9 gennaio 2006

Linguaggi di Programmazione

Nome e Cognome	
Matricola	
Corso di laurea	
Telefono	

1. Specificare la BNF di un linguaggio *Haskell*-like per la definizione di tipi, in cui ogni programma è composto da una sequenza non vuota di definizioni, come nel seguente esempio:

```
type Codice = String
type Costo = Int
type Articolo = (Codice,Costo)
type Articoli = [Articolo]
type Prodotti = Articoli
type T = [(Int, (Codice,Costo), String, [Prodotti], Bool)]
type Funz = (Int -> Int)
type Funz3 = (Int -> Int -> Int)
type Omega = (Int -> Articolo -> (String,Funz) -> [Funz3] -> Bool)
type Lista = [Omega]
type Tup = (Lista)
```

I tipi elementari sono **`Int`**, **`String`** e **`Bool`**. I costruttori di tipo sono la tupla, la lista e la funzione. Si assume che una funzione abbia almeno un parametro in ingresso. Il tipo funzione è sempre racchiuso tra parentesi tonde. La definizione di un tipo può coinvolgere nomi di altri tipi. I costruttori di tipo sono pienamente ortogonali tra loro.

2. Specificare la semantica denotazionale di una espressione logica definita dalla seguente grammatica:

expr → **true** | **false** | **id** | *and-expr*
and-expr → $expr_1$ **and** $expr_2$

Si assume che:

a) **id** rappresenti il nome di una variabile logica;
b) la valutazione di *and-expr* sia in corto circuito, con valutazione degli operandi da sinistra a destra;
c) sia disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
d) μ(**id**, *s*) = **?**, qualora il valore della variabile **id** non sia definito;
e) il linguaggio di specifica denotazionale non disponga della congiunzione logica (**and**).

3. Definire nel linguaggio funzionale *Scheme* la funzione `zip`, avente in ingresso due liste, `lista1` e `lista2`, che computa la lista di coppie di elementi che sono nella stessa posizione nelle rispettive liste, come nei seguenti esempi:

lista1	**lista2**	**(zip lista1 lista2)**
`()`	`()`	`()`
`()`	`(1 2)`	`()`
`(1 2 3)`	`()`	`()`
`(a b c)`	`(1 2 3 4)`	`((a 1)(b 2)(c 3))`
`(() 1 (a b))`	`(() (3 4) 5 zeta)`	`((()())(1 (3 4))((a b) 5))`

4. Definire nel linguaggio *Haskell* la funzione `last`, che restituisce l'ultimo elemento di una `lista` in ingresso, come nei seguenti esempi:

lista	**last lista**
`[]`	*non definita*
`[1,2,3]`	`3`
`["alfa"]`	`"alfa"`
`[(1,True),(2,False),(3,True)]`	`(3,True)`

5. Specificare in *Prolog* il predicato `intersezione,` avente tre argomenti di tipo lista, che risulta vero qualora la terza lista rappresenti l'intersezione insiemistica delle prime due liste. Si assume che ciascuna delle prime due liste non abbia duplicazione di elementi.

6. Nell'ambito del paradigma orientato agli oggetti, definire e giustificare (sulla base di un semplice esempio) la regola di covarianza del parametro di uscita nei metodi.