Homework 1
ENE4014 Programming Languages, Spring 2020
Woosuk Lee
due: 4/13(Mon), 24:00

Exercise 1 Write a function

$$\texttt{rev\_append} : \texttt{`alist} \to \texttt{`alist} \to \texttt{`alist}.$$

`rev_append l1 l2` reverses `l1` and concatenates it to `l2`. For example,

$$\begin{aligned}\texttt{rev\_append}\ [1;2;3]\ [4;5] &= [3;2;1;4;5]\\ \texttt{rev\_append}\ [3;5]\ [2;5] &= [5;3;2;5]\\ \texttt{rev\_append}\ [3;5]\ [] &= [5;3]\\ \texttt{rev\_append}\ []\ [2;1] &= [2;1]\end{aligned}$$

□

Exercise 2 Write a function

$$\texttt{range} : \texttt{int} \to \texttt{int} \to \texttt{int list}.$$

`range lower upper` generates a sorted list of integers in the range [`lower` ... `upper`]. If `lower` is greater than `upper`, the function returns the empty list. For example,

$$\begin{aligned}\texttt{range}\ 1\ 3 &= [1;2;3]\\ \texttt{range}\ (-2)\ 2 &= [-2;-1;0;1;2]\\ \texttt{range}\ 2\ 2 &= [2]\\ \texttt{range}\ 2\ (-2) &= []\end{aligned}$$

□

Exercise 3 Write a function

$$\texttt{partition} : (\texttt{`a} \to \texttt{bool}) \to \texttt{`a list} \to \texttt{`a list} * \texttt{`a list}.$$

`partition p l` returns a pair of lists `(l1, l2)`, where `l1` is the list of all the elements of `[l]` that satisfy the predicate `p`, and `l2` is the list of all the elements

of `l` that do not satisfy `p`. The order of the elements in the input list is preserved. For example,

$$\begin{array}{rcl}
\texttt{partition (fun x} \rightarrow \texttt{x} > 0)\ [-1;2;0;3] & = & ([2;3],[-1;0]) \\
\texttt{partition (fun x} \rightarrow \texttt{x} = 0)\ [3;2;0;1] & = & ([0],[3;2;1]) \\
\texttt{partition (fun x} \rightarrow \texttt{x mod 2} = 1)\ [1;2;3;4] & = & ([1;3],[2;4]) \\
\texttt{partition (fun x} \rightarrow \texttt{x mod 2} = 0)\ [1;2;3;4] & = & ([2;4],[1;3]) \\
\texttt{partition (fun x} \rightarrow \texttt{false})\ [1;2;3;4] & = & ([],[1;2;3;4]) \\
\texttt{partition (fun x} \rightarrow \texttt{true})\ [1;2;3;4] & = & ([1;2;3;4],[])
\end{array}$$

□

```
type formula = TRUE | FALSE
  | NOT of formula
  | ANDALSO of formula * formula
  | ORELSE of formula * formula
  | IMPLY of formula * formula
  | LESS of expr * expr
and expr = NUM of int
  | PLUS of expr * expr
  | MINUS of expr * expr
```

Exercise 4 Consider the above propositional formula:
Write a function

$$\texttt{eval} : \texttt{formula} \rightarrow \texttt{bool}$$

that computes the truth value of a given formula. For example,

```
eval (IMPLY (IMPLY (TRUE, FALSE), TRUE))
```

evaluates to *true*, and

```
eval (LESS (NUM 5, PLUS (NUM 1, NUM 2)))
```

evaluates to *false*. □

Exercise 5 Binary trees can be defined as follows:

```
type btree = Empty | Node of int * btree * btree
```

For example, the following `t1` and `t2` are binary trees.

```
let t1 = Node (1, Empty, Empty)
let t2 = Node (1, Node (2, Empty, Empty), Node (3, Empty, Empty))
let t3 = Node (1, Node (2, Node (3, Empty, Empty), Empty), Empty)
```

Write a function

$$\texttt{height} : \texttt{btree} \rightarrow \texttt{int}$$

that computes the height of a given binary tree. The height of a given binary tree is inductively defined as follows:

$$\begin{aligned} \texttt{height Empty} &= 0 \\ \texttt{height (Node(n, l, r))} &= \begin{cases} (\texttt{height l}) + 1 & (\texttt{height l} > \texttt{height r}) \\ (\texttt{height r}) + 1 & (\text{otherwise}) \end{cases} \end{aligned}$$

For example,

$$\begin{aligned} \texttt{height Empty} &= 0 \\ \texttt{height t1} &= 1 \\ \texttt{height t2} &= 2 \\ \texttt{height t3} &= 3 \end{aligned}$$

□

Exercise 6 Write a function

$$\texttt{notexists} : \texttt{int} \to \texttt{btree} \to \texttt{bool}.$$

`notexists n t` returns true if `n` is not in the tree `t` (otherwise false). For example,

$$\begin{aligned} \texttt{notexists 1 Empty} &= \texttt{true} \\ \texttt{notexists 1 t1} &= \texttt{false} \\ \texttt{notexists 2 t1} &= \texttt{true} \\ \texttt{notexists 2 t2} &= \texttt{false} \end{aligned}$$

□

Exercise 7 The `fold` function for lists

$$\texttt{fold} : (\texttt{'a} \to \texttt{'b} \to \texttt{'a}) \to \texttt{'a} \to \texttt{'b list} \to \texttt{'a}$$

recombines the results of recursively processing its constituent parts, building up a return value through use of a given combining operation. For example,

$$\texttt{fold f a [b1; ...; bn]} = \texttt{f (...(f (f a b1) b2)...) bn}.$$

Extend the following function that takes three lists. Write a function

$$\texttt{fold3} : (\texttt{'a} \to \texttt{'b} \to \texttt{'c} \to \texttt{'d} \to \texttt{'a}) \to \texttt{'a} \to \texttt{'b list} \to \texttt{'c list} \to \texttt{'d list} \to \texttt{'a}$$

of which meaning is defined as follow:

$$\begin{aligned} &\texttt{fold3 f a [b1; ...; bn] [c1; ...; cn] [d1; ...; dn]} \\ &= \texttt{f (...(f (f a b1 c1 d1) b2 c2 d2)...) bn cn dn}. \end{aligned}$$

You may assume that all the given lists are of the same length. □

Exercise 8 Write a function

$$\texttt{(iter } n \ f\texttt{)} = \underbrace{f \circ \cdots \circ f}_{|n|}$$

The function returns the identity function (fun x $\rightarrow$ x) when $n = 0$.

For example,

```
(iter n (fun x -> x + 2)) 0
```

returns $2 \times n$. □

Exercise 9 Write a function

```
sigma :  int * int * (int -> int) -> int.
```

such that `sigma(a,b,f)` returns $\Sigma_{n=a}^{b} f(n)$. □

Exercise 10 Write a function

```
cartesian:  'a list -> 'b list -> ('a * 'b) list
```

that returns a list of from two lists. That is, for lists A and B, the Cartesian product $A \times B$ is the list of all ordered pairs (a, b) where $a \in A$ and $b \in B$. For example, if $A = [\text{“}a''; \text{“}b''; \text{“}c'']$ and $B = [1; 2; 3]$, $A \times B$ is defined to be

$$[(\text{“}a'', 1); (\text{“}a'', 2); (\text{“}a'', 3); (\text{“}b'', 1); (\text{“}b'', 2); (\text{“}b'', 3); (\text{“}c'', 1); (\text{“}c'', 2); (\text{“}c'', 3)]$$

□