Rapport Évaluateur-Typeur Lambda-Calcul

Yacine KESSAL (21311739)

17 novembre 2024

Table des matières

1 Introduction

L'objectif de ce projet est de développer un évaluateur et un typeur pour un λ-calcul, puis de l'enrichir progressivement au cours des différentes étapes. Le projet est structuré en plusieurs parties :

- **Partie 2 :** Implémentation d'un λ-calcul de base avec des vars, des applications et des abstractions.
- **Partie 3 :** Base du Typeur de λ-calcul.
- **Partie 4 :** Extension de l'evaluateur/typeur avec les listes, let, IfZero, IfEmpty ...
- **Partie 5 :** Intégration des références et de la mémoire avec les opérations `ref`, `deref` et `assign`.
- **Partie 6 :** Choix d'éxtension

2 Partie 2 : Implémentation du λ-calcul de base

- Complétée et testée
- Pas de points faibles à lister cette partie marche correctement
- **Résumé :**
  - Définition des types abstraits et des structures pour représenter le λ-calcul de base.
  - Implémentation des fonctions de substitution et de conversion alpha.
  - Les tests montrent que les implementations sont correctes et cohérentes.
- **Quelques tests :**

```
let k = Abs("x", Abs("y", Var "x"))
let identity = Abs("x", Var("x"));;
let sii = App(App(s, identity), identity)
let one = Abs("f", Abs("x", App(Var("f"), Var("x"))));;
```

3 Partie 3 : Base du Typeur de λ-calcul.

- Complétée et testée
- Pas de points faibles à lister cette partie marche correctement
- **Résumé :**
  - Définition des types simples
  - Implémentation de la génération d'équations et de l'algorithme d'unification
  - Implémentation des fonctions de substitution et d'inférence de type

Quelques tests :

```
let test_typage_entier () =
  let env = [] in
  let term = Int 42
;;
 Résultat : Type de Var 'x': T1
```

4 Partie 4 : Extension de l'evaluateur/typeur avec les listes, let, IfZero, IfEmpty.

— Complétée et testée
— Pas de points faibles à lister cette partie marche correctement
— **Résumé :**
  — Ajout des listes,let, IfZero, IfEmpty
  — Implémentation de la généralisation pour le polymorphisme de let

Quelques tests :

```
let test_typage_let_polymorphisme () =
  let term = Let ("id", Abs ("x", Var "x"), App (Var "id", Var "y")) in
  let env = [("y", TVar "B")]
;;
 Type inféré pour Let polymorphisme : B
```

5 Partie 5 : Intégration des références et de la mémoire avec les opérations.

— Complétée et testée
— Point faible : la question 5$\dot{3}$ ne marche pas malgrès les efforts fournis
— **Résumé :**
  — Ajout des listes,let, IfZero, IfEmpty
  — Implémentation de la généralisation pour le polymorphisme de let

Quelques tests :

```
let test_typage_assign () =
  let env = [] in
  let term = Assign (Ref (Int 10), Int 20)
;;

Type inféré pour Assign (Ref 10, 20) : unit
```

6 Extension réalisée

— Lexer et Parser

7 Difficultés rencontrées

— **Erreurs de parsing** : L'intégration du lexer et du parser a rencontré des difficultés au niveau de la reconnaissance du language
— **Polymorphisme faible**
— **Tests et débogage** : Les tests des expressions complexes ont révélé des bugs difficiles à localiser et je devais afficher toutes les etapes pour arriver à les trouver.