TP 2 : Higher Order

©2022 Ghiles Ziat
ghiles.ziat@epita.fr

Exercise I : Recurrence relation

Let u_n be defined by :

$$\begin{cases} u_0 & = & 42 \\ u_{n+1} & = & 3u_n + 4 \end{cases}$$

Q1 – Give the definition of the function `u : int -> int` such that `(u n)` returns $u_{\mathtt{n}}$ assuming `n` positive.

Q2 – This type of calculation can be generalized. If f is the unary function $x \mapsto 3x + 4$ then $u_n = f^n(42)$, where f^n is the iteration of the function f, n times. Intuitively, $f^n(a) = f(f(\dots f(a)\dots))$. We define f^n as follows :

$$\begin{cases} f^0(x) & = & x \\ f^{n+1} & = & f(f^n(x)) \end{cases}$$

To obtain a general mechanism for this kind of calculation, we define a *higher-order function*, say $\mathcal{I}$ that takes as argument an integer, index of the iteration, the function f to iterate and the value a of the base case and which satisfies the following equations :

$$\begin{cases} \mathcal{I}(0, f, a) & = & a \\ \mathcal{I}(n+1, f, a) & = & f(\mathcal{I}(n, f, a)) \end{cases}$$

Following this recursive pattern, give the definition of the function `iter : int -> (int -> int) -> int -> int` such that `(iter n f a)` gives $\mathtt{f}^{\mathtt{n}}(\mathtt{a})$, for $\mathtt{n} \geq 0$.

Q3 – Verify that the calls `(u 10)` and `(iter 42 f 10)`, where f is an anonymous function that maps x to $(3x + 4)$, yield the same result.

Exercise II : Merging Lists

We propose to implement the function `flatten` which flattens a list of lists in the following sense :

if `l = [ [`$a_{1,1}$`;`$\cdots$`;`$a_{n_1,1}$`]; `$\cdots$`; [`$a_{1,m}$`;`$\ldots$`;`$a_{n_m,m}$` ]]`

then `flatten l = [`$a_{1,1}$`;`a_{2,n_1}`;`$\ldots$`;`$a_{n_1,1}$`;`$a_{1,2}$`;`$\ldots$`;`$a_{n_2,2}$`;`$\ldots$`;`$a_{1,m}$`;`$\ldots$`;`$a_{n_m,m}$`]`

Example
```
flatten [[1;2;3];[4;5;6];[];[7;8]] = [1;2;3;4;5;6;7;8]
flatten [[];[];[];[]] = []
```

Q1 – Give the function signature `flatten`

Q2 – Propose an implementation of the function `flatten`

Q3 – Write a function `revert_and_push` that takes into argument two lists, returns (in the sense of lists) the first and the puts at the head of the second.

Example
```
revert_and_push [3;2;1] [4;5;6] = [1;2;3;4;5;6]
```

Exercise III : Permutations of a list

Q1 – Write a function `insert` that takes two arguments : an element `x` to insert and a list in which to insert x. This function should return a list of lists where x has been inserted at all possible positions.

Example
```
insert 0 [] = [[0]]
insert 0 [1;2;3] = [[0;1;2;3];[1;0;2;3];[1;2;0;3];[1;2;3 ;0]]
```

Q2 – We will now use this insert function to write our `perm` function. For this we will iterate through the list and for each item, insert the item at all the positions of the permutations of the rest of the list (you can reuse the function `flatten` from the exercise previous).

Example
```
perm[0;1;2] = [[0;1;2]; [1;0;2]; [1;2;0]; [0;2;1]; [2;0;1]; [2;1;0]]
```

Exercise IV : Inverse function

Q1 – Define a function `equiv` that takes a list of elements l, two functions f and g, and verifies that f and g are equivalent for all entries of l, *i.e.* $\forall x \in l, f(x) = g(x)$.

We want to have a mechanism that given a function f builds its inverse function $g = f^{-1}$.

Q2 – Define a function `inverse` that takes a list of elements $\{e_1, e_2, \ldots, e_n\}$ and a function f to invert. It should return an anonymous unary function which will verify if its argument is equal to one of the images $\{(f\ e_1), (f\ e_2), \ldots, (f\ e_n)\}$ and return its pre-image if it is the case. We can raise an error if the function is called on an element whose pre-image is not in the list.

Example

```
(inverse [0;1;2] succ) 1 = 0
(inverse [0;1;2] succ) 3 = 2
(inverse [0;1;2] succ) 4 = error "entry not in co-domain"
```

Q3 – Define a function `square` that maps to an integer x its square, and a function `mysqrt` which will be the inverse of `square` on the list of the first 10 integers.

Q4 – Check that your function `mysqrt` is equivalent to the primitive `sqrt` already defined in the language on a given list of perfect squares (*e.g.* `[1, 4, 9, 16]`).