2 Luglio 2012

Linguaggi di Programmazione

Cognome e nome	
Email	

1. Specificare la definizione regolare relativa ad una lista (anche vuota) di record composti da tre elementi: una parola, un numero (senza segno) ed un insieme (anche vuoto, eventualmente disomogeneo) di parole o numeri,

```
[(sole,10,{}),(12_beta, 0.12,{0,12.35,21}),(Gamma_1_2_tre,128.09,{luna,2_stelle})]
```

sulla base dei seguenti vincoli lessicali:

- Non è ammessa spaziatura;
- Una parola è composta da caratteri alfanumerici ed underscore (questi ultimi non possono essere consecutivi, né iniziare o terminare la parola);
- Un numero può essere un intero o un reale (con parte decimale obbligatoria);
- Non sono ammessi numeri con prefisso di zeri non significativi.

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

```
program
  int a, b, c;
  string x, y;
begin
  a = 12;
  x = "alfa";
  if a == 10 then
    y = "beta";
  elsif "beta" == y then
    x = "gamma";
    y = "omega";
  else
    b = a;
  endif;
  while a == b do
    a = 10;
  endwhile;
end.
```

La sezione di dichiarazione delle variabili (che è opzionale) precede il corpo del programma. Quest'ultimo è composto da una lista non vuota di istruzioni (assegnamenti, selezioni a più vie, cicli). L'espressione di assegnamento è una costante o una variabile. Le condizioni sono espresse dal confronto di uguaglianza tra una variabile ed una costante o tra due variabili. Nella selezione a più vie, i rami `elsif` (in numero illimitato) ed `else` sono opzionali.

3. Specificare la semantica operazionale della seguente espressione relazionale:

```
select [a > b and c = 10] R
```

sulla base dei seguenti requisiti:

- L'operatore logico di congiunzione è valutato in corto circuito;
- L'operando dela selezione è una tabella i cui attributi sono di tipo intero;
- Nel caso di errore semantico, si restituisce la chiamata della funzione `errore` il cui argomento è un messaggio di errore;
- È disponibile la funzione ausiliaria `schema(X)`, che restituisce la lista dei nomi degli attributi della tabella `X`.

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

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

in cui:

- **id** rappresenta il nome di una variabile logica;
- **xor** rappresenta l'operatore di disgiunzione logica esclusiva (vero se e solo se un solo operando è vero);
- la valutazione dell'operatore **xor** è da sinistra a destra.

Si chiede di:

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

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 ad eccezione della negazione (`!`).

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

$$somme([x_1, x_2, \ldots, x_n]) = [y_i, y_2, \ldots, y_n],\ where\ y_i = \Sigma_{j=1}^{i}(x_j),\ i \in [1..n]$$

Nel caso di lista vuota, `somme` restituisce la lista vuota.

6. È data la seguente dichiarazione nel linguaggio *Haskell*, per la rappresentazione di alberi:

```
data Albero = Nodo Int [Albero]
```

Si chiede di codificare le seguenti funzioni (protocollo incluso):

- `concatena`: riceve una lista di liste di interi e restituisce la concatenazione delle liste di interi;
- `appiattisci`: riceve un albero di interi e restituisce la lista di interi contenuta nell'albero;
- `complementa`: riceve un albero di interi e restituisce un albero di interi isomorfo all'albero in ingresso, in cui tutti i numeri sono stati cambiati di segno.

7. Specificare nel linguaggio *Prolog* il predicato `bysum(X)`, in cui `X` è una lista di interi, il quale risulta vero quando ogni numero in `X` è la somma di due numeri qualunque in `X`.

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

- `?- X = Y.`
- `?- X == Y.`
- `?- X = 2+(24-13), X = Y.`
- `?- X = Y, X == Y, X = 18.`
- `?- 27 is X+4.`
- `?- X is Y+27.`