Master d'informatique | **module TEP** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2005-2006**

Examen du 10 décembre 2007

1 - λ-calcul : stratégie d'évaluation et typage

Soient les λ-termes suivants :
$T = \lambda xy.x$
$S = \lambda xyz.xz(yz)$
$\Delta = \lambda x.xx$
$I = \lambda x.x$
$Q_1 = S\ T\ T$
$Q_2 = S\ T\ (\Delta\Delta)\ I$

1. Réduisez quand cela est possible les λ-termes Q_1 et Q_2 suivant la stratégie d'évaluation retardée (l'argument est passé tel quel à la fonction) et la stratégie d'évaluation immédiate (l'argument est évalué avant d'être passé à la fonction).
2. Quelle est la stratégie d'évaluation d'Objective Caml ?
3. Les termes Q_1 et Q_2 sont-ils typables en Objective Caml ? Pourquoi ?
4. Donner les types de T, S et Q_1
5. Construire l'arbre de preuve du type de Q_1.

2 - Typage : partage d'environnement et fonctionnelles

Donner les schémas de type O'Caml, quand ils existent, des déclarations O'Caml suivantes et indiquer les erreurs de typage, quand elles ont lieu, en précisant la raison :

1. ```
   let f  = function  x -> x :: [] ;;
   let succ = function x -> x + 1;;
   let non = function x -> if x then false else true;;
   ```
2. ```
   let g = ref f;;
   let h = ref f;;
   let j = h;;
   ```
3. ```
   let aex1 = g := succ;;
   let aex2 = !g 1;;
   let aex3 = g := non;;
   let aex4 = !g true;;
   ```
4. ```
   let bex1 = h := non;;
   let bex2 = !h true;;
   let bex3 = h == g;;
   let bex4 = j := succ;;
   let bex5 = !j 1;;
   ```

3 - Surcharge et liaison tardive en Java

Soient les déclarations de classe suivantes :

```
class A {
    void m(A x){System.out.println ("cas A.m(A)");}
    void m(A x, B y){System.out.println ("cas A.m(A,B)");}
}
class B extends A {
    void m(B x){System.out.println ("cas B.m(B)");}
    void m(B x, B y){System.out.println ("cas B.m(B,B)");}
}
class C extends B {
    void m(A x){System.out.println ("cas C.m(A)");}
    void m(C x){System.out.println ("cas C.m(C)");}
    void m(B x, B y){System.out.println ("cas C.m(B,B)");}
    void m(C x, C y){System.out.println ("cas C.m(C,C)");}
}
```

et soient les déclarations de variable suivantes :

```
class Sur {
  public static void main(String[] args) {
      A a1 = new A();
      A a2 = new A();
      B b1 = new B();
      B b2 = new B();
      C c1 = new C();
      C c2 = new C();
      A a3 = b1;
      A a4 = c1;
      B b3 = c1;
...
}}
```

Dans les quatre questions suivantes, indiquer en le justifiant la signature de la méthode `m` que détermine l'algorithme de résolution de la surchage de Java (version >= 1.5) ainsi que la méthode exécutée.

1.
```
a1.m(b1,c1);
b1.m(b1,c1);
c1.m(b1,c1);
```

2.
```
a3.m(b1,c1);
a4.m(b1,c1);
b3.m(b1,c1);
```

3.
```
a1.m(a3,b3);
b1.m(a3,b3);
c1.m(a3,b3);
```

4.
```
a3.m(b1,b3);
a4.m(b1,b3);
b3.m(b1,b3);
```

4 - Style objet et fonctionnel en O'Caml et Java

Soit le programme O'Caml suivant :

```
class integer v =
object
  val i = v
  method m z = z + i
end;;

let oi = new integer 3;;
```

```
class boolean v =
object
  val b = v
  method m z = if b then true  else z
end;;

let ob = new boolean true;;
```

1. Donner le type de la fonction `f` suivante :

   ```
   let f x y =   x#m y ;;
   ```

2. Tracer les évaluations suivantes en indiquant les appels de méthodes et de fonctions :

   ```
   let aex1 = f oi 1;;
   let aex2 = f ob true;;
   ```

3. Donner le type de la fonction `g` suivante :

   ```
   let g x y z =  x#m (y#m z);;
   ```

4. Tracer les évaluations suivantes en indiquant les appels de méthodes et de fonctions :

   ```
   let bex1 = g oi oi 1;;
   let gex2 = g ob ob true;;
   ```

5. Comment simuler un tel mécanisme en Java ? Implanter si c'est possible votre solution ou indiquer pourquoi ce n'est pas possible.

5 - Classes et méthodes génériques en Java

Le programme Java suivant utilise les classes génériques pour implanter le modèle de conception «Visiteur» sur des formules logiques :

Visiteur

```
abstract class Visiteur <T> {
  abstract T visite(Constante <T>  c);
  abstract T visite(Non <T> n);
}
```

Formule

```
abstract class Formule <T>   {
  abstract  T accepte(Visiteur <T> v);
}
```

```
class Constante <T> extends Formule <T> {
  boolean  b;
  Constante(boolean b) {this.b = b;}
  Constante() {this.b = false;}
  boolean valeur(){return b;}
  T accepte(Visiteur <T> v) {return v.visite(this);}
}

class Non <T>  extends Formule <T>  {
  Formule <T> f;
  Non(Formule<T> f) {this.f = f;}
  Formule<T> sous_formule(){return f;}
  T accepte(Visiteur <T> v) {return v.visite(this);}
}
```

```
class VisiteurEval  extends Visiteur <Boolean>   {
  Boolean  visite(Constante <Boolean> c){return c.valeur();}
  Boolean  visite(Non <Boolean>  n){
      return (! (n.sous_formule().accepte(this)));
  }
}

class VG {
  static Formule<Boolean> vrai(){return new Constante <Boolean>(true);}
  static Formule<Boolean> faux(){return new Constante <Boolean>(false);}
  static Formule <Boolean> non(Formule <Boolean> f){
      return new Non<Boolean>(f);}

  public static void main(String[] args) {
    Formule <Boolean> f1 = vrai();
    Formule <Boolean> f2 = faux();
    Formule <Boolean> f3 = non(f2);
    VisiteurEval  v1 = new VisiteurEval ();
    VisiteurEval  v2 = new VisiteurEval  ();

    boolean b1 = f1.accepte(v1);
    System.out.println(b1);
    boolean b2 = f2.accepte(v2);
    System.out.println(b2);
    boolean b3 = f3.accepte(v2);
    System.out.println(b3);
}}
```

1. Indiquer les relations entre classes et expliquer le déroulement du programme principal.
2. Définir une classe `VisiteurTS <String>` qui convertit une formule en chaîne de caractère. Ecrire le programme principal pour trois formules : VRAI, FAUX et NON VRAI.
3. Le fait de paramétrer les formules en fonction du Visiteur accepté n'est pas satisfaisant. Pour cela on définit les classes abstraites suivantes :

   ```
   abstract class Visiteur <T> {
     abstract T visite(Constante c);
     abstract T visite(Non n);
   }
   abstract class Formule   {
     abstract <T> T accepte(Visiteur <T> v);
   }
   ```

   qui tout en étant génériques ne nécessitent pas de paramétrer la classe `Formule` mais seulement de définir une méthode générique `accepte`. Ecrire Les classes `Constante`, `Non` qui héritent de `Formule` et `VisiteurEval` qui hérite de `Visiteur <Boolean>`.
4. Modifier le programme principal pour tenir compte de ces modifications.
5. Implanter la classe `VisiteurTS` qui hérite de `Visiteur <String>`.
6. Indiquer ce que l'on peut factoriser dans le programme principal pour parcourir les formules VRAI, FAUX et NON VRAI par un visiteur d'évaluation et un visiteur de conversion en chaînes de caractères.