ProgettoLC parte1 Gruppo 3 Relazione

143162 Gianluca D'Abrosca
142222 Riccardo Lunni
143095 Francesco Zuccato

21 maggio 2022

a) Per l'implementazione di lexer e parser era possibile prima abbozzarli con BNFC e poi, necessariamente, realizzarli in Alex e Happy. Si è optato per procedere direttamente alla scrittura in Alex e Happy.

Alex In Alex si sono definiti i seguenti token:

- *TokenStartChildren*, per identificare le paresentesi graffe aperte;
- *TokenEndChildren*, per identificare le parentesi graffe chiuse;
- *TokenInt*, per individuare i valori interi;
- *TokenDouble*, per individuare valori in virgola mobile.

Gli spazi bianchi, come richiesto dalla consegna, sono stati ignorati, indifferentemente dal tipo e dal numero di occorrenze consecutive.

Happy La grammatica utilizzata dai parser in Happy è la seguente:

$$\begin{aligned} RadiceNum &\rightarrow num \mid num\ \{\ FigliNum\ \} \\ FigliNum &\rightarrow RadiceNum \mid RadiceNum\ FigliNum \end{aligned}$$

La stessa grammatica è stata definita per entrambi i parser. La differenza tra le due sta nel tipo di token che viene usato per identificare i num: *int* per gli alberi di interi, *double* per gli alberi di numeri in virgola mobile. Si noti che la grammatica non accetta l'albero vuoto, deve essere presente almeno un nodo radice.

b) Al secondo punto è richiesto di calcolare $(L(G) \setminus L(P))$, con $L(P) = \{\, w \in T \mid w = w^R \,\}$. Ovvero, la differenza insiemistica tra il linguaggio generato dalla grammatica appena presentata meno l'insieme delle stringhe palindrome, dato l'alfabeto $\{a, b, \{, \}\}$.
Prima di procedere con la soluzione è sorta un'ambiguità sull'interpretazione di a e b: sono essi da considerare dei numeri, il che implicherebbe che sarebbero gli unici due numeri presenti in $L(P)$ ed $L(G)$, oppure delle cifre (ad esempio 0 e 1), che consentirebbero di generare tutti i numeri naturali in base 2.
Si è deciso di considerare entrambe le possibilità.

Considerazioni comuni Per ottenere la differenza insiemistica è utile prima calcolarsi l'intersezione tra i due insiemi. Si nota banalmente che tutti gli alberi con almeno un nodo figlio (ad es. $a\{b\}$) non potranno mai essere una stringa palindroma in quanto iniziano con un a o b e terminano con $\}$.
Resta da analizzare il caso base degli alberi-foglia, che però nel caso 1 corrispondono ad a o b, mentre nel caso 2 all'insieme delle stringhe $(a, b)^*$.

1) a,b come numeri Il caso 1 è banale, gli unici due alberi-foglia possibili sono proprio a e b, che sono palindromi. Quindi in questo caso sono esclusi tutti gli alberi-foglia da $L(G)$.

2) a,b come cifre Se consideriamo a e b essere dei digit, produrranno un'intersezione non vuota solo le parole palindrome (ad es. $aa, aba, b, aabbaa$ ecc.).
Quindi si ha che da L(G) non saranno rimossi tutti gli alberi-foglia, ma solo quelli il cui valore è formato da stringhe palindrome.

c) Nel file *demo.hs* è stata implementato il predicato *isAlmostBalanced*, che ritorna $True$ se la distanza massima tra le altezze dei figli di ciascun nodo è 1.
L'algoritmo è il seguente: per ogni nodo x si ottiene l'altezza minima tra quella dei nodi figli, $min(h(sons(x)))$, tramite *minHeight*, e la si confronta con $h(x)$, l'altezza del nodo x. La consegna stabilisce che un albero è quasi bilanciato se "per ogni nodo le altezze di tutti i figli differiscono al massimo di 1". Considerando che $h(x) = 1 + max(h(sons(x)))$, ovvero l'altezza di x sarà pari ad uno più l'altezza massima dei suoi figli, è sufficiente verificare che $\forall x : h(x) - min(h(sons(x))) < 3$. Una volta effettuato questo controllo sul nodo radice, lo si mette in *and* con il resto dell'albero, procedendo ricorsivamente per tutti i suoi figli fino al caso base delle foglie, che sono bilanciati per definizione non avendo figli.

d) In *demo.hs* sono state create due funzioni per testare il tutto, una per gli interi, `test_int` e una per i numeri in virgola mobile, `test_double`. Sono stati testati vari casi sia con alberi bilanciati che non. La stringa vuota non è stata inclusa tra i test in quanto il caso dell'albero vuoto si è deciso di non accettarlo nella grammatica. Per ognuno degli alberi si sono applicati prima il lexer in `Alex`, poi il parser in `Happy` ed infine il predicato *isAlmostBalanced*.
Gli alberi di test sono stati generati sia come interi che come numeri in virgola mobile.
Per eseguire i test sarà sufficiente prima compilare con il comando `make`, che genererà i due file Haskell *Alex.hs* e *Happy.hs* ed il binario *demo* nella cartella `__build` e poi eseguire i test con `make demo`.

È possibile inoltre eseguire dei propri test con alberi custom tramite `ghci`, o specificato il path dei file Haskell con il comando `ghci demo.hs __build/Alex.hs __build/Happy.hs` oppure includendo `__build` al path della compilazione con `ghci -i__build demo.hs`.
Per testare gli alberi sarà infine sufficiente chiamare la funzione `test_int` (o `test_double`) seguita dalla lista di alberi da testare. Ad esempio: `test_int [ "1", "2 {3}" ]`.