Institut National Universitaire Champollion

TD 1 - Types et Vérifications de types

Types de données, preuves
L3 INFO - Semestre 6

Exercice 1 - Environnement de typage

Quel est l'environnement de typage à l'issue de ces requêtes ?

```
let x = 3  ; ;
♯x  : int = 3
let plus = fun x y -> x + y  ; ;
♯plus  : int -> int -> int = <fun>
let x = 2 in x + x  ; ;
```

```
♯-  : int = 4
let y = 4  ; ;
♯y  : int = 4
plus x y  ; ;
♯-  : int = 7
```

Vérifications de types simples

Exercice 2 - Vérification de type

Vérifier le type des expressions suivantes dans l'environnement `Env = [(a,int) ;(f,int -> int)]`

1. `(f a)`
2. `fun (x  : int) -> f`
3. `fun (x  : int) -> (f a)`
4. `fun (x  : int) -> (f x)`
5. `(fun (x  : int) -> f) a`
6. `fun (a  : bool) -> (f a)`

Exercice 3 - Vérification de type

Vérifier le type des expressions suivantes sur l'environnement

`Env = [(b, bool) ; (x, int) ; (f, int -> int -> int) ; (p, int -> bool)]`

1. `(f x)`
2. `(f x 3)`
3. `fun (y  : int) -> (f x y)`
4. `fun (y  : int) -> (f x)`
5. `fun (y  : int) -> (p y)`
6. `fun (y  : int) -> (p b)`

Extensions Couple + Let

Exercice 3 - Extension Couple

1. Écrire une fonction `egale  : int -> int -> bool` qui teste l'égalité de deux entiers.

   ```
   egale 3 4  ; ;
   -  : bool = false
   ```

2. Vérifier le type des expressions suivantes :
   (a) `egale 3`
   (b) `fun (x :int) -> egale 3 x`
   (c) `(3 , fun (x  : int) -> egale 3 x)`

Exercice 4 - Extensions (couple, let) :

Vérifier les types des expressions suivantes dans l'environnement

`Env = [(a,bool) ;(f, int -> bool)]`

1. `(a, f 3)`
2. `(false, fun (x :int) -> x * x)`
3. `(fun (x :int) -> f x, a )`
4. `(f 3, 3)`
5. `let x = 3 in f x`
6. `let y = true in (f,y)`

Extension Type Somme

Exercice 5 - Type somme :

On considère toujours le type `arbre_bin` vu en cours :

```
type arbre_bin =
Feuille of int
| Noeud of int * arbre_bin * arbre_bin
```

1. Rappeler comment l'environnement de typage est modifié suite à la définition de ce type.
2. Vérifier le type des expressions suivantes :
   (a) `Feuille 3`
   (b) `Feuille true`
   (c) `Noeud (3, Feuille 4, Feuille 5)`
   (d) `fun (x : int) -> fun (y : int) -> Noeud(x+y, Feuille x, Feuille y)`
3. Vérifier le type de l'expression suivante :
   `match (Noeud (3, Feuille 1, Feuille 2)) with Feuille x -> x + 1 | Noeud (x, l1, l2) -> x+2`

Exercice 6 - Type somme :

On considère dans cet exercice le type `arb_int` défini par :

```
type arb_int =
IFeuille
| INoeud of int * Iarb_int * arb_int ;;
```

1. Comment l'environnement de typage est-il modifié ?
2. Vérifier le type des expressions suivantes :
   (a) `IFeuille true`
   (b) `INoeud (5, IFeuille, IFeuille)`

Polymorphisme

Exercice 7 - Polymorphisme :

1. Définir les types polymorphes suivants :
   (a) `p1_arbre_bin` avec des `Feuille` et `Noeud` d'un seul type.
   (b) `p2_arbre_bin` avec des `Feuille` d'un type `'a` et `Noeud` d'un type `'b`.
2. On se donne la fonction suivante :

```
let rec nds_interne (abin : 'a p1_arbre_bin ) = match abin with
Feuille _ -> [ ]
| Noeud (a,f1,f2) -> a : : (nds_interne(f1) @ nds_interne(f2)) ; ;
```

   Quel est le type de cette fonction ? Quelle différence avec :

```
let rec nds_interne = function
Feuille _ -> [ ]
| Noeud (a,f1,f2) -> a : : (nds_interne(f1) @ nds_interne(f2)) ; ;
```

3. Les types suivants sont-ils des instances des types `p1_arbre_bin` et `p2_arbre_bin` ? Lorsque c'est le cas, préciser quelle est la substitution.
   (a) `int p1_arbre_bin`
   (b) `int bool p1_arbre_bin`
   (c) `(int * bool) p1_arbre_bin`
   (d) `(int, bool) p2_arbre_bin`
   (e) `('a, int) p2_arbre_bin`
   (f) `int * bool p2_arbre_bin`
4. **Vérifier** le type de
   (a) `nds_interne (Feuille "BonneAnnée")`
   (b) `nds_interne (Noeud (1 , Feuille true, Feuille false))`
   (c) `nds_interne (Noeud ('a', Feuille 1 , Feuille 2.5))`

Rappel des règles

Constantes : Toute constante a son type "naturel" :

$$\frac{n \in \mathbb{Z}}{\texttt{Env} \vdash \texttt{n} \ : \ \texttt{int}} \qquad \frac{b \in \{true, false\}}{\texttt{Env} \vdash \texttt{b} \ : \ \texttt{bool}}$$

Variables : $\dfrac{\texttt{tp(x,Env) = T}}{\texttt{Env} \vdash \texttt{x} \ : \ \texttt{T}}$

où `tp(x, Env) = T` si `(x,T)` est la déclaration la plus à gauche dans `Env`

Abstraction : $\dfrac{\texttt{(x,A)} \ : : \ \texttt{Env} \vdash \texttt{e} \ : \ \texttt{B}}{\texttt{Env} \vdash \texttt{fun (x} \ : \ \texttt{A) -> e} \ : \ \texttt{A -> B}}$

Application : $\dfrac{\texttt{Env} \vdash \texttt{f} \ : \ \texttt{A -> B} \qquad \texttt{Env} \vdash \texttt{a} \ : \ \texttt{A}}{\texttt{Env} \vdash \texttt{(f a)} \ : \ \texttt{B}}$

Couple : $\dfrac{\texttt{Env} \vdash \texttt{e} \ : \ \texttt{T} \qquad \texttt{Env} \vdash \texttt{e'} \ : \ \texttt{T'}}{\texttt{Env} \vdash \texttt{(e,e')} \ : \ \texttt{T * T'}}$

Let : $\dfrac{\texttt{Env} \vdash \texttt{e} \ : \ \texttt{A} \qquad \texttt{(x} \ : \ \texttt{A)} \ : : \ \texttt{Env} \vdash \texttt{e'} \ : \ \texttt{B}}{\texttt{Env} \vdash \texttt{let x = e in e'} \ : \ \texttt{B}}$

Filtrage :

$$\frac{\begin{array}{c}\texttt{Env} \vdash \texttt{e} \ : \ \texttt{T} \\ \texttt{[(}x_1\texttt{ , }T_1\texttt{) , ..., (}x_n\texttt{ , }T_n\texttt{)] @ Env} \vdash \texttt{e'} \ : \ \texttt{T}\end{array}}{\texttt{Env} \vdash \texttt{match e with ...|}\ \texttt{C(}x_1\texttt{, ..., }x_n\texttt{) -> e'} \ : \ \texttt{T'}}$$

où `T` est un type somme, et `C of` T_1 `*` ... `*` T_n est l'un des constructeurs de `T`.

Polymorphisme (Modification de la règle d'application) :

$$\frac{\texttt{Env} \vdash \texttt{f} \ : \ \texttt{A -> B} \qquad \texttt{Env} \vdash \texttt{a} \ : \ \texttt{A } \sigma}{\texttt{Env} \vdash \texttt{(f a)} \ : \ \texttt{B } \sigma}$$