Programmazione Funzionale – Giugno 2022

Nota: è indispensabile specificare il tipo e dare una descrizione dichiarativa di ogni funzione ausiliaria utilizzata (anche locale), altrimenti non verrà presa in considerazione (ad eccezione delle funzioni il cui tipo e specifica sono già dati nel testo).

1. Sia data la seguente definizione di tipo per rappresentare espressioni booleane costuite con gli operatori "not", "and" e "or":

```
type 'a expr =
    Atom of 'a
  | Not of 'a expr
  | And of 'a expr * 'a expr
  | Or of 'a expr * 'a expr
```

   Diciamo che una α `list` `L` soddisfa una α `expr` `E` se:

   - se `E=Atom x`: `x` è un elemento di L;
   - se `E=Not` $\texttt{E}_1$: `L` non soddisfa $\texttt{E}_1$;
   - se `E=And(`$\texttt{E}_1$`,`$\texttt{E}_2$`)`: `L` soddisfa $\texttt{E}_1$ e `L` soddisfa $\texttt{E}_2$;
   - se `E=Or(`$\texttt{E}_1$`,`$\texttt{E}_2$`)`: `L` soddisfa $\texttt{E}_1$ oppure `L` soddisfa $\texttt{E}_2$.

   Definire una funzione `eval:` α `list` $\rightarrow$ α `expr` $\rightarrow$ `bool`, tale che `eval list expr = true` se `list` soddisfa `expr`, altrimenti `eval list expr = false`.

   Ad esempio, si avrà `eval [1;2;3] (And(Atom 3,Not(Atom 5))) = true` perché `eval [1;2;3] (Atom 3) = true` – dato che 3 appartiene alla lista – e `eval [1;2;3] (Atom 5) = false`.

2. Data una lista associativa `listassoc:` $(\alpha \times \beta$ `list) list`, la lista associata da `listassoc` a un elemento `x` di tipo α è l'elemento associato a `x` in `listassoc`, se un tale elemento è definito, altrimenti è la lista vuota.

   Definire una funzione `list_of:` $\alpha \rightarrow (\alpha \times \beta$ `list) list` $\rightarrow \beta$ `list`, tale che `list_of x listassoc` riporti la lista associata a `x` da `listassoc` (la funzione non deve mai sollevare eccezioni).

   Ad esempio, si avrà:
   `list_of "b" [("a",[1;2;3]);("b",[4;5]);("c",[6;7;8]);("b",[9;10])] = [4; 5]`, e `list_of "z" [("a",[1;2;3]);("b",[4;5]);("c",[6;7;8])] = []`.

3. Definire un tipo di dati α `graph` per la rappresentazione di grafi orientati mediante liste di archi e definire, adattando opportunamente l'implementazione della visita in profondità di un grafo, una funzione `good_nodes` di tipo

   $$\alpha \texttt{ graph} \rightarrow (\alpha \times \beta \texttt{ list}) \texttt{ list} \rightarrow \beta \texttt{ expr} \rightarrow \alpha \rightarrow \alpha \texttt{ list}$$

   tale che `good_nodes graph listassoc expr start` riporti la lista dei nodi `x` raggiungibili da `start` (nel grafo `graph`), tali che la lista associata da `listassoc` a `x` soddisfi l'espressione `expr`.