2 dicembre 2009

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	

1. Specificare la grammatica BNF di un linguaggio *PHP*-like, per la specifica di prototipi di funzioni, come nella seguente frase:

```
function alfa($a, $b, &$c, $min=1, $max=20);
function beta(&$sole, $luna);
function gamma();
function delta($nome='stella', $cognome='cometa', $anni=21);
```

Ogni frase specifica una lista (non vuota) di prototipi di funzioni. La lista dei parametri formali (eventualmente vuota) elenca una serie di nomi di variabili (preceduti dal simbolo speciale **$**). Se un parametro è passato per referenza (invece che per valore), viene prefisso dal simbolo **&**. I parametri in coda alla lista possono avere un valore di default (intero o stringa). I parametri con valori di default non possono essere passati per referenza.

2. È data la seguente tabella degli operatori logici (per la quale si assume priorità decrescente dall'alto verso il basso), in cui tutti gli operatori binari sono valutati in corto circuito,

Operatore	*Descrizione*	*Tipo*	*Associatività*	*Ordine valutazione*
$\neg$	Negazione	unario	destra	-
$\vee$, $\wedge$	Disgiunzione, congiunzione	binario	sinistra	da destra a sinistra
$\Rightarrow$	Implicazione	binario	destra	da sinistra a destra

ed il seguente predicato: $a \wedge b \vee c \Rightarrow \neg x$

- Rappresentare l'albero del predicato.
- Specificare la semantica operazionale del predicato.

Il linguaggio di specifica operazionale è definito dalla seguente EBNF:

program → stat-list
stat-list → { stat **;** }$^+$
stat → assign-stat | if-stat | return-stat
assign-stat → **id :=** expr
expr → **true** | **false** | **id**
if-stat → **if id then** stat-list { **elsif id then** stat-list } [**else** stat-list] **endif**
return-stat → **return** expr

in cui **return** rappresenta l'istruzione di terminazione del programma di specifica operazionale con restituzione del risultato.

3. Definire nel linguaggio *Scheme* la funzione **`itera`**, la quale, ricevendo in ingresso una funzione unaria **`f`** , un valore **`x`** ed un numero naturale **`n`**, computa la lista di **`n+1`** elementi, ottenuta partendo da **`x`** ed applicando **`f`** ripetutamente al valore precedente, come nei seguenti esempi:

f	x	n	(itera f x n)
doppio	2	0	(2)
doppio	2	1	(4 2)
doppio	2	5	(64 32 16 8 4 2)
cdr	(1 2 3 4 5)	2	((3 4 5) (2 3 4 5) (1 2 3 4 5))
not	#t	3	(#f #t #f #t)

4. Definire nel linguaggio *Haskell*, mediante la notazione di pattern-matching, la funzione **`uguali`** che, avente in ingresso una lista, stabilisce se tutti gli elementi di tale lista siano uguali fra loro (per definizione, nel caso di lista vuota, **`uguali`** risulta vera). Si richiede anche la specifica completa del protocollo di tale funzione.

5. E' dato un fatto *Prolog* relativo ai tasselli di un domino, come nel seguente esempio:

```
tasselli([t(3,4),t(5,3),t(4,1),t(1,5)]).
```

Si chiede di specificare il predicato **`domino(X)`**, che risulta vero qualora **`X`** sia una lista composta da tutti i tasselli ordinati secondo le regole del domino. Nel nostro esempio avremmo:

```
?-  domino(X).
X = [t(4, 1), t(1, 5), t(5, 3), t(3, 4)] ;
X = [t(3, 4), t(4, 1), t(1, 5), t(5, 3)] ;
X = [t(1, 5), t(5, 3), t(3, 4), t(4, 1)] ;
X = [t(5, 3), t(3, 4), t(4, 1), t(1, 5)] ;
No
```

6. Dopo aver illustrato chiaramente il significato degli operatori *Prolog* **`=`**, **`==`**, **`is`**, si spieghi il meccanismo di interpretazione del seguente goal (indicando anche la prima risposta dell'interprete):

```
?- X = Y, X = 3+4, X == Y, Z is Y.
```