MFI 2002/2003 - Recupero Compitino n. 1

Cognome e Nome:

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

`Prog`	::=	**with** `Decs` **eval** `Exp`.
`Dec`	::=	**var** `Type Id= Exp;` | **const** `Type Id= Exp;`.
`Exp`	::=	**0** | **1** | ... | **9** | `Exp Exp Exp` | **R:**`Exp`,**G:**`Exp`,**B:**`Exp` | `Id=Exp` | **sum** | `Exp==Exp`.
	::=	| `Exp>Exp` | **if**(`Exp`) {`Exp`} **else** {`Exp`} | π_R | π_B | π_G | `Exp(Exps)` | λ `Id : Type.Exp` | `Exp` $\circ$ `Exps`.
`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 e variabili) 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 3) Dare i tipi delle funzioni di semantica statica, definendo eventuali insiemi e funzioni ausiliari che si usino in essi. (si faccia attenzione a mantenere distinti gli identificatori dichiarati come variabili, che possono essere riassegnati, da quelli dichiarati come costanti, che possono solo essere usati ma mai riassegnati)

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 8) 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.

   const `Type Id= Exp;`

   var `Type Id= Exp;`

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 ::=` **sum** dove **sum** è una *funzione* che somma componente a componente due colori; il risultato è indefinito se una (o più) somma supera 100 (punti 2).

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

   `Exp ::= Id=Exp` che è corretta solo se l'identificatore è una *variabile* dichiarata con lo stesso tipo dell'espressione (e il cui risultato è il valore dell'espressione + il side-effect dovuto all'assegnazione) (punti 2).

   `Exp` ::= λ `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).