Présentation finale projet 2 : Fouine

Dziki Yanis et Gardies Vincent

Sommaire

- Organisation du codes et options
- Tests effectués
- Points notables de l'implémentation
- Pistes d'améliorations et conclusions

Organisation du code et options

- Une division en plusieurs fichiers :
  - inférence
  - unification
  - affichage
  - options
  - exceptions
  - lexer
  - parser
- Des options supplémentaires
  - `-tree`
  - `-showinf`
  - `-debug`
  - ...

Tests effectués

- Moulinette de tests pour l'évaluation
- Comparaison avec Ocaml pour le typage

Points notables de l'implémentation

Nous allons seulement parler des points qui diffèrent des préconisations des notes de cours (nous ne parlerons donc pas des listes, let et séquences d'expressions) :

- Choix généraux d'implémentation
- Implémentation du `Match ... With` / `Try ... With`
- Implémentation des fonctions récursives
- Choix pour l'implémentation du typage

Choix généraux d'implémentation

- Environnement
- Fonctions
- Références
- Exceptions

Match / Try ...With

- Type utilisé : `expr * (motif * expr) list`
- La fonction `filtre`
- `Try ... With`, un dérivé d'un `Match ... With` pour des exceptions.

Fonctions récursives

- Booléen de récursivité
- Gestion de l'environnement de la fonction

Inférence de types

- Le type t :
  - Var
  - T
  - Prime
  - None
- Un type récursif, dans l'esprit d'Ocaml
- La fonction `find_type : expr -> t`
- Gestion des variables de même nom

Unification

- Une boucle sur la liste d'inférence
- La fonction `type_fixed`
- La fonction `fusion`

Conclusions

- Une implémentation complexe qui nous a pris du temps
- Un manque de temps pour le polymorphisme