18 Giugno 2012

Linguaggi di Programmazione

Cognome e nome	
Email	

1. Specificare la definizione regolare relativa ad un insieme (anche vuoto) di identificatori, come nel seguente esempio,

```
{Alfa, Beta22, Gamma_1_delta, Luna_Sole_stelle, Xilofono_1_2_345_omega}
```

sulla base dei seguenti vincoli lessicali:

- ogni identificatore è separato dal successivo da una virgola seguita da uno spazio;
- un identificatore inizia con una lettera maiuscola ed è seguito da zero o più caratteri alfanumerici;
- un identificatore può includere caratteri underscore;
- un underscore non può seguire un altro underscore e nemmeno essere l'ultimo carattere dell'identificatore.

2. Specificare la grammatica BNF di un linguaggio per la dichiarazione di variabili, come nel seguente esempio:

```
num, i, j, k: integer;
name, surname: string;
flag: boolean;
a: array [1..100] of integer;
m: array [10..20] of array ['a'..'z'] of boolean;
epsilon: set of integer;
omega: set of set of string;
```

Ogni frase contiene almeno una dichiarazione. I tipi atomici sono `integer`, `string` e `boolean`. I costruttori di tipo sono `array` e `set`. L'indice di un array può essere un intero o un carattere, il cui range è specificato nella dichiarazione. I costruttori di tipo sono ortogonali solo a se stessi.

3. Specificare la semantica operazionale dei seguenti operatori relazionali:

- `exists` [*p*] *R* : vero se e solo se esiste almeno una tupla in *R* che soddisfa il predicato *p*;
- `all` [*p*] *R* : vero se e solo se tutte le tuple di *R* soddisfano il predicato *p* oppure *R* è vuota.

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

$expr \rightarrow$ **true** | **false** | **id** | $expr_1$ **entails** $expr_2$

in cui:

- **id** rappresenta il nome di una variabile logica;
- **entails** rappresenta l'operatore di implicazione logica;
- la valutazione dell'operatore di implicazione è in corto circuito (da sinistra a destra).

Si chiede di:

- rappresentare la tabella di verità dell'operatore **entails**;
- sulla base della relativa tabella di verità, definire la regola di corto circuito per l'operatore **entails**;
- specificare la semantica denotazionale di una espressione logica.

Si assume che:

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

5. Specificare nel linguaggio *Scheme* la funzione `sommapotenze`, avente in ingresso una lista di interi, così definita:

$$sommapotenze([x_1, x_2, \ldots, x_n]) = \Sigma_{i=1}^{n}(x_i^i)$$

Nel caso limite di lista vuota, `sommapotenze` vale 0.

6. È data la seguente dichiarazione nel linguaggio *Haskell*, relativa ad espressioni di vettori di numeri reali:

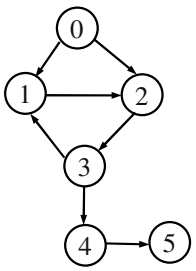
```
data Expr = Vec [Float]
          | Var String
          | Sum Expr Expr
          | Sub Expr Expr
          | Mul Expr Expr
          | Div Expr Expr

type State = [(String, [Float])]
```

in cui `Vec`, `Var`, `Sum`, `Sub`, `Mul`, e `Div` si riferiscono, rispettivamente, ad una istanza, una variabile, una somma di vettori, una differenza di vettori, una moltiplicazione di vettori e una divisione di vettori. Ogni operazione aritmetica su due vettori genera un vettore (di dimensione minima far i due vettori), in cui ogni numero *i*-esimo del vettore generato corrisponde al risultato dell'operazione aritmetica applicata agli elementi *i*-esimi dei due vettori operando.

Si chiede di definire in *Haskell*, mediante la notazione di pattern-matching, le seguenti funzioni (protocollo incluso):

- `matVec`: riceve in ingresso un vettore, un operatore aritmetico (`+`, `-`, `*`, `/`) ed un altro vettore; computa l'operazione aritmetica sui due vettori corrispondente all'operatore aritmetico;
- `eval`: riceve in ingresso una espressione di vettori e uno stato; computa il valore della espressione nello stato.

7. È data una base di fatti *Prolog* che definisce un grafo orientato ciclico, come nel seguente esempio:




```
nodi([0,1,2,3,4,5]).
archi([arco(0,1),
       arco(0,2),
       arco(1,2),
       arco(2,3),
       arco(3,1),
       arco(3,4),
       arco(4,5)]).
```

Si chiede di specificare il predicato `raggiunge(X,Y)`, vero se e solo esiste un cammino che parte dal nodo `X` e termina nel nodo `Y`. (Nota: Essendo il grafo ciclico, è necessario evitare loop nella ricerca ...).

8. Analizzare l'interpretazione delle seguenti interrogazioni *Prolog*, indicando (e motivando) per ognuna di esse la risposta dell'interprete:

- `?- A = B.`
- `?- A == B.`
- `?- A = 5*(12-7), A = B.`
- `?- A = B, A == B, A = 25.`
- `?- 14 is A+4.`
- `?- A is B+4.`