20 Gennaio 2015

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Specificare la definizione regolare relativa ai simboli **parola** e **numero**, sulla base dei seguenti vincoli lessicali:

- Una **parola** è composta da almeno tre caratteri alfanumerici, ha un prefisso (non vuoto) di lettere ed un suffisso (non vuoto) di cifre, come nei seguenti esempi: `sole25`, `k25`, `M55`, `K200`, `kL5`, `Aa2`, `w123456789`.
- Un **numero** è una costante intera o reale, opzionalmente preceduta dal segno. Se intera, non sono inclusi zeri non significativi. Se reale, sia la parte intera che la parte decimale (composta da almeno due cifre) non include zeri non significativi, come nei seguenti esempi: `0`, `-2`, `+0`, `+10`, `124`, `0.24`, `15.07`, `-123.450009`.

2. Specificare la grammatica EBNF di un linguaggio per la definizione di una lista (anche vuota) di protocolli di funzioni stile Haskell, come nel seguente esempio:

```
revint :: [Int] -> [Int]
reverse :: [a] -> [a]
member :: Eq a => a -> [a] -> Bool
createTuple :: Int -> Char -> (Int, Char)
select :: [Int] -> (Int -> Bool) -> [Int]
vsort :: (Ord a, Visible a) => [a] -> [Char]
mistery :: (Alfa a, Beta a) => (Eq a => a -> (Char,Bool)) -> [[[(Char->Int)]]]
```

I tipi atomici sono `Int`, `Bool` e `Char`. I costruttori di tupla e di lista sono rappresentati rispettivamente dalle parentesi tonde e quadre. Il protocollo di un parametro funzione è incluso tra parentesi tonde. Ogni tupla ha almeno due elementi. Ogni funzione ha almeno un parametro in ingresso. È possibile inserire opzionalmente un contesto, eventualmente con vincoli multipli (in tal caso i vincoli sono posti tra parentesi tonde). Nella EBNF, i parametri generici vengono specificati dal terminale **type-var**, mentre i nomi delle classi di tipi vengono specificati dal terminale **type-class**.

3. È data la seguente tabella degli operatori (con precedenza decrescente verso il basso):

Operatori	*Associatività*	*Ordine di valutazione*
+ , - , *, /	sinistra	Da destra a sinistra
=	non associativo	Da sinistra a destra
and, or	destra	Da sinistra a destra

e la seguente espressione: `x - y + v / w * z - k = (a + b) * c and d = e or f = g`

Assumendo che gli operatori logici siano valutati in corto circuito, si chiede di:

a. Inserire le parentesi nella espressione, indicando così l'associazione degli operandi agli operatori;

b. Rappresentare l'albero della espressione;

c. Specificare la semantica operazionale della valutazione della espressione sulla base dei seguenti vincoli:

- Il linguaggio di specifica include tutti gli operatori indicati nella tabella, l'assegnamento (**:=**), la negazione logica (**not**), le costanti booleane, la selezione a più vie ed il **return**;
- Ogni operatore (ad eccezione dell'assegnamento) non può essere applicato al risultato di altre operazioni.

4. È dato un linguaggio delle espressioni logiche definito dalla seguente grammatica BNF :

expr → **true** | **false** | **id** | $expr_1$ **nand** $expr_2$

in cui:

- **id** rappresenta il nome di una variabile logica;
- **nand** rappresenta l'operatore logico binario così definito:

  A **nand** B = **not** (A **and** B)

- la valutazione dell'operatore **nand** è da sinistra a destra.

Si chiede di:

- rappresentare la tabella di verità dell'operatore **nand**;
- sulla base della relativa tabella di verità, specificare l'espressione condizionale (non triviale) che definisce l'operatore **nand**;
- specificare la semantica denotazionale di una espressione logica sulla base della espressione condizionale.

Si assume che:

- sia disponibile una funzione `valore`(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*; (nel caso in cui la variabile non abbia un valore, `valore` restituisce `ERRORE`);
- il linguaggio di specifica denotazionale non disponga di alcun operatore logico ad eccezione della negazione (**!**).

5. Specificare nel linguaggio *Scheme* la funzione `sommatriple`, avente in ingresso una lista (anche vuota) di numeri, la quale computa la lista delle somme delle triple, come nei seguenti esempi (eventuali numeri in eccesso vengono sommati tra loro per comporre l'ultimo numero della lista risultato):

```
(sommatriple '()) = ()
(sommatriple '(1)) = (1)
(sommatriple '(1 4)) = (5)
(sommatriple '(1 2 3)) = (6)
(sommatriple '(2 4 6 8 10)) = (12 18)
(sommatriple '(2 4 6 8 10 12 20)) = (12 30 20)
```

6. Definire nel linguaggio *Haskell* la funzione `diversi` (protocollo generico incluso), la quale riceve in ingresso una lista e stabilisce se tutti gli elementi della lista sono diversi fra loro (per definizione, nel caso di lista vuota, `diversi` risulta vero). È richiesto che, se necessario, il protocollo includa anche il contesto.

7. Specificare in *Prolog* il predicato `sommatriple(L,S)`, in cui `L` è una lista (anche vuota) di numeri, mentre `S` è la lista delle somme delle triple, come specificato al punto **5**.

8. Utilizzando semplici esempi, illustrare la relazione che sussiste in *Haskell* tra funzioni polimorfe e funzioni overloaded.