Programming Paradigms 2025
Session 13: Lazy evaluation

Problems for solving and discussing

Hans Hüttel

2 December 2025

Problems that we will definitely talk about

1. ***(15 minutes)*** We can define the following:

   ```
   x = 1 : (map (1+) x)
   ```

   and then evaluate take 5 x.

   One might think that in fact the following happens:

   ```
   take 5 x
   = take 5 (1:2:map (+1) x)
   = take 5 (1:2:map (+1) [1, 2])
   = take 5 (1:2:2:3:map (+1) x)
   = take 5 (1:2:2:3:map (+1) [1, 2, 2, 3])
   = take 5 (1:2:2:3:2:3:3:4:map (+1) x)
   ...
   ```

   Explain precisely why this is wrong. Saying that ”That is because the Haskell interpreter gives a different result” is not a valid answer – you have to provide an evaluation sequence as the ones presented in the text for today.

2. ***(20 minutes)***

   A long time ago we saw the function

   ```
   fib 1 = 1
   fib 2 = 1
   fib n = fib (n-1) + fib (n-2)
   ```

   and discovered that computing fib 50 was not easy. Why was that?

   Now define a function fibsfrom such that fibsfrom n1 n2 computes the infinite list of Fibonacci numbers starting with n1 and n2. Then try to compute fib 50. What happens – and why?

3. ***(15 minutes)***

   In Haskell, the value undefined is polymorphic – it has type a for every type a. One can put it anywhere in an otherwise well-typed expression and the result is well-typed. But if one tries to evaluate the expression, the Haskell interpreter throws the exception ”undefined”.

   Here is a function called indflet .

   ```
   indflet _ []         = []
   indflet _ [x]        = [x]
   indflet e (x:y:ys) = x : e : indflet e (y:ys)
   ```

   First try to figure out *without asking the Haskell interpreter* what the type of indflet is and what the function does. Next try to figure out *without asking the Haskell interpreter* why an exception is throwh when you evaluate

   ```
   head (indflet 1 (2:undefined))
   ```

4. ***(25 minutes)***

   Define a function allBinaries :: [String] that wil give us the infinite ordered list of strings that correspond to binary numbers, with the least significant bit *first*, no trailing zeros, i.e.

   ```
   allBinaries = ["0","1","01","11","001" ,...].
   ```

   Please ***do not attempt to do this*** *by trying to convert integers to binary strings.* Instead, generate the strings directly and find out how to leave out the invalid strings from the infinite list.

More problems to solve at your own pace

a) The function zipWith in the prelude has type zipWith :: (a −> b −> c)−> [a] −> [b] −> [c] and applies its first argument in a pairwise fashion to the elements of lists given as second and third arguments,

```
zipWith (+) [1,2,3] [1000,2000,3000]
```

gives us the list [1001,2002,3003].

Define the infinite list fibonacci of Fibonacci numbers using the zipWith function.

b) Define a version of the function from problem 3 that is called fletind and does not throw an exception when you evaluate

```
head (fletind 1 (2:undefined))
```

c) A problem, due to the mathematician W. R. Hamming, is to write a program that produces an infinite list of natural numbers with the following properties:

i The list is in ascending order, without duplicates.

ii The list begins with the number 1.

iii If the list contains the number x, then it also contains the numbers $2x$, $3x$, and $5x$.

iv iv The list contains no other numbers.

Define a function hamming that will give us such a list.