Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2013/2014

23 luglio 2014

1. Provare che la seguente grammatica è ambigua.

   ```
   Exp ::= foreach Exp do Exp | foreach Exp do Exp and Exp | ( Exp ) | empty | vector<Exp, Exp>
   ```

   Definire una grammatica non ambigua che generi lo stesso linguaggio.

2. (a) Definire, in modo diretto e senza parametro di accumulazione, la funzione

   ```
   erase : 'a -> 'a list -> 'a list
   ```

   che cancella un elemento da una lista ordinata **senza ripetizioni**.

   (b) Usando la funzione `erase` del punto precedente, e la funzione

   ```
   itlist : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b
   let rec itlist f a = function x::l -> itlist f (f x a) l | _ -> a;;
   ```

   definire la funzione

   ```
   first_only : 'a list -> 'a list -> 'a list
   ```

   tale che, prese due liste ordinate senza ripetizioni l_1 ed l_2, restituisca la lista (ordinata e senza ripetizioni) che contiene tutti e soli gli elementi di l_1 che non appartengono ad l_2.

   (c) Definire la funzione `first_only` in modo diretto, senza parametro di accumulazione.

3. Considerare le seguenti dichiarazioni di classi Java, contenute nello stesso package:

   ```
   class X {
       private String m(Number n){ return "Foo"; }
       protected String m(Integer i){ return "Bar " + m(42.0); }
   }
   class Y extends X {
       protected String m(Integer i){ return "Baz " + super.m(i); }
   }
   class Z extends Y {
       protected String m(Number... n){ return "Goo"; }
   }
   class Main {
       public static void main(String[] args) {
           X x = new X();
           Y y = new Y();
           Z z = new Z();
           ...
      }
   }
   ```

   Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Main` l'espressione indicata.

   Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

   (a) `out.println(x.m(0));`

   (b) `out.println(y.m(1));`

   (c) `out.println(x.m(100.5));`

   (d) `out.println(((X) y).m(2));`

   (e) `out.println(z.m(256.123));`

   (f) `out.println(z.m(3));`

4. Completare i metodi delle classi `SetUtil` e `Test`. Il metodo `multiplyAll` **deve essere definito** utilizzando la classe `Prod` e il metodo `SetUtil.fold`.

```
import java.util.Set;

interface Function<X, Y> { // Functions of type X -> Y
        Y apply(X x);    // applies this function to x (and returns the result of type Y)
}

interface SetFactory {
        <X> Set<X> newEmptySet(); // creates an empty set of X
}

class SetUtil {
        private final SetFactory setFactory; // use this to create empty sets
        public SetUtil(SetFactory setFactory) {
                this.setFactory = setFactory;
        }

        // returns true iff predicate p is true for all elements of s
        public <X> boolean all(Set<X> s, Function<X, Boolean> p) {/* completare */}

        // returns true iff predicate p is true for at least one element of s
        public <X> boolean any(Set<X> s, Function<X, Boolean> p) {/* completare */}

        // returns the results of applying f to all elements of s
        public <X, Y> Set<Y> map(Set<X> s, Function<X, Y> f) {/* completare */}

        // f is a curried function of type X -> Y -> X where the accumulator has type X
        // the method returns f (... (f (f initVal e_1) e_2) ...) e_n where s = {e_1, e_2, ..., e_n}
        public <X, Y> X fold(Set<Y> s, Function<X, Function<Y, X>> f, X initVal) {/* completare */}

        // returns the union of inSet1 and inSet2
        public <X> Set<X> union(Set<X> inSet1, Set<X> inSet2) {/* completare */}

        // returns the intersection of inSet1 and inSet2
        public <X> Set<X> intersect(Set<X> inSet1, Set<X> inSet2) {/* completare */}
}

public class Test {

        // defines the function f such that f x y = x * y
        static final class Prod implements Function<Integer, Function<Integer, Integer>> {
                @Override
                public Function<Integer, Integer> apply(final Integer x) {
                        return new Function<Integer, Integer>() {
                                /* completare */
                        };
                }
        }

        // returns the product of all elements of s
        // by using class Prod and method SetUtil.fold
        static Integer multiplyAll(Set<Integer> s) {/* completare */}
}
```