AltriEsercizi5 - Esercizio1

Esercizio (1)

a. *Si compili la classe Factorial discussa a lezione (mercoledì 13/04/16);*
b. *Si definisca e si compili una nuova classe Main che usa il metodo Factorial.Fact per calcolare e stampare il fattoriale di un intero n.*

Allo scopo si utilizzi il file Factorial incluso nell'allegato.

Esercizio (2)

a. *Si discuta l'estensione della classe Factorial in una nuova classe per MemozedFactorial;*
b. *Si mostri la nuova struttura del programma, la si compili e*
c. *la si applichi come in (b) dell'esercizio precedente*

AltriEsercizi5 - Esercizio3-hints

Esercizio

Sia C+ il linguaggio C esteso con il costrutto abstype visto a lezione (lucidi 12/4/2016). Si consideri in tale linguaggio, Il programma AbstractOpair.c+ nell'allegato, formato dalla definizione del tipo astratto absOPair e da una procedura main avente come corpo un codice "code".
Si assuma "code" un qualunque, corretto, codice C (o C+). Si chiede di dimostrare che se "code" introduce e usa un valore v, di tipo absOPair allora la proprietà:

$$\texttt{getL(v)} \leq \texttt{getR(v)}$$

è sempre vera.

Hint. Abbiamo visto a lezione (12/04/16) che gli ADT introducono un invariante di rappresentazione `I`. Ma `I` non è l'unico invariante che un ADT introduce. In particolare, la proprietà $P(\texttt{v}) \equiv \texttt{getL(v)} \leq \texttt{getR(v)}$ è anch'essa un invariante, sebbene lo sia sui valori (astratti) di tipo absOPair.
Si tratta quindi, di dimostrare che P è un invariante su absOPair. Come si procede?

Dimostrazione

AltriEsercizi5 - Esercizio3-Facts

Esercizio

Sia C+ il linguaggio C esteso con il costrutto abstype visto a lezione (lucidi 12/4/2016). Si consideri in tale linguaggio, Il programma AbstractOpair.c+ nell'allegato, formato dalla definizione del tipo astratto absOPair e da una procedura main avente come corpo un codice "code".
Si assuma "code" un qualunque, corretto, codice C (o C+). Si chiede di dimostrare che se "code" introduce e usa un valore v, di tipo absOPair allora la proprietà:

$$\texttt{getL(v)} \leq \texttt{getR(v)}$$

è sempre vera.

Hint. Abbiamo visto a lezione (12/04/16) che gli ADT introducono un invariante di rappresentazione `I`. Ma `I` non è l'unico invariante che un ADT introduce. In particolare, la proprietà $\texttt{P(v)} \equiv \texttt{getL(v)} \leq \texttt{getR(v)}$ è anch'essa un invariante, sebbene lo sia sui valori (astratti) di tipo absOPair.
Si tratta quindi, di dimostrare che `P` è un invariante su absOPair. Come si procede?

Dimostrazione
Fact1. Costruttori generano valori che soddisfano P
Abbiamo il solo mk:
$(\forall \texttt{x}, \texttt{y} \in [\texttt{int}])\ \texttt{P(mk(x, y))}$. Infatti, consideriamo il corpo di mk: 3 stm. Il primo alloca memoria per una coppia. Il secondo, se $\texttt{x} < \texttt{y}$ restituisce la coppia (x, y) che soddisfa P. Il terzo, è eseguito solo se $\texttt{y} \leq \texttt{x}$ e restituisce la coppia (y, x) che soddisfa P.
Fact2. Selettori/osservatori lasciano immutato il valore (e le sue proprietà)
Fact3. Produttori, eventualmente applicati a valori che soddisfano P, generano nuovi valori che soddisfano P
Non ne abbiamo.
Fact4. Modificatori, applicati a valori che soddisfano P, modificano in valori che ancora soddisfano P.
Abbiamo il solo putL:
$(\forall \texttt{x} \in [\texttt{int}], \forall \texttt{y} \in [\texttt{absOPair}])\ \texttt{P(y)} \Longrightarrow \texttt{P(y')}$, dove sia $\texttt{y}'$ il valore modificato dall'esecuzione di `putL(x, y)`. Infatti, consideriamo il corpo di putL: 1 stm. Se $\texttt{x} \leq \texttt{y} \rightarrow \texttt{max}$ allora $\texttt{y}'$ è la coppia $(\texttt{x}, \texttt{y} \rightarrow \texttt{max})$ che soddisfa `P`. Altrimenti y non è modificato e $\texttt{P(y')} == \texttt{P(y)}$

Da Fact1-Fact4 e dal fatto che absOPair è astratto, quindi gli unici usi di `v`, in "code", sono attraverso costruttori e operazioni, **segue** l'asserto su "code".

AltriEsercizi5 - Esercizio4

Esercizio

Si consideri Il programma AbstractOpair.c+ nell'allegato, scritto in C esteso con un costrutto per ADT. Si sopprimano tutte le righe con l'indicazione "//not for C" e si dica:
(a) Il programma ottenuto è un programma C che definisce un tipo `absOPair`*? (b) Sia "code" un qualunque, corretto codice C, e* `v` *un valore di tipo* `absOPair` *e* `P` *l'invariante dell'esercizio precedente. Allora,* `P(v)`*?: Provare che ciò non è sempre vero.*

Hint. (a) Banale. (b) Scrivere un codice "code" che falsifica la proprietà.

Dimostrazione

AltriEsercizi5 - Esercizio5-6

Esercizio (5)

a. *Si compili la classe AbstractCounter discussa a lezione (mercoledì 13/04/16);*

b. *Si discuta il comportamento del programma quando usiamo a seguente classe:*

```
public class UseOfAbstractCounter{
AbstractCounter A = new AbstractCounter();
A.c = 15;
}
```

Esercizio (6)

a. *Si compili la classe NewCounter discussa a lezione (mercoledì 13/04/16);*

b. *Si discuta il comportamento del programma quando usiamo la classe ACUse inclusa nell'allegato:*

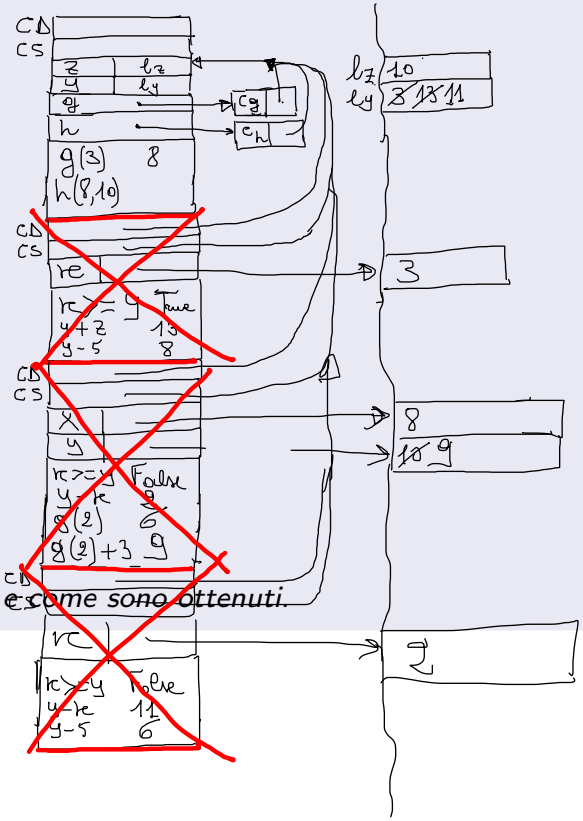
AltriEsercizi4 (lunedí 11/4/2016) Esercizio7

Esercizio

Si mostri la sequenza di AR generata dalla valutazione del seguente frammento di programma di un linguaggio a blocchi, scope statico, trasmissione per valore e per funzione con deep-binding, struttura C-like di comandi ed espressioni:

```
{
    int z = 10;
    int y = 3;
    int g(int x){;
      if (x >= y) y = y + z;
      else y = y − x;
      return y − 5;
    }
    void h(int x, int y){;
      if (x >= y) z = g(x);
      else y = g(y − x) + 3;
      print (x, y, z);
    }
    h(g(y), z);
    print (y);
}
```

In particolare si mostri quali valori sono stampati e come sono ottenuti.



8 9 10 11

uso scope dinamico

AltriEsercizi5 - Esercizio8

Esercizio

Si discuta un programma per la definizione di tabelle hash a liste di collisione e con funzione hash a divisione.

- L'esercizio6 precedente mostra che l'implementazione del fattoriale con memoization richiede tabelle di memorizzazione di modeste dimensioni e che l'aritmetica delle macchine attuali non operano con fattoriale oltre un piccolo naturale (prossimo a 20)
- Nondimeno, l'impiego di memoization può richiedere l'uso di tabelle hash per rappresentare la mappa finita delle applicazioni di funzione correntemente utilizzate dalla computazione. Limitandoci a funzioni monodiche, l'argomento una chiave, e l'immagine della funzione il valore associato alla chiave.
- **Tabelle a liste di collisione**: Hanno chiavi condivise da più argomenti (distr. unifor.) ed associato ad ogni chiave una lista di coppie $(a_n, v_n)...(a_1, v_1)$, dove a_n è l'n-esimo argomento con tale chiave utilizzato, nella computazione del programma, come argomento della funzione, e v_n è l'immagine calcolata.
- **Funzione hash a divisione**: `chiave(a) = P mod a`, dove P è un intero (primo non prossimo a potenza di 2).
- Implementare in Java per il prossimo lunedì 11/4/2016, con operazioni:
  `add, remove, find`.