Progetto di Linguaggi e Compilatori – Parte 3
A.A. 2019-20

Gruppo 2

Attenzione: una volta letto il presente testo è *obbligatorio* consegnare alla scadenza quanto elaborato, indipendentemente dal fatto che lo si consideri adeguato o meno.

Entro il termine stabilito si *deve* inviare via email dentro un unico file compresso, il cui nome sia "`ProgettoLC parte3 Gruppo 2`", tutti (e soli) i sorgenti sviluppati, in modo che sia possibile verificare il funzionamento di quanto realizzato. In tale file compresso *non* devono comparire file oggetto e/o binari o qualsiasi altro file che viene generato a partire dai sorgenti. Sempre in detto file va inoltre inclusa una relazione in formato PDF dal nome "`ProgettoLC parte3 Gruppo 2 Relazione`". La relazione può contenere immagini passate a scanner di grafici o figure fatte a mano.

Si richiede:

- una descrizione dettagliata di tutte le tecniche **non**-standard impiegate (le tecniche standard imparate a lezione non vanno descritte).
- una descrizione delle assunzioni fatte riguardo alla specifica, sia relativamente a scelte non previste espressamente dalla specifica stessa, che a scelte in contrasto a quanto previsto (con relative motivazioni).

  Non è assolutamente utile perdere tempo per includere nella relazione il testo dei vari esercizi o un suo riassunto o una qualsiasi rielaborazione, incluso descrizioni del problema da risolvere. Ci si deve concentrare solo sulla descrizione della soluzione e delle eventuali variazioni rispetto a quanto richiesto.
- una descrizione sintetica generale della soluzione realizzata.
- di commentare (molto) sinteticamente ma adeguatamente il codice prodotto.
- di fornire opportuno Makefile (compliant rispetto allo standard GNU make) per
  - poter generare automaticamente dai sorgenti i files necessari all'esecuzione (lanciando `make`);
  - far automaticamente una demo (lanciando `make demo`).
- di implementare la soluzione in Haskell, utilizzando BNFC o Happy/Alex.

Esercizio

Si consideri una rappresentazione in sintassi concreta per alberi pesati con un numero arbitrario di figli dove un albero non vuoto viene descritto con il valore in radice e, se non è un nodo foglia, immediatamente dopo abbiamo la sequenza dei figli (separata da "blanks" quando serve). Per distinguere sotto-alberi non foglia si usino le parentesi. In ogni nodo (della sintassi astratta) si deve mantenere, oltre al campo valore, l'altezza del nodo stesso.

(a) Per detti alberi, si progetti un opportuna sintassi astratta *polimorfa* e si realizzino due parsers (con relativo lexer) per alberi (pesati) di numeri interi e alberi (pesati) di numeri in virgola mobile, che utilizzino entrambi detta sintassi astratta.

(b) Considerando il caso in cui $\mathbf{T} = \{\texttt{a}, \texttt{(}, \texttt{)}\}$ e detta G la grammatica implementata nel parser, si determini $\{w \in \mathbf{T}^* \mid w = w^R\} \setminus \mathcal{L}(G)$ (w^R indica la stringa w rovesciata).

(c) Rappresentando alberi con la sintassi astratta proposta, si scriva una funzione `maximalWeight` che, preso un albero di valori numerici positivi, determina il massimo dei pesi di tutti i cammini, indipendentemente dall'orientamento degli archi. Il peso di un cammino, è la somma dei valori dei nodi del cammino.

(d) Si combini quindi il parser per numeri interi con `maximalWeight` in una funzione `test` per cui preparare dei test case significativi.