Typage et évaluation d'un petit langage de programmation (groupe 2)

1 But du projet

Le but de ce projet est de réaliser un outil permettant de réaliser le typage et l'évaluation d'un petit langage de programmation. Afin de rendre le projet plus "modulaire" ce langage sera progressivement enrichi.

2 Première partie : Les expressions

Dans cette première partie nous ne considérerons que des **expressions**.

2.1 Définition des expressions

L'ensemble des **expressions arithmético-logiques** noté dans la suite `Expr` est défini inductivement comme suit.

- une **variable** (de type OCAML `string`) est une expression **expression arithmético-logique**,
- Les flottants et les deux constantes `Vraie` et `Faux` sont des **expressions arithmético-logiques**
- si e est une **expression arithmético-logique** alors `not`(e) et `moins`(e) sont des **expressions arithmético-logiques**,
- si e_1 et e_2 sont des **expressions arithmético-logiques** alors `and`(e_1, e_2), `or`(e_1, e_2), `plus`(e_1, e_2), `mult`(e_1, e_2), `div`(e_1, e_2), `moins`(e_1, e_2) et `egal(`e_1`,`e_2`)` sont des **expressions arithmético-logiques**,
- enfin si e_1, e_2 et e_3 sont des **expressions arithmético-logiques** alors `if` e_1 `then` e_2 `else` e_3 est une **expression arithmético-logique**.
- si e_1 et e_2 sont des **expressions arithmético-logiques** et si `x` est une variable alors `let x =` e_1 `in` e_2 est une **expression arithmético-logique**,

Certaines expressions ainsi constructibles sont sémantiquement incorrectes au sens ou elles ne sont pas "bien typées". Ainsi par exemple, la construction: `if` 0 `then Vrai else 3` est invalide à plusieurs titres : dans un `if` la première expression devrait être un "booléen" et les deux branches devraient avoir le même type.

2.2 Les types des expressions

Nous allons donc augmenter notre langage d'un système de type. Pour ce niveau du langage, seuls deux types seront nécessaires : les `flottants` et les `booléens`. Les règles de typage de notre langage seront les suivantes étant donné un environnement de typage (un ensemble de couples (variable,type)) que nous définirons plus précisément dans une prochaine section.

- Dans l'environnement Γ, une variable x possède le type t si le couple (x, t) est présent dans Γ
- les flottants ont toujours le type `flottant` dans tout environnement et les deux constantes `Vraie` et `Faux` ont toujours le type `booleen` dans tout environnement.
- Si e possède le type `booleen` dans Γ alors `not(`e`)` possède le type `booleen` dans Γ.
- Si e possède le type `flottant` dans Γ alors `moins(`e`)` possède le type `flottant` dans Γ.
- Si e_1 et e_2 sont de type `booleen` dans Γ alors `and`$(e1, e2)$ et `or`$(e1, e2)$ sont de type `booleen` dans Γ.
- Si e_1 et e_2 sont de type `flottant` dans Γ alors `plus`$(e1, e2)$, `mult`$(e1, e2)$, `div`$(e1, e2)$ et `moins`$(e1, e2)$, sont de type `flottant` dans Γ et `egal(`e_1`,`e_2`)` est de type `booleen` dans Γ.
- si e_1 est de type t dans Γ, que x n'apparaît pas dans Γ et si e_2 est de type t' dans $(x, t)U\Gamma$ alors `let x =` e_1 `in` e_2 est de type t' dans Γ.
- enfin si e_1, e_2 et e_3 sont respectivement de type *booleen*, t et t dans Γ alors `if` e_1 `then` e_2 else e_3 est de type t dans Γ.

2.3 Les valeurs des expressions

L'évaluation des **expressions arithmético-logiques** dans un environnement d'évaluation (un ensemble de couples (variable, valeur)) suit les définitions intuitives des opérations.

2.4 Les environnements

Un **environnement** est, comme nous l'avons déjà vu un ensemble de couples dont la première composante est une variable. Nous avons déjà utilisé en TP des environnements (d'évaluation) sous forme de listes de couples. L'un des inconvénients de ce type d'environnements à base de listes est le temps de recherche d'une variable. Il faut en effet parcourir toute la liste pour trouver une variable. Nous allons utiliser un type d'environnement plus efficace à base d'arbres binaires de recherche. Ayant besoin de plusieurs types d'environnements, vous utiliserez des ABR polymorphes.

2.5 Travail à fournir

1. Donnez une représentation du type des expressions arithmético-logiques.
2. Implantez le type des environnements et les fonction d'ajout et de recherche d'une variable dans ces environnements. Le type et les fonctions devront permettre de simuler à la fois les environnements de typage et d'évaluation.
3. Implantez une fonction permettant de typer une expression arithmético-logique dans un environnement de typage donné. Cette fonction lèvera une exception `Type_invalide` que vous aurez préalablement définie.
4. Implantez une fonction d'évaluation des expressions arithmético-logiques dans une environnement d'évaluation donné.

3 Première augmentation de la syntaxe du langage

Le langage précédent ne permet pas la définition de fonctions. Nous allons donc modifier la syntaxe des constructions `let` comme suit. En plus de la construction précédente, nous ajoutons la construction : `let` f `(`x`:`t`) =` e_1 `in` e_2 où f et x sont des variables, t est un type de base (voir plus loin) et e_1 et e_2 sont des expressions arithmético-logiques.

Afin de pouvoir, typer ces nouvelles expressions, nous modifions la syntaxe des types comme suit :

- Les types `flottant` et les `booleen` sont les types de base
- Un type est soit un type de base soit un type de la forme $t_1 \rightarrow t_2$ où t_1 et t_2 sont des types de base.

Le typage de la nouvelle expression s'effectue comme suit :

Si e_1 est de type t_2 dans $(x; t_1)U\Gamma$ et si e_2 est de type t_3 dans $(f, t_1 \rightarrow t_2)U\Gamma$ alors `let` f `(`x`:`t`) =` e_1 `in` e_2 est de type t_3 dans Γ.

L'évaluation de la nouvelle construction sera effectuée de la manière habituelle.

3.1 Travail à fournir

En changeant les types de données nécessaires, afin de garder une trace des questions précédentes, réimplantez les questions de la première partie.

4 Pour ceux qui ont fini : seconde augmentation de la syntaxe du langage

Augmentez le langage avec une construction `let` récursive.