Compte rendu de projet TAS

Evaluateur-Typeur de Lambda-Calcul

Malek BOUZARKOUNA — 28706508

17 novembre 2024

Superviseur académique :
DEMANGEON Romain

Table des matières

1 Introduction

Ce projet consiste en la conception d'un évaluateur et d'un typeur pour le λ-calcul enrichi, implémenté en OCaml. L'objectif de ce rapport est de présenter l'avancement du projet et d'expliquer les différentes étapes de son développement.

1.1 Aide et Contributions

Durant ce projet, plusieurs aides ont été apportées pour faciliter son bon déroulement. Tout d'abord, Yanis Tabellout m'a expliqué la gestion des adresses dans la section 5.1, ce qui m'a permis de mieux comprendre et structurer cette partie, contribuant ainsi à la clarté du code.

Pour certains aspects techniques, notamment les termes complexes, j'ai utilisé des prompts comme :

"Peux-tu me définir les termes techniques de cette partie et me donner la structure du code ?"

Cela m'a permis d'obtenir des explications claires et d'avoir une meilleure compréhension des concepts sous-jacents.

Ou encore pour le débogage :

"Peux-tu relever les incohérences logiques ou des cas que je n'aurais pas vus ?"

Cela m'a permis d'identifier rapidement des erreurs et des scénarios manquants, ce qui a grandement amélioré l'efficacité du processus de développement.

De plus, une partie des tests a été générée et validée avec l'assistance de modèles d'IA tels que ChatGPT et Claude 3.5, ce qui a permis de couvrir un large éventail de cas de tests.

1.2 Références et Ressources

Parmi les ressources utilisées, le travail de Christophe Deleuze sur l'évaluateur de λ-calcul en OCaml a servi de référence précieuse [2].

De plus, des discussions et solutions trouvées sur Stack Overflow, ont été explorées pour approfondir notre compréhension des concepts et affiner notre approche [1].

2 Parties traitées

2.1 Partie 2 : Évaluateur pour un λ-calcul pur

Toutes les fonctionnalités ont été implémentées et testées : représentation des termes en λ-calcul, pretty printer, alpha-conversion, substitution, réduction Call-by-Value, et normalisation (avec gestion des divergences). Le travail a été facilité par le travail de Christophe Deleuze sur l'évaluateur de calcul λ en OCaml sur cette partie [2].

2.2 Partie 3 : Types simples pour le λ-calcul

La représentation des types, le pretty printer, et la génération des équations de typage ont été réalisés et testés. Malgré les principales difficultés liées à l'inférence de type, notamment la gestion de la fonction `unify` et de la génération des équations, tout a été testé et fonctionne correctement. Les fonctions de vérification d'occurrence, substitution, et unification ont également été implémentées et validées.

2.3 Partie 4 : λ-calcul enrichi et polymorphe

2.3.1 4.1 Évaluateur

La syntaxe des termes et l'évaluateur ont été mis à jour pour inclure les entiers, les listes, les opérateurs de branchement, et l'opérateur de point fixe. Le traitement des opérateurs de listes a posé des difficultés, notamment dans les branchements. Tout a été testé, y compris les fonctions polymorphes et la fonction factorielle.

2.3.2 4.2 Types

La syntaxe des types a été mise à jour pour gérer les entiers, les listes et le let-polymorphisme. La génération d'équations a été adaptée pour gérer les opérateurs, les entiers, et les branchements. Une fonction de généralisation a été implémentée. L'unification a été modifiée pour gérer les types $\forall$ et les listes. Les tests ont été réalisés avec succès, incluant les fonctions polymorphes et la fonction factorielle. Les difficultés rencontrées dans la partie 4.1 se sont également présentées dans cette section, en particulier lors du traitement des branchements et de l'unification.

2.4 Partie 5 : Traits impératifs

2.4.1 5.1 Évaluateur

La mise à jour de la syntaxe, du pretty printer et des constructeurs a été effectuée pour inclure les nouveaux opérateurs liés aux traits impératifs (déréférencement, création de régions et assignation). La mise à jour de l'évaluateur a également été réalisée, notamment pour la gestion de la réduction des nouveaux termes. Des difficultés ont été rencontrées lors de l'implémentation du traitement des régions et de la gestion de l'état, en particulier pour la gestion des adresses de mémoire. Toutefois, après avoir demandé des explications à Yanis Tabellout sur son approche, notamment en ce qui concerne les adresses, j'ai pu surmonter ces obstacles. Tout a été testé et fonctionne correctement.

2.4.2 5.2 Types

La mise à jour des types a été réalisée pour inclure le type `unit` et le constructeur de type `Ref T`. La génération des équations a été adaptée pour ces nouveaux types, et l'unification a été mise à jour pour traiter les constructeurs de types. Quelques difficultés ont été rencontrées, notamment dans l'adaptation de l'unification pour ces nouveaux types, mais tout a été testé et fonctionne.

2.4.3 5.3 Polymorphisme faible

La gestion du polymorphisme faible a été l'aspect le plus complexe de cette section. L'introduction de la notion de non-expansivité pour les termes et de polymorphisme faible pour les types a été difficile à intégrer dans le système de typage. Ce travail a nécessité des ajustements dans le typeur pour assurer un système de type correct pour le langage. Bien que le terme `let l = ref [] in let l := [(^x.x)] in (hd !l) + 2` soit "moralement mal-typé", le système a été mis à jour pour le rejeter, et tout fonctionne maintenant après des tests.

2.5 Partie 6 : Aller plus loin

- **Lexer/Parseur pour le λ-calcul** : Un lexer et un parseur ont été implémentés pour analyser et transformer les termes en λ-calcul en une structure compréhensible pour l'évaluateur. Cela permet de mieux gérer la syntaxe des termes et de préparer le projet à une gestion plus complexe de la syntaxe. La gestion des erreurs de syntaxe a également été intégrée.
- **Gestion des exceptions** : Une gestion des exceptions a été partiellement mise en place. Bien que des mécanismes de base aient été introduits pour intercepter certains types d'erreurs, une gestion plus détaillée et complète des exceptions pourrait être implémentée pour couvrir davantage de scénarios d'exécution.

3 Force et Faiblesse

3.1 Points forts

- **Implémentation robuste de l'évaluateur CBV** : L'évaluateur fonctionne correctement dans la plupart des scénarios, avec une gestion précise des substitutions et de l'alpha-conversion.
- **Bonne gestion des substitutions et de l'alpha-conversion** : La gestion des substitutions a été bien implémentée, et les problèmes de capture de variables sont efficacement évités grâce à l'alpha-conversion.
- **Tests unitaires couvrant les fonctionnalités essentielles** : Des tests unitaires ont été réalisés pour vérifier le bon fonctionnement des principales fonctionnalités de l'évaluateur et du système de types.
- **Structure du projet bien organisée avec Dune** : L'utilisation de Dune pour la gestion du projet permet une organisation claire et une gestion efficace des dépendances, facilitant la compilation et l'exécution des tests.

3.2 Points faibles

— **Code parfois redondant** : Certaines parties du code peuvent être optimisées, car des sections similaires ont été réécrites à plusieurs endroits.

— **Clarté du code et des types dans les sections typage** : La gestion des types et certaines parties du code dans les sections liées à l'inférence de type peuvent manquer de clarté, en particulier lorsqu'il s'agit de la gestion des équations de typage et de l'unification.

Références

[1] Stack Overflow Contributors. *How to implement lambda calculus in OCaml ?* Accédé : 2024-11-17. 2018. URL : https://stackoverflow.com/questions/48049472/how-to-implement-lambda-calculus-in-ocaml.

[2] Christophe Deleuze. *Lambda-calculus evaluator in OCaml.* Accédé : 2024-11-17. n.d. URL : https://raw.githubusercontent.com/wiki/cdeleuze/lambda.ml/lambda.pdf.