml, il suffit de copier le code de la forme let () = ... présent sous des commentaires explicitant ce qu'on cherche à tester. Ces tests ont étaient générés avec ChatGPT, dont voici quelques exemples de prompt :

- " *code complet* créer un main qui va générer des tests d'inférence de type sur Izte et Iete)"
- " *code complet* créer un main pour tester l'inférence de type avec let (en utilisant la fonction inference)"
- " *code complet* génère moi un main qui va tester ltr_cbv_norm' sur mes nouveaux termes : entier, addition et soustraction"

Nous avons échangé quelques idées avec Mohamed Amine ZEMALI. Par exemple, l'utilisation d'une fonction déterminant l'égalité structurelle entre 2 types me vient de lui, ce qui m'a été très utile pour corriger mon unification.

2 Résumé

2.1 Partie 2

Toute cette partie est fonctionnelle, on peut la tester en copiant le main sous le commentaire "(* tests Partie 2 *)" dans le fichier testsEval.ml

La fonction utilisée pour tous les tests de normalisation puis d'inférence de type est ltr_cbv_norm qui est simplement la fonction de normalisation demandée. À la place de timeout, j'ai fait le choix d'implémenter mes fonctions partielles (de normalisation et d'unification) avec une variable de fuel, permettant de se limiter à un nombre d'étapes définies à l'avance. L'idée du fuel me vient d'un TME de SVP, ce n'est pas parfait mais ça me semblait plus cohérent avec la philosophie d'OCaml et plus "pur".

2.2 Partie 3

Toute cette partie est fonctionnelle. Pour la tester, voir le main sous "(* Tests partie 3 *)" dans testsInference.ml

2.3 Partie 4

Toute cette partie est fonctionnelle. Pour la tester, voir le main sous "(* Tests des entiers, additions et soustractions *)", "(* Tests des listes*)", "(* Tests d'Izte et Iete *)", (* Tests point fixe *) dans testsEval.ml et "(* Tests typage entier, addition et soustraction *)", "(* Tests typage listes, head, tail*)", "(* Tests inférence avec let *)" et "(* Tests inférence point fixe *)" dans testsInference.ml

2.4 Partie 5

Les sous-parties 5.1 et 5.2 sont fonctionnelles on peut les tester avec "(* Tests évaluation unit, ref, deref et assign *)" dans testsEval.ml et "(* tests d'inférence pour unit, ref, deref et assign *)" dans testsInference.ml

3 Points forts

- Maintenabilité et lisibilité du code : J'estime que mon projet est relativement ≪propre≫ et bien décomposé en plusieurs fonctions dont les rôles sont clairs. L'ajout de nouveaux traits à mon lambda-calcul était particulièrement aisé.
- Utilisation de fuel : Pour garantir la terminabilité de ma normalisation de termes et mon inférence de type, l'utilisation de fuel permet d'améliorer la robustesse de mon code et de facilement déterminer en combien d'étapes on obtient une normalisation ou inférence.

- Large gamme de tests

4 Points faibles

- Présentation des tests : Les tests devraient être réunis dans un seul fichier expliquant clairement les enjeux de chaque test
- Gestion de la mémoire : Dans la partie 5, pour les traits impératifs, j'initialise une variable globale regions. La mémoire devrait être passée en paramètre de ltr_ctb_step, ce qui empêcherait d'avoir à réinitialiser la mémoire entre chaque test. L'utilisation de variable globale n'est pas recommandée et nous éloigne de la "pureté" qu'on peut atteindre avec un langage fonctionnel.
- Faiblesses dans l'inférence de type avec le combinateur de point fixe : après plusieurs tests, je pense que mon inférence de type ne fonctionne pas parfaitement dans tous les cas.

5 Difficultés rencontrées

Lors de la réalisation de ce projet, dès la partie 3, j'ai eu des difficultées à réaliser mon algorithme d'unification, pour comparer 2 types, j'utilisais simplement "=" alors qu'il fallait créer une fonction permettant de comparer l'égalité structurelle entre 2 types, c'était simplement dû aux constructeurs de type. J'ai dû implémenter une autre fonction pour résoudre mes problèmes d'unification : substitutDansType qui applique les substitutions à chaque types de mon système d'équation (par exemple X1 = N).

L'unification du forall m'a également posé beaucoup de problèmes. En effet, la nécessité de générer de nouvelles variables de type pour l'inférence de type m'a beaucoup perturbé, car cela me paraissait arbitraire.

L'évaluation du combinateur de point fixe m'a posé beaucoup de problème, soit la fonction ne se réduisait pas lorsque je lui fournissais un argument, soit la récursion était infinie.