Haskell Exercises 4: ZF-expressions

Antoni Diller

26 July 2011

(1) Define a function \( \text{unique} :: [\text{Int}] \rightarrow [\text{Int}] \) which returns the list of numbers in a list that occur exactly once in that list. For example, \( \text{unique} [2, 4, 2, 1, 4] = [1] \).

(2) Redefine the function \( \text{member} \), explained in question (2.10), using a list comprehension (that is to say, a ZF-expression).

(3) Consider the list of numbers \( \text{wirth} \) defined as follows:
   
   (i) 1 is a member of \( \text{wirth} \).
   (ii) If \( x \) is a member of \( \text{wirth} \), then so are \( 2 \times x + 1 \) and \( 3 \times x + 1 \).
   (iii) These are the only elements of \( \text{wirth} \).

   Give a definition of \( \text{wirth} \) which produces the elements in increasing order.

(4) Define the function \( \text{sumsq} \), which takes an integer \( n \) as its argument and returns the sum of the squares of the first \( n \) integers, using a ZF-expression.

(5) Using a ZF-expression define a Boolean-valued function \( \text{prime} \) such that \( \text{prime} i = \text{True} \) if and only if \( i \) is a prime number.

(6) Using a ZF-expression define the function \( \text{factors} \) such that \( \text{factors} n \) is the list of all the factors of \( n \) in increasing order. For example, \( \text{factors} 6 = [1, 2, 3, 6] \).

(7) A number \( n \) is said to be perfect if the factors of \( n \), including 1 but excluding \( n \), add up to \( n \). For example, 6 is perfect because \( 6 = 1 + 2 + 3 \). Using the function \( \text{factors} \) from question (6) define the list of all perfect numbers.

(8) Define a function \( \text{perms} \) which takes a finite list \( xs \) as its argument and returns the list of all the permutations of \( xs \).

(9) Define a function \( \text{isSquare} \) which tests to see if a positive number is equal to the square of some integer. For example,

\[
\text{isSquare} \ 7 = \text{False}, \\
\text{isSquare} \ 16 = \text{True}.
\]
(10) Using a single list-comprehension define the *gcd* function. Call your function *gcdlist*. The *gcd* function can be defined like this:

\[
gcd(i, j) = \begin{cases} 
i, & \text{if } j = 0 \\ 
gcd(j, i \mod j), & \text{if } j \neq 0 \end{cases}
\]

(11) Define the function *power* using a list-comprehension, where

\[
\text{power } x \ n = x^n.
\]

Here, *x* can be any number, but *n* must be an integer (either positive, negative or zero).

(12) The function *iteraten* is defined as follows:

\[
\text{iteraten} :: \text{Int} \to (\text{a} \to \text{a}) \to \text{a} \to \text{a}
\]

\[
\text{iteraten } 0 \ f \ x = x \\
\text{iteraten } n \ f \ x = \text{iteraten } (n - 1) \ f \ (f \ x)
\]

Define a function *limit* such that *limit* \( f \ x = \text{iteraten } n \ f \ x \) where \( n \) is the least number such that \( \text{iteraten } n \ f \ x = \text{iteraten } (n + 1) \ f \ x \)