26 novembre 2007

# Linguaggi di Programmazione

| | |
|---|---|
| *Nome e Cognome* | |
| *Corso di laurea* | |
| *Telefono* | |
| *Email* | |

1. Specificare la grammatica BNF di un linguaggio per la specifica di definizioni di protocolli di funzioni *Haskell*-like. Ecco un esempio di frase:

```
alfa :: Int -> Int
beta :: [Int] -> Bool
gamma :: Char -> (Int, Bool)
zeta :: [[(Char, (Bool, Int))]] -> Int
f10 :: (Int, Bool) -> [Char]
g20 :: Int -> (Int -> Bool -> Char) -> Bool
omega :: (Int -> (Int -> Bool)) -> [[Int]] -> (Bool, Char)
```

I tipi atomici sono **`Int`**, **`Bool`** e **`Char`**. I costruttori di tupla e di lista (ortogonali tra loro) sono indicati rispettivamente dalle parentesi tonde e dalle parentesi quadre. Un parametro di tipo funzione è specificato dal relativo protocollo tra parentesi tonde. Ogni funzione ha almeno un parametro di ingresso. Ogni frase contiene almeno una definizione.

2. Specificare la semantica denotazionale della seguente istruzione (ciclo a conteggio):

**for** n **do** L

in cui n rappresenta una costante intera. La lista L di istruzioni viene ripetuta n volte (zero volte se $n \leq 0$). Specificatamente, si definisca la funzione $M_f$(**for** n **do** L, s), in cui s rappresenta lo stato del programma, assumendo di avere a disposizione la funzione semantica $M_l$(L, s) : lista di istruzioni $\rightarrow$ nuovo stato o **errore**.

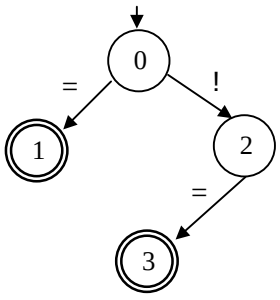
3. Definire nel linguaggio *Scheme* la funzione **`domino`** che, ricevendo una **`lista`** di coppie di numeri, stabilisce se tali coppie sono disposte come i tasselli nel gioco del domino (uguaglianza del secondo numero di ogni coppia con il primo numero della successiva coppia). Ecco alcuni esempi:

| **`lista`** | **`(domino lista)`** |
|---|---|
| `()` | `#t` |
| `((1 2))` | `#t` |
| `((1 2)(2 6))` | `#t` |
| `((1 2)(2 6)(6 34)(34 0)` | `#t` |
| `((1 3)(4 5)(5 7))` | `#f` |

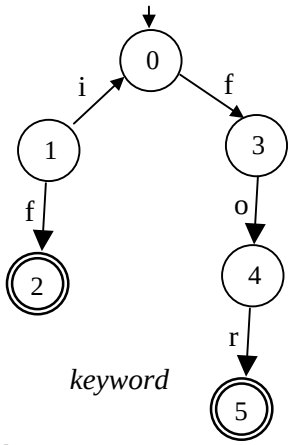
4. Definire nel linguaggio *Haskell*, mediante la notazione di pattern-matching, la funzione **`catena`**, avente in ingresso una lista **`funzioni`** di funzioni unarie che mappano un intero in un intero, ed un **`valore`** intero, la quale computa la composizione di tutte le funzioni (nella lista) applicata a **`valore`**. Formalmente, se $f_1$, $f_2$, …, $f_n$ sono le funzioni nella lista, **`catena`** computa $f_1(f_2(\ldots(f_n($**`valore`**$))\ldots))$. Se **`funzioni`** è vuota, **`catena`** restituisce **`valore`**. Ecco alcuni esempi:

| **`funzioni`** | **`valore`** | **`catena funzioni valore`** |
|---|---|---|
| `[]` | `3` | `3` |
| `[quad, cube, fact]` | `3` | `quad(cube(fact(3))) = 1296` |
| `[quad, fib, fib]` | `6` | `quad(fib(fib(6))) = 441` |
| `[fib, quad]` | `5` | `fib(quad(5)) = 75025` |

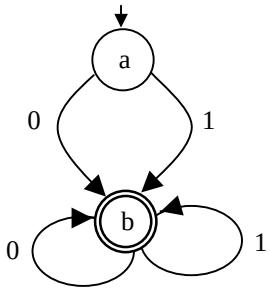
5. Si assume di avere una base di fatti *Prolog* relativa alla specifica di una serie di automi. Ogni automa è rappresentato da un nome, uno stato iniziale, un insieme di stati finali e da un insieme di transizioni. Ogni cammino dallo stato iniziale ad uno stato finale genera una parola riconosciuta dall'automa. Ecco tre automi, rispettivamente di nome *comparison*, *keyword* e *binary*:




*comparison*




*keyword*




*binary*

ed ecco la corrispondente base dei fatti *Prolog*:

```
automa(comparison, 0, [1,3], [tr(0,'=',1),tr(0,'!',2),tr(2,'=',3)]).

automa(keyword, 0, [2,5], [tr(0,i,1),tr(1,f,2),tr(0,f,3),tr(3,o,4),tr(4,r,5)]).

automa(binary, a, [b], [tr(a,0,b),tr(a,1,b),tr(b,0,b),tr(b,1,b)]).
```

Si chiede di specificare il predicato **`riconosce(A,P)`** che risulta vero qualora l'automa di nome **`A`** riconosca la parola **`P`**.

6. .