9 settembre 2009

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	

1. Specificare la grammatica EBNF di un linguaggio per la definizione di tipi di dati, come nel seguente esempio:

```
type Giorni is (Lun, Mar, Mer, Gio, Ven, Sab, Dom);

subtype Lavorativi is Giorni range {Lun..Ven};

subtype Indice is Integer range {1..100};

type Temperatura: Real;

type Temperature is array [1..10] of Temperatura;

type Tabella is array [1..7] of Temperature;

type Studente is record
  nome: String;
  cognome: String;
  matricola: Integer;
end record;
```

Ogni frase contiene almeno una definizione. I tipi primitivi sono **`Integer`**, **`Real`**, **`Bool`** e **`String`**. Il tipo enumerativo elenca i suoi elementi in una lista di identificatori. Si possono definire sottotipi di interi o enumerativi specificando il range. Esistono due costruttori (ortogonali) di tipo: array e record, i quali possono essere applicati unicamente a nomi di tipi, sottotipi o tipi primitivi. Si richiede di massimizzare i vincoli sintattici.

2. È data la seguente tabella di operatori, per la quale si assume priorità decrescente dall'alto verso il basso,

Operatore	*Tipo*	*Associatività*	*Ordine valutazione*	*Corto circuito*
`not`	unario	destra	-	-
`=`	binario	-	da sinistra a destra	no
`or`	binario	destra	da sinistra a destra	si
`and`	binario	sinistra	da destra a sinistra	no
`?:`	ternario	-	da sinistra a destra	si

e la seguente espressione:

```
a and b or not c = d ? x and y : x or y or z
```

- Rappresentare l'albero della espressione.
- Specificare la semantica operazionale della espressione.

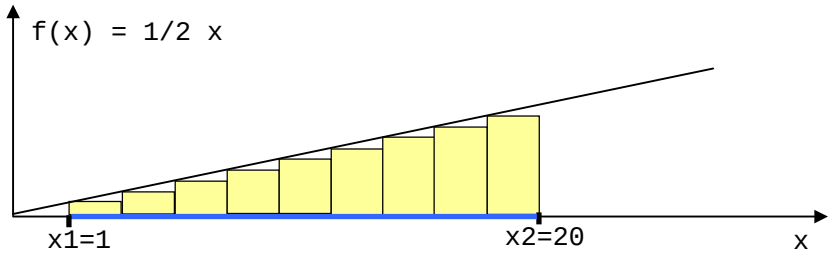
Il linguaggio di specifica operazionale è definito dalla seguente EBNF:

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

in cui **!** rappresenta la negazione logica, **&&** la congiunzione logica, **==** l'uguaglianza e **return** l'istruzione di terminazione del programma di specifica operazionale con restituzione del risultato.

3. Definire nel linguaggio *Scheme* la funzione **`media`** che, ricevendo in ingresso una **`lista`** (non vuota) di numeri, ne computa la media.

4. Definire nel linguaggio *Haskell* la funzione **`integrale`** (protocollo incluso), avente in ingresso una funzione **`f`** (che mappa un numero reale in un numero reale), i due estremi di integrazione **`x1`** e **`x2`** (**`x1`** < **`x2`**) e un intero **`n`** che rappresenta il numero di segmenti in cui dividere l'intervallo di integrazione [**`x1`**, **`x2`**]. Tale funzione computa l'integrale (approssimato, come sommatoria dell'area dei rettangoli) di **`f`** in [**`x1`**, **`x2`**], come nel seguente esempio:



in cui: **`integrale`** `f 1 20 9` `= 89.72222` (sommatoria di 9 aree di rettangoli con uguale base).

Si noti che la precisione del risultato aumenta con **`n`**, ad esempio: **`integrale`** `f 1 20 10000` `= 99.75184`

5. Specificare nel linguaggio *Prolog* il predicato **`media(L,M)`**, che risulta vero qualora **`M`** sia la media dei numeri nella lista **`L`**.

6. Illustrare, sulla base di alcuni esempi, le differenze semantiche tra i seguenti predicati *Prolog*:

- `X = Y`
- `X == Y`
- `X is Y`