Master d'informatique | **module TEP** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2004-2005**

Examen du 15 novembre 2004

1 - λ-calcul pur : terme de Maurey

Un entier de Church est représenté comme un terme dont le corps comprend n (si le nombre en question est n) applications successives d'une fonction à un argument. Voici quelques exemples d'entiers de Church :
- $\overline{0} = \lambda fx.x$
- $\overline{1} = \lambda fx.fx$
- $\overline{n} = \lambda fx.f^n x$ ce qui équivaut à $\lambda fx.(f(f \ldots (f\ x) \ldots))$ où f est appliquée n fois.

On étudie le terme de Maurey définit ainsi : $M = \lambda nm.(nF(\lambda x.a)(mF(\lambda x.b)))$
où $F = \lambda fg.(gf)$ et a et b sont deux variables libres.

1. Donner la forme normale des λ-termes suivants, et si oui indiquer une réduction permettant de l'obtenir.
   - $\overline{0}\ F$, $\overline{1}\ F\ (\lambda x.a)$, $\overline{2}\ F\ (\lambda x.b)$
   - Appliquer M au différents couples construit à partir des entiers $\overline{0}, \overline{1}, \overline{2}, \overline{3}$ (ex : $M\ \overline{1}\ \overline{2}$).
2. A quoi correspond le terme M pour ces premiers entiers ? Pouvez-vous indiquer le comportement de ce terme pour tout couple d'entiers de Church ?
3. (question facultative) Ecrire un λ-terme clos correspondant à la fonction *max* pour deux entiers de Church.

2 - Typage : les entiers de Church en O'Caml

Donner le type O'Caml, s'il existe, des déclarations et calculer les valeurs des expressions des phrases O'Caml suivantes :

1. `let r1 = function  f  -> function x -> f ( f x);;`
2. `let r2 = function n -> function f  -> function x -> f ( n f x);;`
3. `let r3 = function n -> n (function x -> x + 1) 0;;`
4. `r3 r1;;`
5. `r3 (r2 r1);;`
6. `let r4 = function n -> function m -> n r2 m;;`
7. `r3 (r4 r1 (r2 r1));;`
8. (question facultative) Prouver dans un λ-calcul simplement typé (avec let, *int* et `+` de type $int \to int \to int$) le type de `r3`.

3 - Surcharge et liaison tardive en Java

On suppose que le choix du type d'une méthode en cas de surcharge ne tient compte que du type des paramètres de la méthode. Soit le programme Java suivant :

```
class A {
    int m (A x) { System.out.println (" m : A1 "); return 1;}
    int m (B x) { System.out.println (" m : A2 "); return 2;}
    int m (C x) { System.out.println (" m : A3 "); return 3;}
    int o (C x) { System.out.println (" o : A  "); return 11;}
    int p (A x, B y) {System.out.println(" p : Aa "); return 21;}
    int p (B x, C y) {System.out.println(" p : Ab "); return 31;}

}

class B extends A {
    int m (A x) { System.out.println (" m : B1 "); return 4;}
    int m (B x) { System.out.println (" m : B2 "); return 5;}
    int m (C x) { System.out.println (" m : B3 "); return 6;}
    int o (B x) { System.out.println (" o : B  "); return 12;}
    int p (B x, B y) {System.out.println(" p : Ba "); return 41;}
    int p (C x, A y) {System.out.println(" p : Bb "); return 51;}
}

class C extends B {
    int m (A x) { System.out.println (" m : C1 "); return 7;}
    int m (B x) { System.out.println (" m : C2 "); return 8;}
    int m (C x) { System.out.println (" m : C3 "); return 9;}
    int o (A x) { System.out.println (" o : C  "); return 13;}
    int o (B x) { System.out.println (" o : C_b "); return 23;}
    int p (B x, B y) {System.out.println(" p : Ca "); return 61;}
}
```

```
class q3 {
    public static void main(String[] args) {
      A a = new A();
      B b = new B();
      C c = new C();
      B b2 = c;
      A a2 = b2;
      int r1,r2,r3;

      System.out.println("Question a");
      r1 = a2.m(a) + a2.m(b) + a2.m(c);
      r2 = a2.m(a) + a2.m(b2) + a2.m(a2);
      System.out.println("r1= "+r1+ " r2= "+r2);

      System.out.println("Question b");
      r1 = b.o(b2) + b2.o(b) ;
      r2 = c.o(a2) + c.o(b2) + c.o(c);
      System.out.println("r1= "+r1+ " r2= "+r2);

      System.out.println("Question c");
      r1 = a.p(c,b) + b.p(b,c) + a.p(b,c);
      r2 = a2.p(b,b) + b2.p(b,b) + c.p(b,b) ;
      r3 = a.p(b2,b2) + b2.p(b2,b2) + c.p(b2,b2);
      System.out.println("r1= "+r1+ " r2= "+r2+" r3="+r3);
    }
}
```

1. Pour chaque méthode, faire un tableau indiquant la classe de définition et le type des paramètres pour toutes les définitions de cette méthode.
2. Indiquer sur les appels de la méthode `main` les choix du type de la méthode surchargée et indiquer les différents affichages en expliquant le déroulement de l'exécution.
3. Ecrire un appel à la méthode `p` qui entraîne une erreur à la compilation qui n'arrive pas à choisir le type d'une méthode surchargée et justifier cett erreur.
4. On change la relation d'ordre pour la sélection du type d'une méthode surchargée. Cette relation prend en compte la classe de définition d'une méthode (en testant le produit de la classe de définition avec la signature de la méthode). Indiquer alors les différences de compilation sur ce même programme.

4 - Classes paramétrées

Soit la classe O'Caml `['a] queue` suivante :

```
class ['a] queue () =  object
  val mutable q:'a list=[]
  method enq x = q <- q@[x]
  method deq = match q with
     [] -> failwith"Empty"  | h::r -> q<-r; h
end
```

1. Donner le type O'Caml des méthodes `enq` et `deq`.
2. Ecrire une classe paramétrée Java (1.5) équivalente.
3. Construire deux instances `qa` et `qb` de cette classe ; `qa` est une queue contenant des noms (String) et `qb` des numéros (Integer).
4. Indiquer ce qui se passe à la compilation et à l'exécution de ce fragment de programme par rapport à la classe que vous avez écrite.

```
queue qq = qb;
qq.enq("Essai");
Object o = qq.deq();
String s = (String)o;

qq.enq(o);
qq.enq(new Integer(33));
System.out.println(qq.deq());
System.out.println(qq.deq());

qq.enq(o);
qq.enq(new Integer(12));
int r = qb.deq() + qb.deq();
System.out.println(r);
```