Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2011/2012

31 agosto 2012

1. Sia $\mathcal{L}$ il linguaggio generato dalla seguente grammatica a partire dal simbolo non terminale `Exp0`.

```
Exp0  ::= Atom || Exp0 | Atom
Atom ::= Id | Bit | ( Exp0 )
Id ::= x | y | z
Bit ::= 0 | 1
```

   (a) Dimostrare che la stringa `(x||y)||1` appartiene a $\mathcal{L}$, mostrando un albero di derivazione per essa.

   (b) Definire una grammatica **non ambigua** che estenda quella data aggiungendo l'operatore binario infisso `&&` in modo che abbia la precedenza sull'operatore `||` e sia **associativo da destra**.

2. Definire le seguenti funzioni in Caml.

   (a) Definire la funzione `sel1 : ('a * 'b) list -> 'a list` tale che

   `sel1 [(`a_1`,`b_1`);...;(`a_n`,`b_n`)] = [`a_1`;...;`a_n`]`

   Esempio:

```
# sel1 [("Helter Skelter",1968);("A Day in the Life",1967);("Blackbird",1968)];;
- : string list = ["Helter Skelter"; "A Day in the Life"; "Blackbird"]
```

   (b) Definire la funzione `filter : ('a -> bool) -> 'a list -> 'a list` tale che `filter` p l si valuti nella lista che contiene tutti e soli gli elementi e di l per cui p e è vero.

   Esempio:

```
# filter (fun x -> x > 1967) [1968;1967;1968];;
- : int list = [1968; 1968]
```

   (c) Usando le funzioni specificate ai punti (a) e (b), definire la funzione

   `sel1where2eq : 'a -> ('b * 'a) list -> 'b list`

   tale che `sel1where2eq` e `[(`a_1`,`b_1`);...;(`a_n`,`b_n`)]` si valuti nella lista che contiene tutti e soli gli elementi a_i per i quali $b_i = e$ è vero.

   Esempio:

```
# sel1where2eq 1968 [("Helter Skelter",1968);("A Day in the Life",1967);("Blackbird",1968)];;
- : string list = ["Helter Skelter"; "Blackbird"]
```

3. Considerare le seguenti interfacce e classi Java:

```
public interface IPredicate<T1, T2> {
    boolean isTrueOn(T1 el1, T2 el2);
}
public interface IPair<T1, T2> {
    public T1 getFst();
    public T2 getSnd();
}
import java.util.Iterator;
public interface IBinaryRelation<T1, T2> {
    public Iterator<T1> iterator1();
    public Iterator<Pair<T1, T2>> filteredIterator(IPredicate<T1, T2> pred);
}
import java.util.*;
public class BinaryRelation<T1, T2> implements IBinaryRelation<T1, T2> {
    final private List<Pair<T1, T2>> pairs;
    public BinaryRelation() {
        pairs = new LinkedList<Pair<T1, T2>>();
    }
    public BinaryRelation(Iterator<Pair<T1, T2>> extIt) {
        this();
        ListIterator<Pair<T1, T2>> intIt = this.pairs.listIterator();
        while (extIt.hasNext())
            intIt.add(extIt.next());
    }
    @Override
    public Iterator<T1> iterator1() {
        return new Iterator<T1>() {
            Iterator<Pair<T1, T2>> it = pairs.iterator();
            @Override
            public boolean hasNext() {
                // completare
            }
            @Override
            public T1 next() {
                // completare
            }
            @Override
            public void remove() {
                throw new UnsupportedOperationException();
            }
        };
    }
    @Override
    public Iterator<Pair<T1, T2>> filteredIterator(final IPredicate<T1, T2> pred) {
        return new Iterator<Pair<T1, T2>>() {
            ListIterator<Pair<T1, T2>> it = pairs.listIterator();
            {   // instance initializer executed just after
                // the initialization of the iterator it
                goToNext();
            }
            private void goToNext() {
                // find the next element of the iterator
                // that satisfies the predicate
                while (it.hasNext()) {
                    Pair<T1, T2> curPair = it.next();
                    if (pred.isTrueOn(curPair.getFst(), curPair.getSnd())) {
                        it.previous();
                        return;
                    }
                }
            }
            @Override
            public boolean hasNext() {
                // completare
            }
            @Override
            public Pair<T1, T2> next() {
                // completare
            }
            @Override
            public void remove() {
                throw new UnsupportedOperationException();
            }
        };
    }
}
```

La classe generica `BinaryRelation` implementa una relazione binaria come lista di coppie.

Il metodo `iterator1` crea un iteratore che restituisce progressivamente la prima componente di tutte le coppie contenute nella relazione.

Il metodo `filteredIterator` crea un iteratore che restituisce progressivamente tutte le coppie contenute nella relazione che soddisfano il predicato che viene passato come argomento.

(a) Completare la seguente classe.

```
public final class Pair<T1, T2> implements IPair<T1, T2> {
// completare
}
```

(b) Completare il seguente frammento di codice che crea un predicato che è soddisfatto se e solo se il secondo argomento non è minore di 1968.

```
IPredicate<String, Integer> pred1 = new IPredicate<String, Integer>() {
// completare
};
```

(c) Completare la definizione dei metodi `hasNext` e `next` della classe anonima dichiarata nel metodo `iterator1`.

(d) Completare la definizione dei metodi `hasNext` e `next` della classe anonima dichiarata nel metodo `filteredIterator`.

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class Point {
    int x = 0, y = 0;
    void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
public class RealPoint extends Point {
    float x = 0.0f, y = 0.0f;
    void move(int dx, int dy) {
        move((float) dx, (float) dy);
    }
    void move(float dx, float dy) {
        x += dx;
        y += dy;
    }
    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
import static java.lang.System.out;
public class Test {
    RealPoint rp = new RealPoint();
    Point p1 = new Point();
    Point p2 = rp;
    ...
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato.

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)
```
rp.move(0.5f, 0.5f);
out.println(rp);
```

(b)
```
rp.move(0.5f, 0.5f);
rp.move(1, 2);
out.println(rp);
```

(c)
```
rp.move(0.5f, 0.5f);
rp.move(1, 2);
out.println((Point) rp);
```

(d)
```
((Point)rp).move(0.5f, 0.5f);
out.println(rp);
```

(e)
```
p1.move(1, 2);
out.println(p1);
```

(f)
```
rp.move(0.5f, 0.5f);
p2.move(-1, -2);
out.println(p2);
```