Principles and Paradigms of Programming Languages

a.a. 2017/18

Esercizi Haskell (1)

1 novembre 2017

Mini-guida all'interprete interattivo L'interprete interattivo si chiama `ghci`; sotto Windows, suggeriamo di usare `WinGHCi`. Invocando l'interprete si ottiene un prompt e si può scrivere del codice Haskell da valutare. Per esempio:

```
3*2
```

o

```
(\x -> x+1) 41
```

Con `:?` potete ottenere la lista dei comandi ghci disponibili, i più utili sono:

- `:l` per caricare un file (con `WinGHCI` è anche possibile aprire direttamente con un doppio clic un file con estensione `hs`, oppure scegliere "Load..." dal menu File di WinGHCi)
- `:r` per ricaricare il file corrente
- `:t` per vedere il tipo di un'espressione
- `:set +t` per ottenere anche il tipo oltre alla valutazione (la variabile speciale `it` conserva il valore dell'ultima valutazione)

Avvertenze generali Nel seguito sono proposti molti esercizi, potete farli tutti o saltarne alcuni a seconda della vostra competenza (precedente a questo corso) sul paradigma funzionale. In alcuni casi abbiamo utilizzato nomi diversi per funzioni che sono predefinite per evitare conflitti.

Esercizi per principianti Definire le seguenti funzioni e poi valutare le espressioni date controllando il risultato e il suo tipo. Ricordiamo che *non* è possibile visualizzare una funzione, ma è possibile visualizzarne il tipo.

la funzione identità `myid`, `myid 1`, `myid True`
la funzione `prod` che moltiplica due interi, `prod 3 4`
la funzione `twice` che raddoppia un intero, `twice 3`
il predicato `isEven` che controlla se un intero è pari, `isEven 3`
la composizione di due funzioni `compose`, `compose isEven id 2`, `compose isEven id 3`, `compose isEven id`
`mysum g n` = la somma da 0 a `n` di `g(i)`
`sumsquare` = la somma da 0 a `n` di `i*i` come *istanza* (ossia, ottenuta per applicazione parziale) della funzione precedente
`forloop n body s` = esegue `n` volte `body` a partire da `s`, `forloop 2 (\x -> x+1) 5`
(risultato: 7)
la funzione `leq` che, date due funzioni `f` and `g` da interi a interi, controlla se `f<=g` per gli interi da `n` a `m`,
`leq id twice 1 10`

Esercizi su liste

`prodlist` = il prodotto di una lista di interi
`prodlist` come istanza di `foldl` (l'iteratore visto a lezione)
`member x xs` = controlla se `x` è presente nella lista `xs`
`member` come istanza di `foldl`
`leq` utilizzando la list comprehension e la funzione predefinita `and` (cercare la definizione)

`copy n x` = la lista formata da `n` copie di `x`
`copy` come istanza of `forloop`
`copy` usando la list comprehension
`mytake n xs` = restituisce i primi `n` elementi della lista `xs`
`mydrop n xs` = rimuove i primi `n` elementi della lista `xs`
`poslist xs` = gli elementi positivi di `xs` (come istanza di `filter`)
`forall p xs` = controlla se il predicato `p` vale per tutti gli elementi della lista `xs`
`allpos xs` = controlla se tutti gli elementi di una lista di interi sono positivi (come istanza della funzione precedente)
`split p xs` = restituisce due liste, formate rispettivamente dagli elementi di `xs` su cui vale `p` e da quelli su cui non vale
definire un'istanza di `map` che, data una lista di coppie, controlla se sono tutte formate da elementi uguali
`exists p xs` = controlla se almeno un elemento di `xs` soddisfa `p`
`foldright` = iteratore analogo a `foldl`, ma che inizia l'iterazione dall'ultimo elemento
`composelist` restituisce la composizione di una lista di funzioni (come istanza di `foldright`)
`insert x xs` inserisce al posto giusto `x` in una lista supposta ordinata
`insertsort xs` restituisce la versione ordinata di `xs` inserendo via via gli elementi di `xs` al posto giusto (come istanza di `foldl`)