Programmation Fonctionnelle, L3 Informatique, 2019-2020

Projet, vérification de type monomorphe

Le projet sera réalisé en binômes. Le programme doit être clair, commenté et utiliser les principes de la programmation fonctionnelle.

Contexte

Considérons le langage mini-ML, dont les expressions de type sont décrites par la syntaxe abstraite suivante :

<type>	::=	T	type de base (int, bool ou char)
	|	<type> `->` <type>	type des fonctions
	|	<type> `*` <type>	type des paires

Par exemple, `int * char -> int` est le type d'une fonction qui prend en argument un couple comprenant un entier et un caractère et qui retourne un entier.

Les expressions du langage mini-ML sont décrites par la syntaxe abstraite suivantes :

<expr>	::=	x	identificateur (nom de variable)
	|	c	constante
	|	op	opération primitive
	|	`fun` x `:` <type> `->` <expr>	abstraction de fonction
	|	<expr> <expr>	application de fonction
	|	(<expr>, <expr>)	construction d'une paire
	|	`let` x `:` <type> = <expr> `in` <expr>	déclaration de variable locale

Dans le cadre de ce projet, nous considérerons les constantes entières : 1, 2, 3, 4, 5, les constantes caractères : a, b, c, d, et les constantes booléennes true et false. Les opérations primitives ainsi que leurs types sont les suivantes :

+	`int * int -> int`
-	`int * int -> int`
<	`int * int -> bool`
>	`int * int -> bool`
if_then_else	`bool * (int * int) -> int`

Par exemple, `+ (3, 2)` est la somme de 3 et 2, `fun x : int -> + (x, 1)` est la fonction "successeur", et :

```
let max : int*int -> int =
fun xy : int * int -> if_then_else (> xy, xy) in
max (1, 3)
```

est la définition de la fonction "maximum" et son utilisation pour calculer le maximum de 1 et 3.

Certaines expressions de mini-ML sont bien typées. Il est alors possible de calculer leurs types. Par exemple, la constante `3` est le type primitif `int`, celui de l'opération primitive `+` est `int * int -> int`, et celui de l'expression `+ (3, 2)` est `int`. Si `x` est une variable de type `char`, alors `(x, 1)` est de type `char * int`, et l'expression `fun x : int -> + (x, 1)` est de type `int -> int`.

D'autres expressions comportent des erreurs. Par exemple, `< (2, c)` est mal typée, car `<` attend un argument de type `int*int`, alors que `(2, c)` est de type `int*char`. Ou encore, `(y, 1)` est incorrecte si `y` n'est pas une variable connue.

Le calcul des types, ou la détection des erreurs, peut être réalisé à partir des règles ci-dessous, où E est l'environnement qui associe à toute variable x son type $E(x)$, a, a1, a2 des expressions de mini-ML, et t, t1, t2 des types de mini-ML. On note la relation de typage $E \vdash a : t$, si l'expression a a le type t dans l'environnement E. Par extension, on note E_c l'environnement des constantes, ainsi $E_c(c)$ fournit le type de la constante c, et E_{op} l'environnement des opérations primitives, ainsi $E_{op}(op)$ est le type de l'opération op. Par exemple, $E_c(1) =$ `int` et $E_{op}(+) =$ `int * int -> int`.

$$E \vdash x : E(x) \ \ (var) \qquad E \vdash c : E_c(c) \ \ (const) \qquad E \vdash op : E_{op}(op) \ \ (op)$$

$$\frac{E \cup \{x : t1\} \vdash a : t2}{E \vdash \texttt{fun}\ x : t1 {-}{>} a : t1 {-}{>} t2} \ (fun) \qquad \frac{E \vdash a1 : t1 {-}{>} t2 \quad E \vdash a2 : t1}{E \vdash a1\ a2 : t2} \ (app)$$

$$\frac{E \vdash a1 : t1 \quad E \vdash a2 : t2}{E \vdash (a1, a2) : t1 * t2} \ (paire) \qquad \frac{E \vdash a1 : t1 \quad E \cup \{x : t1\} \vdash a2 : t2}{E \vdash (\texttt{let}\ x : t1 = a1\ \texttt{in}\ a2) : t2} \ (let)$$

Par exemple, le calcul du type de `+ (3, 2)` dans l'environnement vide se fait comme suit.

$$\frac{\dfrac{\emptyset \vdash + : \texttt{int} * \texttt{int} {-}{>} \texttt{int} \ \ (op) \qquad \dfrac{\emptyset \vdash 3 : \texttt{int} \ \ (const) \quad \emptyset \vdash 2 : \texttt{int} \ \ (const)}{\emptyset \vdash (3, 2) : \texttt{int} * \texttt{int}} \ (paire)}{\emptyset \vdash +(3, 2) : \texttt{int}} \ (app)$$

Travail à réaliser

Question 1
Définir les types OCaml, pour les types et les expressions du langage mini-ML.

Question 2
Définir une fonction "d'affichage" des expressions mini-ML qui prend une expression et retourne l'expression sous la forme d'une chaîne de caractères, écrite de manière habituelle.

Définir quelques exemples d'expressions mini-ML et vérifier leur forme avec la fonction précédente.

Question 3
Nous allons utiliser les listes d'associations OCaml pour représenter les environnements. Définir les environnements *Ec* et *Eop* comprenant respectivement les constantes et les opérations primitives du langage mini-ML.

Question 4
Définir une fonction de vérification de type, qui prend en argument un environnement et une expression mini-ML et retourne le type de l'expression si l'expression est bien typée et lève une erreur sinon. Cette fonction mettra en oeuvre les règles d'inférence précédentes.

Question 5 (bonus)
Utiliser la fonction précédente pour calculer le type des expressions mini-ML suivantes :

```
fun x : char -> let succ : int -> int = fun x : int -> + (x, 1) in (succ 1, x)
fun x : char -> let succ : int -> int = fun y : int -> + (y, 1) in (succ 1, x)
```

Ont-elles le même type ? Sinon, modifier la fonction précédente pour prendre en compte les variables homonymes.