ProgettoLC parteC Riccardo Cavasin Relazione

Esercizio 1

De-sugaring

Globalizzo `f` e `aggr` esponendo come parametro la variabile catturata `v`, riscrivo l'applicazione parziale dell'operatore infisso `+`, e annoto i tipi delle funzioni inferibili.

```
    g :: Int -> Int -> [Char]
R1  g k n = take n (v:v:foldr (aggr k) [] (enumFromThen 'a' 'c'))
      where
        v = f k '0'

    f :: Int -> Char -> Char
R2  f k x = toEnum ((\y -> y+k) (fromEnum x))

    aggr :: Int -> Char -> [Char]
R3  aggr k x xs = f k x:xs

    take :: Int -> [a] -> [a]
R4  take n _ | n <= 0 = []
R5  take n (x:xs) = x:take (n-1) xs
R6  take _ [] = []

    foldr :: (a -> b -> b) -> b -> [a] -> b
R7  foldr f z [] = z
R8  foldr f z (x:xs) = f x (foldr f z xs)
```

`k :: Int` poiché `fromEnum :: Enum a => a -> Int`.

Esecuzione

Valutazione completa, ordinata della query:

```
let n=4-1 in g (7-n) n
```

1. Si procede con la valutazione di:

   ```
   g (-@Int 7@Int $a[-@Int 4@Int 1@Int]) $a
   ```

2. *R1* Si nota che a differenza dei letterali numerici, i caratteri letterali non sono di tipo polimorfo.

   ```
   take $a[-@Int 4@Int 1@Int] ($c[f $b[-@Int 7@Int $a] '0']:$c:foldr (aggr $b) []
   (enumFromThen@Char 'a' 'c'))
   ```

3. *R4* match: si valuta la guardia:

```
<=@Int $a[-@Int 4@Int 1@Int] 0@Int
```

   - ```
     <=@Int 3@Int 0@Int
     ```
   - `False`

4. riscrittura `$a` ; *R5* match

```
$c[f $b[-@Int 7@Int 3@Int] '0']:take (-@Int 3@Int 1@Int) ($c:foldr (aggr $b) [] (enumFromThen@Char 'a' 'c'))
```

5. *R2*

   - ```
     $c[toEnum@Char ((\y -> +@Int y $b[-@Int 7@Int 3@Int]) (fromEnum@Char '0'))]:take (-@Int 3@Int 1@Int) ($c:foldr (aggr $b) [] (enumFromThen@Char 'a' 'c'))
     ```
   - ```
     $c[toEnum@Char ((\y -> +@Int y $b[-@Int 7@Int 3@Int]) 48@Int)]:take (-@Int 3@Int 1@Int) ($c:foldr (aggr $b) [] (enumFromThen@Char 'a' 'c'))
     ```
   - ```
     $c[toEnum@Char (+@Int 48@Int $b[-@Int 7@Int 3@Int])]:take (-@Int 3@Int 1@Int) ($c:foldr (aggr $b) [] (enumFromThen@Char 'a' 'c'))
     ```
   - riscrittura `$b`

     ```
     $c[toEnum@Char (+@Int 48@Int 4@Int)]:take (-@Int 3@Int 1@Int) ($c:foldr (aggr 4@Int) [] (enumFromThen@Char 'a' 'c'))
     ```
   - ```
     $c[toEnum@Char 52@Int]:take (-@Int 3@Int 1@Int) ($c:foldr (aggr 4@Int) [] (enumFromThen@Char 'a' 'c'))
     ```
   - riscrittura `$c`

     ```
     '4':take (-@Int 3@Int 1@Int) ('4':foldr (aggr 4@Int) [] (enumFromThen@Char 'a' 'c'))
     ```

6. *R4* match: si valuta la guardia:

```
<=@Int $g[(-@Int 3@Int 1@Int)] 0@Int
```

   - ```
     <=@Int 2@Int 0@Int
     ```
   - `False`

7. riscrittura `$g` ; *R5* match

```
'4':'4':take (-@Int 2@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'a' 'c'))
```

8. *R4* match: si valuta la guardia:

```
<=@Int $g[-@Int 2@Int 1@Int] 0@Int
```

   - ```
     <=@Int 1@Int 0@Int
     ```
   - `False`

9. riscrittura `$g` ; valutazione per pattern *R5* di `foldr (aggr 4@Int) [] (enumFromThen@Char 'a' 'c')` ; valutazione per pattern *R7* di `enumFromThen@Char 'a' 'c'`

```
'4':'4':take 1@Int (foldr (aggr 4@Int) [] ('a':enumFromThen@Char 'c' 'e'))
```

10. *R7* mismatch
11. *R8* match

```
'4':'4':take 1@Int ((aggr 4@Int) 'a' (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e')))
```

12. *R3* Si nota che la funzione di aggregazione permette una valutazione lazy di `foldr`.

```
'4':'4':take 1@Int (f 4@Int 'a':(foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e')))
```

13. *R5* match

```
'4':'4':f 4@Int 'a':take (-@Int 1@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e'))
```

14. *R2*

    - ```
      '4':'4':toEnum@Char ((\y -> +@Int y 4@Int) (fromEnum@Char 'a')):take (-@Int 1@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e'))
      ```
    - ```
      '4':'4':toEnum@Char ((\y -> +@Int y 4@Int) 97@Int):take (-@Int 1@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e'))
      ```
    - ```
      '4':'4':toEnum@Char (+@Int 97@Int 4@Int):take (-@Int 1@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e'))
      ```
    - ```
      '4':'4':toEnum@Char 101@Int:take (-@Int 1@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e'))
      ```
    - ```
      '4':'4':'e':take (-@Int 1@Int 1@Int) (foldr (aggr 4@Int) [] (enumFromThen@Char 'c' 'e'))
      ```

15. *R4* match: si valuta la guardia:

```
<=@Int (-@Int 1@Int 1@Int) 0@Int
```

    - ```
      <=@Int 0@Int 0@Int
      ```
    - `True`

```
'4':'4':'e':[]
```

Il risultato della query è l'array di `Char` (stringa):

```
['4','4','e']
```
```

Esercizio 2

Sono riportate solo le funzioni con più di un caso (pattern).

Parte C

- `take`
  i. `n1 _`, `n2 (x:xs)` overlap: `n1 (x:xs)`
  ii. `n _`, `_ []` overlap: `n []`
  iii. `n (x:xs)`, `_ []` no overlap
- `foldr`
  i. `f z []`, `f z (x:xs)` no overlap

Parte A

- `compress`
  i. `C a`, `Q a b c d` no overlap
  - `case`
    a. `Q (C a) (C b) (C c) (C d)`, `_` overlap: `Q (C a) (C b) (C c) (C d)`
- `f`
  i. `_ (C False)`, `(C (r, g, b)) (C True)` no overlap
  ii. `_ (C False)`, `(Q p1 p2 p3 p4) (C True)` no overlap
  iii. `_ (C False)`, `(C {}) (Q m1 m2 m3 m4)` no overlap
  iv. `_ (C False)`, `(Q p1 p2 p3 p4) (Q m1 m2 m3 m4)` no overlap
  v. `(C (r, g, b)) (C True)`, `(Q p1 p2 p3 p4) (C True)` no overlap
  vi. `(C (r, g, b)) (C True)`, `(C {}) (Q m1 m2 m3 m4)` no overlap
  vii. `(C (r, g, b)) (C True)`, `(Q p1 p2 p3 p4) (Q m1 m2 m3 m4)` no overlap
  viii. `(Q p1 p2 p3 p4) (C True)`, `(C {}) (Q m1 m2 m3 m4)` no overlap
  ix. `(Q p1 p2 p3 p4) (C True)`, `(Q p1 p2 p3 p4) (Q m1 m2 m3 m4)` no overlap
  x. `(C {}) (Q m1 m2 m3 m4)`, `(Q p1 p2 p3 p4) (Q m1 m2 m3 m4)` no overlap
  - `case`
    a. `(_, C False)`, `_` overlap: `(x, C False)`