Programmation Fonctionnelle, L3 Informatique, 2021-2022

Projet, typage monomorphe

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

Considérons le langage mini-ML typé, 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	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', les constantes booléennes true et false, et les opérations 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`.

Le calcul des types, peut être réalisée 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_p l'environnement des primitives, ainsi $E_p(c)$ fournit le type de la primitive c. Par exemple, $E_p(1) =$ `int` et $E_p(+) =$ `int * int -> int`.

$$E \vdash x : E(x) \ \ (var) \qquad E \vdash c : E_p(c) \ \ (primitive)$$

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

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

Par exemple, le calcul du type de `+ (3, 2)` dans l'environnement vide se fait comme suit :
(1) $\emptyset \vdash 3 :$ `int` (primitive)
(2) $\emptyset \vdash 2 :$ `int` (primitive)
(3) $\emptyset \vdash + :$ `int` $*$ `int` $->$ `int` (primitive)
(4) $\emptyset \vdash (3, 2) :$ `int` $*$ `int` (paire (1) et (2))
(5) $\emptyset \vdash +(3, 2) :$ `int` (app (3) et (4))

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 l'environnements E_p comprenant 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 monomorphe.

Question 5
Utiliser la fonction de vérification de type de la question 4 pour calculer le type des expressions mini-ML typé 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.