Progetto di Linguaggi e Compilatori – Parte 1
A.A. 2021-22

Gruppo 21-3

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

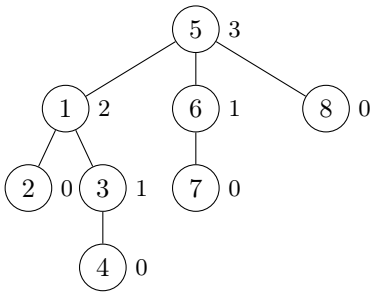
Entro il termine stabilito si *deve* inviare via email dentro un unico file compresso, il cui nome sia "`ProgettoLC parte1 Gruppo 3`", tutti (e soli) i sorgenti sviluppati, in modo che sia possibile verificare il funzionamento di quanto realizzato. In tale file compresso *non* devono comparire file oggetto 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 parte1 Gruppo 3 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).
- Si richiede 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.
- Si richiede una descrizione sintetica generale della soluzione realizzata.
- Si richiede di commentare (molto) sinteticamente ma adeguatamente il codice prodotto.
- Si richiede 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`).
- La soluzione *deve* essere implementata in Haskell, utilizzando Happy/Alex. Si suggerisce di utilizzare BNFC per costruire un primo prototipo iniziale da raffinare poi manualmente, ma non è obbligatorio (si includa anche il sorgente .cf in caso).

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 delimitata con `{` e `}`. Ad esempio, per un albero di numeri interi, un input valido è `5{ 1 {2 3{ 4}}  6 {7 }8}`. che corrisponde all'albero



In ogni nodo (della sintassi astratta) si deve mantenere, oltre al campo valore, quale peso del nodo l'altezza del nodo stesso. Si raccomanda di non confondere l'altezza di un nodo con la sua profondità!

(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 $\mathbf{T} = \{\mathtt{a}, \mathtt{b}, \mathtt{\{}, \mathtt{\}}\}$ e detta G la grammatica implementata nel parser adattata all'alfabeto $\mathbf{T}$, si determini $\mathcal{L}(G) \setminus \{w \in \mathbf{T}^* \mid w = w^R\}$ (w^R indica la stringa w rovesciata).

(c) Rappresentando alberi con la sintassi astratta proposta, si scriva un predicato `isAlmostBalanced` che, preso un albero, determina se ha la seguente proprietà: per ogni nodo le altezze di tutti i figli differiscono al massimo di 1.

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

Attenzione: La soluzione deve fare uso solo degli strumenti introdotti nella parte del corso a cui si riferisce l'esercizio.