Linguaggi e Programmazione Orientata agli Oggetti

Soluzioni della prova scritta del 21 giugno 2021

a.a. 2020/2021

1. (a) Per ogni stringa elencata sotto stabilire, motivando la risposta, se appartiene alla seguente espressione regolare e, in caso affermativo, indicare il gruppo di appartenenza (escludendo il gruppo 0).

```
(\s+)|(0[0-7]*)|(0x1|0x2|0x3)|([0-9][a-zA-Z]+)
```

   i. `"0x4"`
   ii. `"0 0"`
   iii. `"0x"`
   iv. `"0x1"`
   v. `"001"`
   vi. `"0a"`

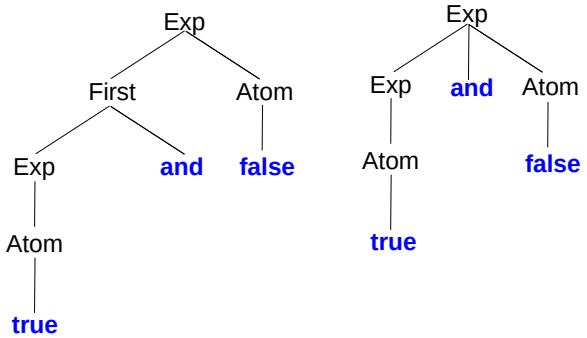
Soluzione:

   i. Gli unici gruppi che definiscono stringhe che iniziano con `"0x"` sono il tre e il quattro, ma le stringhe di tali gruppi non possono terminare con la cifra 4, quindi la stringa non appartiene all'espressione regolare.
   ii. L'unico gruppo che definisce stringhe che contengono spazi bianchi è l'uno, ma le stringhe di questo gruppo non possono contenere cifre decimali, quindi la stringa non appartiene all'espressione regolare.
   iii. L'unico gruppo che definisce stringhe che iniziano con la cifra `'0'` seguita da uno o più lettere minuscole o maiuscole è il quattro, quindi la stringa appartiene a tale gruppo.
   iv. L'unico gruppo che definisce stringhe che iniziano con la cifra `'0'` seguita dal carattere `'x'` e dalla cifra `'1'` è il tre, quindi la stringa appartiene a tale gruppo.
   v. L'unico gruppo che definisce stringhe che iniziano con la sequenza `"00"` seguita da una cifra ottale è il due, quindi la stringa appartiene a tale gruppo.
   vi. L'unico gruppo che definisce stringhe che iniziano con la cifra 0 seguita dalla lettera `'a'` è il quattro, quindi la stringa appartiene a tale gruppo.

(b) Mostrare che la seguente grammatica è ambigua.

```
Exp ::= Exp and Atom | First Atom | Atom
First ::= Exp and
Atom ::= false | true | ( Exp )
```

Soluzione: Basta esibire due diversi alberi di derivazione per una stessa stringa del linguaggio, per esempio `true and false`



(c) Modificare la grammatica definita al punto precedente in modo che **non sia ambigua** e che il linguaggio generato a partire dal non terminale `Exp` **resti invariato**.

Soluzione: La soluzione più semplice consiste nell'eliminare le seguenti produzioni che risultano essere ridondanti:

```
Exp ::= First Atom      First ::= Exp and
```

ottenendo così la grammatica equivalente (rispetto a `Exp`) definita qua sotto:

```
Exp ::= Exp and Atom | Atom
Atom ::= false | true | ( Exp )
```

2. Sia `cat : (string * string) list -> string list` la funzione così specificata:

   `cat [`(x_1, y_1)`; ... ;`(x_k, y_k)`] = [`x_1 ^ y_1`; ... ;`x_k ^ y_k`]`, dove $k \geq 0$ e ^ rappresenta l'operatore di concatenazione tra stringhe in OCaml.

   Esempi:

   ```
   cat [("hello"," world");("ciao ","mondo")]=["hello world"; "ciao mondo"]
   cat []=[]
   ```

   (a) Definire `cat` senza uso di parametri di accumulazione.

   (b) Definire `cat` usando `List.map: ('a -> 'b) -> 'a list -> 'b list`.

   Soluzione: Vedere il file `soluzione.ml`.

3. Completare la seguente classe di iteratori `CharSeqIterator` per iterare, nell'ordine convenzionale dal primo all'ultimo elemento, sui caratteri di una sequenza di tipo `java.lang.CharSequence`.

   Esempio:

   ```
   for (var ch : new CharSeqIterator("abc")) System.out.println(ch); // prints a b c
   for (var ch : new CharSeqIterator("")) System.out.println(ch); // no printed chars
   for (var ch : new CharSeqIterator("aBc")) System.out.println(ch); // prints a B c
   ```

   L'interfaccia predefinita `java.lang.CharSequence` (implementata da `java.lang.String`) contiene, tra gli altri, i metodi `char charAt(int index)` e `int length()`: il primo restituisce il carattere della sequenza che si trova all'indice `index` (gli indici partono da zero), il secondo calcola la lunghezza della sequenza.

   Codice da completare:

   ```
   import java.util.Iterator;
   import java.util.NoSuchElementException;
   import static java.util.Objects.requireNonNull;

   class CharSeqIterator implements Iterator<Character>, Iterable<Character> {

      // dichiarare i campi mancanti
      // invariant charSeq!=null

      public CharSeqIterator(CharSequence charSeq) {
         // completare
      }

      @Override
      public boolean hasNext() {
         // completare
      }

      @Override
      public Character next() {
         // completare
      }

      @Override
      public Iterator<Character> iterator() { return this; }
   }
   ```

   Soluzione: Vedere il file `soluzione.jar`.

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
   String m(double d) { return "P.m(double)"; }
   String m(float f) { return "P.m(float)"; }
}
public class H extends P {
   String m(long l) { return "H.m(long)"; }
   String m(int i) { return "H.m(int)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

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) `p.m(42)`

(b) `p2.m(42)`

(c) `h.m(42)`

(d) `p.m(42.0)`

(e) `p2.m(42.0)`

(f) `h.m(42.0)`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico **`int`** e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché **`int`** $\leq$ **`double`** e **`int`** $\leq$ **`float`**, entrambi i metodi di `P` sono accessibili e applicabili per sottotipo, ma viene scelto il metodo più specifico `m(`**`float`**`)` poiché **`float`** $\leq$ **`double`**.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(`**`float`**`)` in `P` e viene stampata la stringa `"P.m(float)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(`**`float`**`)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa `"P.m(float)"`.

(c) Il literal `42` ha tipo statico **`int`** e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): sia i metodi ereditati da `P`, sia quelli definiti in `H` sono applicabili per sottotipo dato che **`int`** $\leq$ **`double`**, **`int`** $\leq$ **`float`**, **`int`** $\leq$ **`double`** e **`int`** $\leq$ **`long`**, ma viene scelto il metodo più specifico `m(`**`int`**`)` poiché **`int`** $\leq$ **`long`** $\leq$ **`float`** $\leq$ **`double`**.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo di `H m(`**`int`**`)`; viene quindi stampata la stringa `"H.m(int)"`.

(d) Il literal `42.0` ha tipo statico **`double`** e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché **`double`** $\leq$ **`double`** e **`double`** $\not\leq$ **`float`**, l'unico metodo di `P` applicabile per sottotipo è `m(`**`double`**`)`;

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(`**`double`**`)` in `P` e viene stampata la stringa `"P.m(double)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo con segnatura `m(`**`double`**`)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa `"P.m(double)"`.

(f) Il literal `42.0` ha tipo statico **`double`** e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): poiché **`double`** $\not\leq$ **`long`** e **`double`** $\not\leq$ **`int`**, l'unico metodo applicabile per sottotipo è `m(`**`double`**`)` ereditato da `P` come spiegato nei due punti precedenti.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per lo stesso motivo del punto precedente viene stampata la stringa `"P.m(double)"`.