19 dicembre 2006

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	
Telefono	
Email	

1. Specificare la grammatica EBNF dei pattern di un linguaggio *Haskell*-like. Ecco alcuni esempi di pattern:

```
0
12
'a'
0.123
104.27896
True
x
num2
_
(x, y2, False,'w')
[1, 2, _, n, m]
testa:coda
x:y:[]
(2, True, ([1,2,3], x:[_, y], 'q'))
[(_,12), ('a', 0), ('b', 246), (_,n), _]
```

Ogni frase del linguaggio contiene un solo pattern. Semplici pattern possono essere costanti carattere (ad esempio `'a'`), costanti booleane (`True`, `False`) , costanti intere (senza zeri prefissi non significativi, ad esempio `102` ma non `0012`), costanti reali (ad esempio `12.34`, `0.025`, ma non `002.23`), nomi di variabili (necessariamente alfanumeriche, ad esempio `x`, `num3`, `area`, ma non `3num`), ed il carattere underscore `_`. Pattern più complessi possono essere specificati mediante i simboli di tupla (ad esempio `(3,x)`) o di lista (ad esempio `[1,2,n]`). Infine, un pattern può essere specificato mediante il costruttore di lista (ad esempio `n:[1,2,3]`). Sussiste piena ortogonalità tra tuple e liste. Si assumono i seguenti simboli lessicali (di cui non è richiesta la specifica): **true**, **false** (costanti boolenae), **char** (un qualsiasi carattere), **alpha** (un carattere alfabetico), **digit** (una qualsiasi cifra), **nonzero** (una cifra diversa da zero).

2. È dato il seguente frammento di grammatica BNF, relativo alla specifica di una lista di istruzioni di un linguaggio imperativo:

stat-list → *stat stat-list* | *stat*
stat → *assign-stat* | *if-stat* | *while-stat*

Si richiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio assumendo che siano disponibili le seguenti funzioni di mapping, che mappano una istruzione ed un certo stato `s` in un nuovo stato (eventualmente **errore**):

- M_{assign}(*assign-stat*, s)
- M_{if}(*if-stat*, s)
- M_{while}(*while-stat*, s)

In particolare, si richiede la specifica delle seguenti funzioni (che computano il nuovo stato, eventualmente **errore**):

- M_{stat}(*stat*, s)
- M_{list}(*stat-list*, s)

3. Definire nel linguaggio *Scheme* la funzione `apply`, avente in ingresso una funzione unaria `f` ed una `lista`. Si assume che `lista` rappresenti una espressione *Scheme*. La funzione `apply` computa l'applicazione di `f` al valore della espressione rappresentata da `lista`, come nei seguenti esempi:

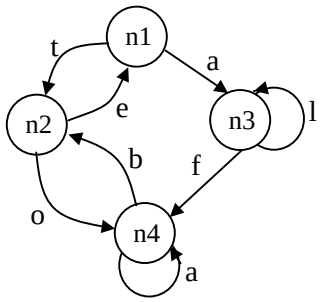
f	**lista**	**(apply f lista)**
number?	(length '(1 2 3))	#t
list?	(+ 1 2 3)	#f
fact	(+ 1 2 3)	720
quadrato	(length '(1 2 3))	9
reverse	(subst 'x 100 '(x 100 y))	(y x x)

4. Mediante la notazione di specifica basata sul pattern-matching, definire nel linguaggio *Haskell* la funzione `take` che, ricevendo una intero `n` ≥ 0 ed una `lista`, restituisce i primi `n` elementi di `lista`. Nel caso in cui `n` sia maggiore della lunghezza di `lista`, `take` restituisce `lista`. Ecco alcuni esempi:

n	**lista**	**take n lista**
3	[1,2,3,4,5]	[1,2,3]
4	"alfabeto"	"alfa"
1	[(25,"dicembre"),(1,"gennaio")]	[(25,"dicembre")]
7	[1,2,3]	[1,2,3]
0	[1,2,3]	[]
5	[]	[]
0	[]	[]
0	"alfa"	""

5. È data una base di fatti *Prolog* relativa alla specifica di un grafo, come nel seguente esempio:

```
nodi([n1,n2,n3,n4]).
archi([arco(n1,a,n3),
       arco(n3,l,n3),
       arco(n3,f,n4),
       arco(n4,a,n4),
       arco(n4,b,n2),
       arco(n2,e,n1),
       arco(n1,t,n2),
       arco(n2,o,n4)]).
```



Definire la regola per il predicato **`parola(P,L,N1,N2)`**, in cui **`P`** è una lista di caratteri, di lunghezza **`L`** ≥ 0, generata da un cammino nel grafo, che parte dal nodo **`N1`** e termina al nodo **`N2`**. Ecco alcuni esempi:

P	**L**	**N1**	**N2**	**parola(P,L,N1,N2)**
[]	0	n3	n3	true
[]	1	n3	n3	false
[a]	0	n1	n3	false
[a]	1	n1	n3	true
[a,l,f,a]	4	n1	n4	true
[a,l,f,a,b,e,t,o]	8	n1	n4	true

6. Illustrare le possibili scelte progettuali relative all'ambiente di referenziazione di un sottoprogramma passato come parametro ad un altro sottoprogramma.