Progetto di Linguaggi e Compilatori 1 – Parte 1
A.A. 2015/16

Gruppo 14

Marco Bucchiarone
Emanuele Tonetti
Francesco Zilli

Esercizio 1

La funzione Haskell `boundedMaximum(n [BST t])` data una lista di BST t trova, se esiste

$$max_n t = max(\forall x \in t \mid x < n).$$

La soluzione all'esercizio è implementata nel modulo BoundMax (`BoundMax.hs`) che viene importato nel `Main`. Nel `Main` la funzione viene chiamata fornendole gli argomenti inseriti a tastiera a runtime (usando le funzioni importate tramite `System.Environment`).

La `boundedMaximum` viene costruita sulla `foldl1`: `MaxNT` individua il massimo tra i minoranti nella lista costruita dalla funzione di visita del BST `getmin` che sfrutta la struttura dati considerata. Il modulo `BoundMax` è testabile nell'interprete di GHC tramite i test case forniti nel file `queryEs1.txt`, il quale comprende anche una rappresentazione grafica di alcuni degli alberi usati per una maggiore leggibilità.

Esercizio 2

Parte A: Dalla sintassi concreta per alberi pesati con numero arbitrario di figli fornita è stata costruita la sintassi astratta *polimorfa*. L'implementazione proposta è suddivisa tra i seguenti file, di cui si da una breve descrizione:

Data.hs contiene la definizione dei tipi di dato per la sintassi astratta polimorfa e per i token,

Lexer.x produce token per i soli elementi dell'alfabeto usati dalla sintassi concreta comprese le rappresentazioni dei dati di tipo `Int` e `Double` ignorando elementi estranei all'alfabeto,

ParserI.y produce di alberi pesati di numeri soli `Int`,

ParserD.y produce di alberi pesati di numeri `Double`.

In entrambi i parser è fornita, nella sezione di codice opzionale, la funzione `detWeight` che calcola il peso dei nodi individuali; il peso viene definito ricorsivamente come la distanza massima di una foglia dal nodo considerato, incrementato di 1 (ad ogni foglia viene assegnato peso pari a 0). Nel caso del parser per alberi (pesati) in virgola mobile si è assunto i `Double` siano rappresentabili da sequenze di cifre da 0 a 9 senza punto decimale. Dalla formulazione della consegna si è assunto la grammatica G, implementata nel parser, non abbia produzioni di tipo ϵ.

Parte B: Considerando l'alfabeto $\Sigma = \{\texttt{[}, \texttt{]}, \texttt{,}, \texttt{a}, \texttt{b}\}$, l'insieme

$$\mathcal{P} = \{w \in \Sigma^* \mid w = w^R\}$$

è l'insieme delle stringhe di elementi dell'alfabeto palindrome. Siccome il linguaggio $\mathcal{L}(G)$ generato dalla grammatica ammette come stringhe palindrome solo gli alberi di altezza zero rappresentati dal singolo nodo "`a`" e "`b`", l'insieme risultante da

$$\mathcal{P} \setminus \mathcal{L}(G)$$

corrisponde all'insieme ottenuto facendo

$$\mathcal{P} \setminus \{x \in \Sigma^* \mid x = a \vee x = b\}.$$

Dimostrazione:

$$\mathcal{P} = \epsilon \cup \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4 \cup \mathcal{L}_5$$

dove

$$\mathcal{L}_1 = \{x \in \mathcal{P} \mid x \textit{ inizia con } [\ \}$$
$$\mathcal{L}_2 = \{x \in \mathcal{P} \mid x \textit{ inizia con }]\ \}$$
$$\mathcal{L}_3 = \{x \in \mathcal{P} \mid x \textit{ inizia con } ,\ \}$$
$$\mathcal{L}_4 = \{x \in \mathcal{P} \mid x \textit{ inizia con } a\ \}$$
$$\mathcal{L}_5 = \{x \in \mathcal{P} \mid x \textit{ inizia con } b\ \}$$

quindi

$$\mathcal{P} \setminus \mathcal{L}(G) = (\epsilon \cup \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4 \cup \mathcal{L}_5) \setminus \mathcal{L}(G)$$

equivalente a

$$(\epsilon \cup \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4 \cup \mathcal{L}_5) - ((\epsilon \cup \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4 \cup \mathcal{L}_5) \cap \mathcal{L}(G))$$

che con semplici passaggi si vede essere uguale ad

$$(\epsilon \cup \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4 \cup \mathcal{L}_5) - ((\epsilon \cup \emptyset \cup \emptyset \cup \emptyset \cup \emptyset \cup a \cap b)$$

ovvero

$$\mathcal{P} \setminus \{x \in \Sigma^* \mid x = a \vee x = b\}.$$

Parte C: La funzione `isSymm` si basa sull'idea che una foglia è sempre simmetrica a se stessa, mentre un nodo lo è solamente se equivale al suo albero trasposto. Per questo calcolo vengono utilizzate due sotto-funzioni:

isEq: stabilisce l'equivalenza strutturale di due alberi;

trasp: calcola la trasposta di un albero in maniera ricorsiva (invertendo i figli di ogni nodo);

Tutto il codice relativo è fornito nel file `Symm.hs`.

Parte D: La funzione isSymm è combinata al parser per interi nel file `TestSymm.hs` che ne importa i rispettivi moduli. Eseguendo il file compilato, con relativi parametri, vengono stampati a video l'albero, il rispettivo albero trasposto ed il risultato della funzione isSymm che indicherà se l'albero è effettivamente simmetrico. È possibile fornire in input il file `examples.txt` per effettuare svariati test case (eseguire l'automatizzazione di questo processo con il comando textttmake demo per veloce riscontro).