

## Solutions for Exercise Sheet 4

For these exercises, we use an applied lambda calculus with booleans, integers and cons-pairs as primitives. The  $\delta$ -conversion rules for the cons-pairs are the following:

$$\begin{aligned} \text{null nil} &= \text{true} \\ \text{null (cons } x \ y) &= \text{false} \\ \text{car (cons } x \ y) &= x \\ \text{cdr (cons } x \ y) &= y \end{aligned}$$

We will utilise the normal infix notation for arithmetic operators, e.g.  $n + (m * 2)$ , instead of the official notation  $+ n (* m 2)$ .

### Exercise 1: Recursive functions

a. Consider the function:

$$N = \lambda f. \lambda n. \text{cons } n \ (f \ (n + 1))$$

Carry out a few beta reductions for the term  $\mathbf{Y} N 1$  and describe its computational effect.

$$\begin{aligned} \mathbf{Y} N 1 &\longrightarrow \underline{N (\mathbf{Y} N) 1} && \text{by the "DNA" rule} \\ &\longrightarrow \text{cons } 1 \ (\mathbf{Y} N \ (1 + 1)) && \text{copying the body of } N \text{ and substituting parameters} \\ &\longrightarrow \text{cons } 1 \ (\mathbf{Y} N 2) && \text{reducing } 1 + 1 \text{ to } 2 \\ \mathbf{Y} N 2 &\longrightarrow \underline{N (\mathbf{Y} N) 2} && \text{by the "DNA" rule} \\ &\longrightarrow \text{cons } 2 \ (\mathbf{Y} N \ (2 + 1)) && \text{copying the body of } N \text{ and substituting parameters} \\ &\longrightarrow \text{cons } 2 \ (\mathbf{Y} N 3) && \text{reducing } 2 + 1 \text{ to } 3 \end{aligned}$$

In general,  $\mathbf{Y} N k \longrightarrow^* \text{cons } k \ (\mathbf{Y} N (k + 1))$ . Therefore, by repeatedly unfolding the recursion combinator, we obtain:

$$\begin{aligned} \mathbf{Y} N 1 &\longrightarrow^* \text{cons } 1 \ (\text{cons } 2 \ (\text{cons } 3 \ (\dots))) \\ &= \text{(list } 1 \ 2 \ 3 \ \dots) \end{aligned}$$

In other words, the normal of  $\mathbf{Y} N 1$  is the infinite list of all positive integers (which is an infinite term).

b. Translate the following Java-like pseudocode into a recursive lambda calculus function:

```
int sum (List<int> l) {
  int result = 0;
  while (!l.isEmpty()) {
    result += l.head();
    l = l.tail();
  }
  return result;
}

 $\mathbf{Y} (\lambda f. \lambda l. \text{if null } l \ \text{then } 0 \ \text{else } (\text{car } l) + (f \ (\text{cdr } l)))$ 
```

## Exercise 2: Infinite data structures

a. Consider the following functions:

$$\begin{aligned} A &= \lambda f. \lambda y. \lambda z. \text{cons} ((\text{car } y) + (\text{car } z)) (f (\text{cdr } y) (\text{cdr } z)) \\ F &= \lambda l. \text{cons } 1 (\text{cons } 1 (\mathbf{Y} A l (\text{cdr } l))) \end{aligned}$$

Here are the first couple of reduction steps for the term  $\mathbf{Y} F$ , always choosing the outermost redex.

$$\begin{aligned} \mathbf{Y} F &\longrightarrow F (\mathbf{Y} F) \\ &\longrightarrow \text{cons } 1 (\text{cons } 1 (\mathbf{Y} A (\mathbf{Y} F) (\text{cdr } (\mathbf{Y} F)))) \end{aligned}$$

Carry out enough beta reductions for the remaining term  $(\mathbf{Y} A (\mathbf{Y} F) (\text{cdr } (\mathbf{Y} F)))$  until you can see a couple of elements of this cons-list. Can you guess what  $\mathbf{Y} F$  reduces to, from these observations?

A straightforward  $\beta$ -reduction of this term can get a bit unwieldy. So, I present some short cuts (which are of the kind that you can also employ in trying to simplify terms).

As a first step, we try to figure out what  $\mathbf{Y} A$  does. Recall that  $\mathbf{Y} A$  means a recursively defined function  $f = A(f)$ . What is the type of this function? Looking at the definition, we find that it takes two arguments  $y$  and  $z$  both of which are lists of integers. The result of  $f$  is again a list of integers. Consider what happens if we apply  $\mathbf{Y} A$  to two infinite lists:

$$\begin{aligned} \mathbf{Y} A (\text{list } y_1 y_2 y_3 \dots) (\text{list } z_1 z_2 z_3 \dots) &\longrightarrow A (\mathbf{Y} A) (\text{list } y_1 y_2 y_3 \dots) (\text{list } z_1 z_2 z_3 \dots) \\ &\longrightarrow^* \text{cons } (y_1 + z_1) (\mathbf{Y} A (\text{list } y_2 y_3 \dots) (\text{list } z_2 z_3 \dots)) \\ &\longrightarrow \text{cons } (y_1 + z_1) (A (\mathbf{Y} A) (\text{list } y_2 y_3 \dots) (\text{list } z_2 z_3 \dots)) \\ &\longrightarrow^* \text{cons } (y_1 + z_1) (\text{cons } (y_2 + z_2) (A (\mathbf{Y} A) (\text{list } y_3 \dots) (\text{list } z_3 \dots))) \end{aligned}$$

If this reduction is carried out *ad infinitum*, we get the infinite list

$$(\text{list } (y_1 + z_1) (y_2 + z_2) (y_3 + z_3) \dots)$$

In other words,  $\mathbf{Y} A$  adds all the corresponding elements of the two argument lists to produce a result list of their sums. Let us use the mnemonic notation  $y \oplus z$  for a term of the form  $\mathbf{Y} A y z$ . We can also redo a similar calculation as above to note that

$$(\text{cons } y_1 y_t) \oplus (\text{cons } z_1 z_t) \longrightarrow^* \text{cons } (y_1 + z_1) (y_t \oplus z_t)$$

Returning to the problem of calculating  $\mathbf{Y} F$ , we use the abbreviation  $\mathbf{L}$  for  $\mathbf{Y} F$ . (This notation has no effect other than reducing the number of parentheses we need in expressions.) The calculations already shown in the problem are now restated as:

$$\begin{aligned} \mathbf{L} &\longrightarrow F (\mathbf{L}) \\ &\longrightarrow \text{cons } 1 (\text{cons } 1 (\mathbf{L} \oplus (\text{cdr } \mathbf{L}))) \end{aligned}$$

Applying  $\text{cdr}$  to this list gives:

$$\text{cdr } \mathbf{L} \longrightarrow^* \text{cons } 1 (\mathbf{L} \oplus (\text{cdr } \mathbf{L}))$$

Now, we are required to calculate the sum of these two lists  $\mathbf{L} \oplus (\text{cdr } \mathbf{L})$ . Call the sum  $\xi_0$ .

it is easy to see that:

$$\begin{aligned} \xi_0 = \mathbf{L} \oplus (\text{cdr } \mathbf{L}) &\longrightarrow^* (\text{cons } 1 (\text{cons } 1 S_0)) \oplus (\text{cons } 1 \xi_0) \\ &\longrightarrow^* \text{cons } (1 + 1) ((\text{cons } 1 \xi_0) \oplus \xi_0) \\ &\longrightarrow \text{cons } 2 ((\text{cons } 1 \xi_0) \oplus \xi_0) \end{aligned}$$

Now, we are tasked with calculating another sum  $(\text{cons } 1 \ \xi_0) \oplus \xi_0$ . Call this list  $\xi_1$  and proceed:

$$\begin{aligned} \xi_1 &= (\text{cons } 1 \ \xi_0) \oplus \xi_0 &\longrightarrow^* & \underline{(\text{cons } 1 \ \xi_0) \oplus ((\text{cons } 2 \ \xi_1))} \\ & &\longrightarrow^* & \text{cons } \underline{(1 + 2)} \ (\xi_0 \oplus \xi_1) \\ & &\longrightarrow & \text{cons } 3 \ (\xi_0 \oplus \xi_1) \\ \xi_2 &= \xi_0 \oplus \xi_1 &\longrightarrow^* & \underline{(\text{cons } 2 \ \xi_1) \oplus (\text{cons } 3 \ \xi_2)} \\ & &\longrightarrow^* & \text{cons } \underline{(2 + 3)} \ (\xi_1 \oplus \xi_2) \\ & &\longrightarrow & \text{cons } 5 \ (\xi_1 \oplus \xi_2) \\ \xi_3 &= \xi_1 \oplus \xi_2 &\longrightarrow^* & \underline{(\text{cons } 3 \ \xi_2) \oplus (\text{cons } 5 \ \xi_3)} \\ & &\longrightarrow^* & \text{cons } \underline{(3 + 5)} \ (\xi_2 \oplus \xi_3) \\ & &\longrightarrow & \text{cons } 8 \ (\xi_2 \oplus \xi_3) \end{aligned}$$

Looking at the information produced about the list  $\mathbf{L}$  so far, we find:

$$\mathbf{L} = (\text{list } 1 \ 1 \ 2 \ 3 \ 5 \ 8 \ \dots)$$

Notice that, after the first two 1's, every element in the list is the sum of the previous two elements. This sequence of integers is called the *Fibonacci series*.

The trick to producing the answer without getting lost in the detail has been to name the intermediate lists being calculated:  $\xi_0, \xi_1, \xi_2, \dots$ . Giving names to these lists has a subtle consequence. If the same list is required in multiple places in the calculation, we are able to reuse its calculation without having to redo it. This kind of evaluation is called “call by need” or “lazy evaluation”.

It is also possible to do this calculation without having to name all the intermediate lists as we have done above. Suppose we write out the known elements of the lists  $\mathbf{L}$ ,  $\text{cdr } \mathbf{L}$  and  $\mathbf{L} \oplus (\text{cdr } \mathbf{L})$  in a table as follows:

$$\begin{array}{rcl} \mathbf{L} & = & 1 \quad 1 \quad \dots \\ \text{cdr } \mathbf{L} & = & 1 \quad \dots \\ \hline \mathbf{L} \oplus (\text{cdr } \mathbf{L}) & = & \dots \end{array}$$

Notice that the unknown elements represented by  $\dots$  are exactly the same in all the three places. (After the first two elements, the remainder of  $\mathbf{L}$  is  $\mathbf{L} \oplus (\text{cdr } \mathbf{L})$ .) We obtain the first element of  $\mathbf{L} \oplus (\text{cdr } \mathbf{L})$  by adding the elements of  $\mathbf{L}$  and  $\text{cdr } \mathbf{L}$ , but this information now applies to  $\mathbf{L}$  and  $\text{cdr } \mathbf{L}$  as well. Notice:

$$\begin{array}{rcl} \mathbf{L} & = & 1 \quad 1 \quad 2 \quad \dots \\ \text{cdr } \mathbf{L} & = & 1 \quad 2 \quad \dots \\ \hline \mathbf{L} \oplus (\text{cdr } \mathbf{L}) & = & 2 \quad \dots \end{array}$$

We can continue doing the calculation of the elements *ad infinitum*:

$$\begin{array}{rcl} \mathbf{L} & = & 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad \dots \\ \text{cdr } \mathbf{L} & = & 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad \dots \\ \hline \mathbf{L} \oplus (\text{cdr } \mathbf{L}) & = & 2 \quad 3 \quad 5 \quad 8 \quad \dots \end{array}$$

- b. Define a function called *series* which takes arguments  $k$  and  $n$ , and generates an infinite series of the integers  $n, n + k, n + 2k, \dots$  as an infinite list.

$$\text{series } k \ n = \text{cons } n \ (\text{series } k \ (n + k))$$

- c. Define a function called *common*, which takes as arguments two infinite lists  $x$  and  $y$  of integers in ascending order, and produces another infinite list of all the integers that occur in both  $x$  and  $y$ .

$$\begin{aligned} \text{common } x \ y &= \text{if } (\text{car } x) = (\text{car } y) \ \text{then } \text{cons } (\text{car } x) \ (\text{common } (\text{cdr } x) \ (\text{cdr } y)) \\ &\quad \text{else if } (\text{car } x) < (\text{car } y) \ \text{then } (\text{common } (\text{cdr } x) \ y) \\ &\quad \text{else } (\text{common } x \ (\text{cdr } y)) \end{aligned}$$

- d. Describe the effect of the expression:

$$\text{common } (\text{series } 2 \ 0) \ (\text{series } 3 \ 0)$$

It is an infinite list of all the multiples of 6, including 0. (The two arguments to *common* are the list of multiples of 2 and the list of multiples of 3. The common elements are then all the multiples of 6.)