# MFI 2002/2003 - Recupero Compitino n. 3

## Cognome e Nome:

Si esibisca una $\lambda$-espressione $\mathcal{E}$ che contenga la variabile libera $y$ e renda i seguenti termini $\alpha/\beta$- equivalenti, illustrando i passaggi di riduzione che provano che la $\mathcal{E}$ è corretta (punti 5).

$$\begin{array}{lcl} t_1 & = & (\ \lambda\, x\, .\, \lambda\, y\, .\, (\ x\ y\ )\ \mathcal{E}\ ) \\ t_2 & = & \lambda\, z\, .\, (\ \mathcal{E}\ \lambda\, x\, .\, (\ \lambda\, y\, .\, (\ x\ y\ )\ \mathcal{E}\ )\ ) \end{array}$$

Si consideri la seguente grammatica $\mathcal{GF}$:

| | | |
|---|---|---|
| `Prog` | ::= | **with** `Decs` **eval** `Exp`. |
| `Dec` | ::= | `Type Id = Exp;`. |
| `Exp` | ::= | **0** \| **1** \| ... \| **9** \| `Exp Exp Exp` \| **R:**`Exp`,**G:**`Exp`,**B:**`Exp` \| $\mathbf{sum}_R$ \| $\mathbf{sum}_G$ \| $\mathbf{sum}_B$ \| `Exp==Exp`. |
| | ::= | \| `Exp>Exp` \| **if**(`Exp`) {`Exp`} **else** {`Exp`} \| $\pi_R$ \| $\pi_B$ \| $\pi_G$ \| `Exp(Exps)` \| $\lambda$ `Id : Type.Exp` \| `Exp` $\circ$ `Exp`. |
| `Exps` | ::= | `Exp` \| `Exp, Exps`. |
| `Decs` | ::= | `Dec` \| `Dec, Decs`. |
| `Type` | ::= | **color** \| `Types` $\rightarrow$ `Type`. |
| `Types` | ::= | `Type` \| `Type` $\times$ `Types`. |

Intuitivamente, questo linguaggio serve per descrivere colori (in formato RGB), funzioni su di essi (filtri) di ogni ordine e grado. I tipi che compaiono nel linguaggio (cioè che possono essere usati per dichiarare costanti) sono solo il tipo colore e quelli funzionali, ma il linguaggio permette anche di definire espressioni intere a 3 cifre e booleane a cui non si può associare un identificatore nell'ambiente, ma che servono per costruire espressioni ad esempio usando i costrutti **R:**`Exp`, **G:**`Exp`, **B:**`Exp` (che rappresenta un colore avente le 3 sottospressioni come valori per le componenti red, green e blue) e **if**(`Exp`) {`Exp`} **else** {`Exp`}.

1. (punti 2) Dare i tipi delle funzioni di semantica statica, definendo eventuali insiemi e funzioni ausiliari che si usino in essi.

2. (punti 2) Dare i tipi delle funzioni di semantica dinamica, definendo eventuali insiemi e funzioni ausiliari che si usino in essi.

3. Dare la/e regola/e della semantica statica e dinamica per il non terminale `Dec`(punti 6) in modo tale che

   - ogni identificatore venga dichiarato al più una volta;
   - se l'espressione è di tipo funzionale (e solo in questo caso), il nome della funzione può comparire nell'espressione e in tal caso corrisponde ad una chiamata ricorsiva.

4. Dare la/e regola/e della semantica statica e dinamica per il non terminale `Exps` relative alle seguenti produzioni

   `Exp` ::= `Exp Exp Exp` corrispondente alla costruzione di un intero nell'intervallo $[0, 100]$ a partire da tre cifre, in notazione decimale(punti 2)

   `Exp` ::= $\mathbf{sum}_R$ dove $\mathbf{sum}_R$ è una *funzione* che somma alla componente red di un colore un intero; il risultato è indefinito se la somma supera 100 (punti 2).

   `Exp` ::= `Exp>Exp` che è corretta solo se le espressioni coinvolte sono di tipo numerico (punti 1).

   `Exp` ::= $\lambda$ `Id` : `Type.Exp` corrispondente alla definizione di una funzione anonima (stile lambda-calcolo) con un solo parametro del tipo dichiarato che può essere una ridefinizione rispetto all'ambiente esterno e usato nel corpo della funzione (punti 4).