Lundi 10 janvier 2005
Durée : 3 heures
Documents autorisés

Examen : Logique, Déduction et Programmation
Master informatique 2004-2005

Examen

Le langage Caml

Exercice 1. Considérons le module `Ensemble` : son interface est

ensemble.mli

```
1 : type 'a t;;
2 : val ajouter : 'a -> 'a t -> 'a t ;;
```

et son implémentation est

ensemble.ml

```
1 : type 'a t = 'a list
2 : let rec ajouter a = function
3 :     [] -> [a]
4 :   | t::q -> if t = a then t::q else t::(ajouter a q);;
```

Après l'avoir compilé :

```
$ ocamlc ensemble.mli ensemble.ml
```

on essai de s'en servir :

```
$ ocaml
       Objective Caml version 3.08.0

# #load "ensemble.cmo";;
# Ensemble.ajouter 1 [];;
                      --
This expression has type 'a list but is here used with type int Ensemble.t
#
```

Expliquer la raison du dernier message d'erreur.

Exercice 2. Voici un petit programme Caml pour traiter les séquences, c.-à-d. les listes possiblement infinies :

seq.ml

```
 1 : type 'a seq = Nil | Cons of 'a * (unit -> 'a seq);;
 2 : let hd = function
 3 :     Cons(t,q) -> t
 4 :   | Nil -> failwith "hd";;
 5 : let tl = function
 6 :     Cons(t,q) -> q ()
 7 :   | Nil -> failwith "tl";;
 8 : let cons t q = Cons(t, fun () -> q);;
 9 : let from_list l = List.fold_right cons l Nil ;;
10 : let rec from k = Cons(k, fun () -> from (k + 1));;
11 : let rec take s n =
12 :   if n = 0 then [] else
13 :     match s with
14 :         Nil -> failwith "take"
15 :       | Cons(t,q) -> t::(take (q ()) (n-1));;
```

1. Calculer le type de chaque fonction définie dans le fichier `seq.ml`.
2. Dire ce qui est produit par l'évaluation de `take (from 30) 2`.
3. Considérons la définition du type `'a seq` :

```
type 'a seq = Nil | Cons of 'a * (unit -> 'a seq);;
```

Expliquer la raison pour laquelle on y trouve le type `unit` dans cette définition.
Par exemple, on pourra :
a) rappeler les conditions sous lesquelles une expression de la forme `Cons(e1,e2)` est un valeur,
b) rappeler les conditions sous lesquelles une expression de type `unit -> 'a` est un valeur,
c) définir un type `'a seqp` à la façon des listes :

```
type 'a seqp = Nilp | Consp of 'a * 'a seqp;;
```

d) écrire une fonction `fromp` qui est une modification de la fonction `from` au type `seqp`.
e) comparer et justifier l'évaluation de `from 1` et `fromp 1`.

4. Donner les étapes de l'évaluation (par valeur) de l'expression `take (from 30) 2`.

Unification et résolution

Exercice 3. Existe t-il un unificateur principal des couples de termes suivants ?

1. $f(g(x), y, k(x))$, $f(y, h(z), k(w))$,
2. $f(x, h(x))$, $f(g(y), z)$,
3. $g(h(x, y), z)$, $g(z, h(f(u), w))$.

Si oui, écrire cette substitution, si non justifier votre réponse.

Exercice 4.

1. Rappeler la forme générale de la règle de factorisation (à droite).
2. Appliquer la règle de factorisation à droite à la clause

$$Q(g(y), x) \Rightarrow P(f(x), y), P(y, f(x))$$

Exercice 5.

1. Rappeler la forme générale de la règle de résolution.
2. Appliquer la règle de résolution au couple de clauses suivantes :

$$P(x) \Rightarrow P(f(x)), R(y) \quad P(y) \Rightarrow Q(y, g(y))$$

Exercice 6. On a discuté, dans le cours, la structure d'un démonstrateur automatique. À un tel démonstrateur on passe en argument une théorie logique $\mathbb{T}$ sous la forme d'un ensemble de clauses. Le démonstrateur essaie de dériver la clause vide en saturant la théorie : il applique toutes les règles du calcul de la résolution à toutes clauses déjà produites pour en produire des nouvelles.
Pour chacun de trois cas suivants, dire si la théorie $\mathbb{T}$ est consistante ou non.

1. Le démonstrateur s'arrête par ce qu'il a dérivé la clause vide.
2. Le démonstrateur s'arrête par ce qu'il ne peut plus appliquer des règles du calcul.
3. Le démonstrateur ne s'arrête jamais : il produit un nombre infini de clauses.

Sémantique

Exercice 7.

1. Définir en Caml (Camllight ou Ocaml) un type `loc` pour y coder les locations du langage IML.
2. Définir en Caml un type `etat` pour y coder les états du langage IML.
3. Définir en Caml un type récursif `aexpr` pour y coder les expression arithmétiques du langage IML.
4. Écrire en Caml une fonction `eval`, dont le type est `eval : aexpr -> etat -> int`, telle que, pour toute expression `a : aexpr`, pour tout expression `e : etat` et pour tout `n : int`, on a

```
# eval a e;;
- : int = n
```

ssi $(a, e) \to n$ ssi $(e, n) \in \mathcal{A}\|a\|$.

Exercice 8. Soit L un ensemble partiellement ordonné, soit $f : L \to L$ une fonction monotone, et soit μ tel que

$$\begin{array}{rcll} & & f(\mu) \leq \mu & \\ f(x) \leq x & \Rightarrow & \mu \leq x\,, & \text{pour tout } x \in L\,. \end{array}$$

Démontrer que μ est un point fixe de f, c.-à-d. démontrer que $f(\mu) = \mu$.