Test 1 : Typage et unification

Algorithmes, types de données et preuves, Année 2014/2015

1 Typage de programmes impératifs

Donner une règle de typage pour typer une boucle `for` simplifiée, de la forme

`for` (*i* = *e*1; *e*2; *i*++) {
  c
}

où i est une variable, $e1$ une initialisation de i, $e2$ une condition d'arrêt de la boucle et ++ l'opérateur d'incrément. c est le corps de la boucle.

2 Vérification de types

Vérifier dans l'environnement `Env = [(f, int -> int); (p, int -> bool)]` les types des expressions Caml suivantes :

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

3 Inférence de types

Inférez le type des expressions suivantes :
— `fun p -> (fun x -> (p x))`
— `fun p -> ((fun x ->p) x)`
— `fun f -> fun g -> fun x -> ((g f) (f x))`

4 Unification – Compréhension

Lors du filtrage dans un langage comme Caml, on a des problèmes d'unification $t_1 \stackrel{?}{=} t_2$ moins complexes que dans l'unification traditionnelle : Seulement l'un des termes (disons t_1) peut contenir des variables, et les variables sont toutes distinctes.

Par exemple, un problème de filtrage peut avoir la forme
`Node(x, Leaf l1, Leaf l2)` $\stackrel{?}{=}$ `Node(3, Leaf 4, Leaf 5)`
tandis que
`Node(x, Leaf y, Leaf y)` $\stackrel{?}{=}$ `Node(3, Leaf 4, Leaf 5)`
serait mal formé (deux occurrences de la variable `y` dans le motif), tout comme
`Node(x, Leaf l1, Leaf l2)` $\stackrel{?}{=}$ `Node(z, Leaf 4, Leaf 5)`
n'est pas possible (occurrence de la variable `z` dans le terme à filtrer).

Quelles règles de l'algorithme d'unification pouvez-vous simplifier dans ce cas, comment et pourquoi ?

5 Unification

Faites l'unification des termes suivants :

1. $f\ X \stackrel{?}{=} f\ 3$
2. $g\ a\ X \stackrel{?}{=} g\ X\ b$
3. $g\ Y\ (f\ Y) \stackrel{?}{=} g\ Y\ X$
4. $g\ Y\ (f\ X) \stackrel{?}{=} g\ f\ X$

A Règles de typage

Règles de typage pour un langage fonctionnel élémentaire :

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

$$\frac{tp(x, Env) = T}{Env \vdash x : T}$$

$$\frac{(x, A) :: Env \vdash e \ : \ B}{Env \vdash \texttt{fun}(x : A) \ \rightarrow \ e \ : \ A \rightarrow B}$$

$$\frac{Env \vdash f \ : \ A \rightarrow B \quad Env \vdash a \ : \ A}{Env \vdash (f\ a) \ : \ B}$$

B Algorithme d'unification

— *Delete :* $(\{t \stackrel{?}{=} t\} \cup E, S) \Longrightarrow (E, S)$

— *Decompose :* $((s_1\ s_2) \stackrel{?}{=} (t_1\ t_2) \cup E, S) \Longrightarrow (\{s_1 \stackrel{?}{=} t_1, s_2 \stackrel{?}{=} t_2\} \cup E, S)$

— *Fail :* $((s_1\ s_2) \stackrel{?}{=} c \cup E, S) \Longrightarrow fail$

— *Clash :* $(c_1 \stackrel{?}{=} c_2 \cup E, S) \Longrightarrow fail$
si c_1 et c_2 sont des constantes différentes

— *Eliminate :* $(X \stackrel{?}{=} t \cup E, S) \Longrightarrow (E[X \leftarrow t]; S[X \leftarrow t] \cup \{X = t\})$
si $t \neq X$ et $X \notin FV(t)$

— *Check :* $(X \stackrel{?}{=} t \cup E, S) \Longrightarrow fail$
si $t \neq X$ et $X \in FV(t)$