Programming Languages: Functional Programming Additional Exercises

Shin-Cheng Mu

Autumn, 2025

1. Recall one of the de Morgan laws: $not\ (a \wedge b) = not\ a \vee not\ b$. The functions and and or below are respectively $(\wedge)$ and $(\vee)$ extended to lists:

$$
\begin{array}{l}
and, or :: \mathsf{List\ Bool} \to \mathsf{Bool} \\
and\ [\,] \quad\ \ = \mathsf{True} \\
and\ (a : as) = a \wedge and\ as\ \ , \\
or\ \ [\,] \quad\ \ = \mathsf{False} \\
or\ \ (a : as) = a \vee or\ as\ \ .
\end{array}
$$

   (a) Prove that $or \cdot map\ not = not \cdot and$ by a typical inductive proof.

> **Solution:** We aim to prove that $or\ (map\ not\ as) = not\ (and\ as)$ by induction on as. In the base case $as := [\,]$ both sides reduce to False. For the case $as := a : as$ we reason:
>
> $$
> \begin{array}{ll}
> & or\ (map\ not\ (a : as)) \\
> = & \quad \{\ \mathsf{definition\ of}\ map\ \} \\
> & or\ (not\ a : map\ not\ as) \\
> = & \quad \{\ \mathsf{definition\ of}\ or\ \} \\
> & not\ a \vee or\ (map\ not\ as) \\
> = & \quad \{\ \mathsf{induction}\ \} \\
> & not\ a \vee not\ (and\ as) \\
> = & \quad \{\ \mathsf{de\ Morgan}\ \} \\
> & not\ (a \wedge and\ as) \\
> = & \quad \{\ \mathsf{definition\ of}\ and\ \} \\
> & not\ (and\ (a : as))\ \ .
> \end{array}
> $$

   (b) Define and in terms of $foldr$.

Solution:

$$and = foldr\ (\wedge)\ \mathsf{True}\ .$$

(c) Prove that $or \cdot map\ not = not \cdot and$ by two $foldr$-fusion (fusing both side into the same $foldr$). Compare your proof with the previous proof by induction.

Solution: The proof goes:

$$\begin{array}{ll} & or \cdot map\ not \\ = & \quad \{\ map \text{ as a } foldr\ \} \\ & or \cdot foldr\ (\lambda a\ bs \to not\ a : bs)\ [\,] \\ = & \quad \{\ foldr\text{-fusion 1.}\ \} \\ & foldr\ (\lambda a\ b \to not\ a \vee b)\ \mathsf{False} \\ = & \quad \{\ foldr\text{-fusion 2.}\ \} \\ & not \cdot foldr\ (\wedge)\ \mathsf{True} \\ = & \quad \{\ and \text{ as a } foldr\ \} \\ & not \cdot and \end{array}$$

The fusion condition for $foldr$-fusion 1. is "proved" simply by expanding the definition:

$$\begin{array}{ll} & or\ (not\ a : bs) \\ = & \quad \{\ \text{definition of } or\ \} \\ & not\ a \vee or\ bs\ . \end{array}$$

(Note that it is in this fusion we discover $(\lambda a\ b \to not\ a \mid b)$.)
The fusion condition for $foldr$-fusion 1. is exactly the de Morgan law.

$$\begin{array}{ll} & not\ (a \wedge b) \\ = & \quad \{\ \text{de Morgan}\ \} \\ & not\ a \vee not\ b\ . \end{array}$$

The proof by $foldr$-fusion does not look textually "shorter", since we went very slow, and added lots verbal explanations around the fusion condition. However, the first fusion condition is simply the definition, while the second condition is the de Morgan law. When one goes quicker, these two steps could be explained away in two sentences. In the proof by fusion we do not need to perform the induction step ourselves. We only need to provide the most essential part to the proof — the de Morgan law.

2. Recall the function $sublists$ :: List $a \to$ List (List a), which computes all sublists of a given

list, defined by:

$$
\begin{aligned}
&\mathit{sublists} :: \mathsf{List}\ a \to \mathsf{List}\ (\mathsf{List}\ a) \\
&\mathit{sublists}\ [\,] \qquad = [[\,]] \\
&\mathit{sublists}\ (x : xs) = \mathit{yss} +\!\!+ \mathit{map}\ (x{:})\ \mathit{yss}\ \ , \\
&\quad \textbf{where}\ \mathit{yss} = \mathit{sublists}\ xs\ \ .
\end{aligned}
$$

For example, $\mathit{sublists}\ [1,2,3] = [[\,],[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]]$.

(a) Define $\mathit{sublists}$ in terms of foldr.

> **Solution:** $\mathit{sublists} = \mathit{foldr}\ (\lambda x\ \mathit{xss} \to \mathit{xss} +\!\!+ \mathit{map}\ (x{:})\ \mathit{xss})\ [[\,]]$

(b) Recall the function $\mathit{exp}\ b\ n$, which computes b^n, and assume that $\mathit{exp}\ b\ n$ meets all the usual arithmetics properties. Prove that $\mathit{length} \cdot \mathit{sublists} = \mathit{exp}\ 2 \cdot \mathit{length}$ — that is, a list of lengh n has 2^n sublists. Prove the property by using foldr-fusion twice. You may need some properties regarding the interaction between length and $(+\!\!+)$ or map. State clearly what properties you assume (you don't have to prove them).

> **Solution:** Consider fusing $\mathit{length} \cdot \mathit{sublists}$. We try to construct a step function that meets the fusion condition:
>
> $$
> \begin{aligned}
> & \mathit{length}\ (\mathit{xss} +\!\!+ \mathit{map}\ (x{:})\ \mathit{xss}) \\
> = {} & \mathit{length}\ \mathit{xss} + \mathit{length}\ (\mathit{map}\ (x{:})\ \mathit{xss}) \\
> = {} & \quad \{\ \mathit{length}\ (\mathit{map}\ f) = \mathit{length}\ \} \\
> & 2 \times \mathit{length}\ \mathit{xss}\ \ .
> \end{aligned}
> $$
>
> Thus the step function is $(\lambda x\ n \to 2 \times n)$.
>
> The property can be proved as below:
>
> $$
> \begin{aligned}
> & \mathit{length} \cdot \mathit{sublists} \\
> = {} & \mathit{length} \cdot \mathit{foldr}\ (\lambda x\ \mathit{xss} \to \mathit{xss} +\!\!+ \mathit{map}\ (x{:})\ \mathit{xss})\ [[\,]] \\
> = {} & \quad \{\ \mathit{foldr}\text{-fusion, as above}\ \} \\
> & \mathit{foldr}\ (\lambda x\ n \to 2 \times n)\ 1 \\
> = {} & \quad \{\ \mathit{foldr}\text{-fusion, see below}\ \} \\
> & \mathit{exp}\ 2 \cdot \mathit{foldr}\ (\lambda x\ n \to 1 + n)\ 0 \\
> = {} & \mathit{exp}\ 2 \cdot \mathit{length}\ \ .
> \end{aligned}
> $$
>
> In the second foldr-fusion, the base value $\mathit{exp}\ 2\ 0$ is 1. The fusion condition is
>
> $$\mathit{exp}\ 2\ (\mathbf{1}_{+}\ n) = 2 \times \mathit{exp}\ 2\ n\ \ ,$$
>
> which holds by definition of exp.

3. This question and the next concern the following binary tree:

$$\textbf{data}\ \mathsf{ITree}\ a = \mathsf{Null} \mid \mathsf{Node}\ (\mathsf{ITree}\ a)\ a\ (\mathsf{ITree}\ a)\ .$$

(Note that the ITree differs a little from the one in the handouts... just for some variation!) Given $t :: \mathsf{ITree}\ a$, $dist\ t$ computes the distance between the root and the nearest Null:

$$\begin{aligned} &dist :: \mathsf{ITree}\ a \to \mathsf{Nat} \\ &dist\ \mathsf{Null} &&= 0 \\ &dist\ (\mathsf{Node}\ t\ x\ u) &&= 1 + (dist\ t \downarrow dist\ u)\ . \end{aligned}$$

"Leftist tree" is a kind of tree intentionally lean towards the left. For a verbal definition: Null is a leftist tree; a tree t having two subtrees is a leftist tree if and only if the $dist$ value of the left subtree is no less than the $dist$ value of the right subtree, and both subtrees are leftist trees too. (We omit other non-structural properties usually included in literatures.)

(a) Define $leftist :: \mathsf{ITree}\ a \to \mathsf{Bool}$ that determines whether a tree is a leftist tree, in a way matching the verbal definition above.

Solution:

$$\begin{aligned} &leftist :: \mathsf{ITree}\ a \to \mathsf{Bool} \\ &leftist\ \mathsf{Null} &&= \mathsf{True} \\ &leftist\ (\mathsf{Node}\ t\ x\ u) &&= dist\ t \geqslant dist\ u \wedge leftist\ t \wedge leftist\ u\ . \end{aligned}$$

(b) It is likely that the definition you gave needs $O(n^2)$ basic operations when the input tree contains n nodes. Construct, by tupling, a more general function that returns more information and uses only $O(n)$ operations, and redefine $leftist$ in terms of the new function. **Note**: a complete solution should contain a specification of the generalized function, a derivation, and a new definition of $leftist$.

Solution: Define $leftdist\ t = (leftist\ t, dist\ t)$. Therefore, $leftist = fst \cdot leftdist$.

Now we construct an inductive definition of $leftdist$. Obviously $leftdist\ \mathsf{Null} = (\mathsf{True}, 0)$. Consider the inductive case:

$$\begin{aligned} &\ \ leftdist\ (\mathsf{Node}\ t\ x\ u) \\ =&\quad \{\ \text{defintion of } leftdist\ \} \\ &\ \ (leftist\ (\mathsf{Node}\ t\ x\ u), dist\ (\mathsf{Node}\ t\ x\ u)) \\ =&\quad \{\ \text{definitions of } leftist \text{ and } dist\ \} \\ &\ \ (dist\ t \geqslant dist\ u \wedge leftist\ t \wedge leftist\ u, 1 + (dist\ t \downarrow dist\ u)) \\ =&\quad \{\ \text{extract common subexpressions}\ \} \\ &\ \ \textbf{let}\ (b, h) = (leftist\ t, dist\ t) \\ &\qquad\ (c, k) = (leftist\ u, dist\ u) \\ &\ \ \textbf{in}\ (h \geqslant k \wedge b \wedge c, 1 + (h \downarrow k)) \\ =&\quad \{\ \text{definition of } leftdist\ \} \\ &\ \ \textbf{let}\ (b, h) = leftdist\ t \end{aligned}$$

$$
\begin{array}{l}
\qquad (c, k) = \mathit{leftdist}\ u \\
\quad \mathbf{in}\ (h \geqslant k \wedge b \wedge c, 1 + (h \downarrow k))\ .
\end{array}
$$

Therefore:

$$
\begin{array}{l}
\mathit{leftdist} :: \mathsf{ITree}\ a \to (\mathsf{Bool}, \mathsf{Nat}) \\
\mathit{leftdist}\ \mathsf{Null} \qquad\qquad = (\mathsf{True}, 0) \\
\mathit{leftdist}\ (\mathsf{Node}\ t\ x\ u) = \mathbf{let}\ (b, h) = \mathit{leftdist}\ t \\
\qquad\qquad\qquad\qquad\qquad\qquad (c, k) = \mathit{leftdist}\ u \\
\qquad\qquad\qquad\qquad\qquad\quad \mathbf{in}\ (h \geqslant k \wedge b \wedge c, 1 + (h \downarrow k))\ .
\end{array}
$$

(c) The fold for ITree is defined by

$$
\begin{array}{l}
\mathit{foldIT} :: (b \to a \to b \to b) \to b \to \mathsf{ITree}\ a \to b \\
\mathit{foldIT}\ f\ e\ \mathsf{Null} \qquad\qquad = e \\
\mathit{foldIT}\ f\ e\ (\mathsf{Node}\ t\ x\ u) = f\ (\mathit{foldIT}\ f\ e\ t)\ x\ (\mathit{foldIT}\ f\ e\ u)\ ,
\end{array}
$$

with a foldIT-fusion theorem:

$$
\begin{array}{l}
h \cdot \mathit{foldIT}\ f\ e = \mathit{foldIT}\ g\ (h\ e) \\
\quad \Leftarrow h\ (f\ y\ x\ z) = g\ (h\ y)\ x\ (h\ z)\ .
\end{array}
$$

And we have that $\mathit{foldIT}\ \mathsf{Node}\ \mathsf{Null} = id$ when id has type $\mathsf{ITree}\ a \to \mathsf{ITree}\ a$. Again let $\mathit{leftdist}\ t = (\mathit{leftist}\ t, \mathit{dist}\ t)$. Try constructing a faster version of $\mathit{leftdist}$ by fusing $\mathit{leftdist} \cdot id$ where id is a foldIT.

Solution:

$$
\begin{array}{ll}
 & \mathit{leftdist} \\
= & \quad \{\ f = f \cdot id\ \} \\
 & \mathit{leftdist} \cdot id \\
= & \quad \{\ id \text{ is a } \mathit{foldIT}\ \} \\
 & \mathit{leftdist} \cdot \mathit{foldIT}\ \mathsf{Node}\ \mathsf{Null} \\
= & \quad \{\ \mathit{foldIT}\text{-fusion, with } ld \text{ constructed below}\ \} \\
 & \mathit{foldIT}\ ld\ (\mathsf{True}, 0)\ .
\end{array}
$$

Below we prove the fusion condition and also construct ld:

$$
\begin{array}{ll}
 & \mathit{leftdist}\ (\mathsf{Node}\ t\ x\ u) \\
= & \quad \{\ \text{defintion of } \mathit{leftdist}\ \} \\
 & (\mathit{leftist}\ (\mathsf{Node}\ t\ x\ u), \mathit{dist}\ (\mathsf{Node}\ t\ x\ u)) \\
= & \quad \{\ \text{definitions of } \mathit{leftist} \text{ and } \mathit{dist}\ \} \\
 & (\mathit{dist}\ t \geqslant \mathit{dist}\ u \wedge \mathit{leftist}\ t \wedge \mathit{leftist}\ u, 1 + (\mathit{dist}\ t \downarrow \mathit{dist}\ u))
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \text{ extract common subexpressions } \} \\
& \textbf{let } (b,h) = (\mathit{leftist}\ t, \mathit{dist}\ t) \\
& \quad\ (c,k) = (\mathit{leftist}\ u, \mathit{dist}\ u) \\
& \textbf{in } (h \geqslant k \wedge b \wedge c, 1 + (h \downarrow k)) \\
= & \{ \text{ definition of } \mathit{leftdist} \} \\
& \textbf{let } (b,h) = \mathit{leftdist}\ t \\
& \quad\ (c,k) = \mathit{leftdist}\ u \\
& \textbf{in } (h \geqslant k \wedge b \wedge c, 1 + (h \downarrow k)) \ . \\
= & \{ \ \mathit{ld} \text{ given below } \} \\
& \mathit{ld}\ (\mathit{leftdist}\ t)\ x\ (\mathit{leftdistu}) \ ,
\end{array}
$$

where ld is defined by:

$$
\mathit{ld}\ (b,h)\ x\ (c,k) = (h \geqslant k \wedge b \wedge c, 1 + (h \downarrow k)) \ .
$$

The proof, in fact, is exactly the inductive case in the previous subproblem. It is a demonstration that our approach of "constructing an inductive definition" is an instance of fold fusion — in particular, fusing with id.

4. The *depth* of a node in an ITree is defined by: the depth of the root is 0; if t becomes a subtree (of a root), its depth is one plus its original depth. The function below computes the depths of each labels in an ITree:

$$
\begin{array}{l}
\mathit{depths} :: \mathsf{ITree}\ a \to \mathsf{List}\ (a, \mathsf{Nat}) \\
\mathit{depths}\ \mathsf{Null} \qquad\qquad = [\,] \\
\mathit{depths}\ (\mathsf{Node}\ t\ x\ u) = \mathit{map}\ (\mathit{second}\ (1+))\ (\mathit{depths}\ t) +\!\!+ [(x,0)] +\!\!+ \\
\qquad\qquad\qquad\qquad\quad \mathit{map}\ (\mathit{second}\ (1+))\ (\mathit{depths}\ u) \ ,
\end{array}
$$

where $\mathit{second}\ f\ (x,y) = (x, f\ y)$ (note that second is ***not*** snd, a function having a similar name). For example, given t :: ITree Char below, we have $\mathit{depths}\ t = [(\texttt{'a'},2), (\texttt{'b'},1), (\texttt{'c'},0), (\texttt{'d'},3), (\texttt{'e'},2), (\texttt{'f'},1), (\texttt{'g'},2), (\texttt{'h'},3)]$.

```
t = Node (Node (Node Null 'a' Null) 'b' Null)
         'c'
         (Node (Node (Node Null 'd' Null) 'e' Null) 'f'
               (Node Null 'g' (Node Null 'h' Null))) ,
```

Given a tree t having n nodes, the function $\mathit{depths}\ t$ needs $O(n^2)$ additions in the worst case. To improve its efficiency, we define:

$$
\mathit{depthsAcc}\ t\ k = \mathit{map}\ (\mathit{second}\ (k+))\ (\mathit{depths}\ t) \ .
$$

(a) Define $depths$ in terms of $depthsAcc$, and derive a definition of $depthsAcc$ that uses only $O(n)$ additions. You might need properties include map fusion, and some properties of $second$. They need not be proved, but state clearly what properties you assumed.

Solution: We may let $depths\ t = depthsAcc\ t\ 0$ provided that the latter has an alternative definition.

Apparently $depthsAcc$ Null $k = [\,]$. Consider the inductive case:

$$
\begin{aligned}
& depthsAcc\ (\mathsf{Node}\ t\ x\ u)\ k \\
= \quad & \{\ \text{definition of } depthsAcc\ \} \\
& map\ (second\ (k+))\ (depths\ (\mathsf{Node}\ t\ x\ u)) \\
= \quad & \{\ \text{definition of } depths\ \} \\
& map\ (second\ (k+))\ (map\ (second\ (1+))\ (depths\ t) +\!\!+ [(x,0)] +\!\!+ \\
& \qquad\qquad map\ (second\ (1+))\ (depths\ u)) \\
= \quad & \{\ map \text{ distributes into } (+\!\!+), \text{ definition of } map\ \} \\
& map\ (second\ (k+))\ (map\ (second\ (1+))\ (depths\ t)) +\!\!+ [(x,k)] +\!\!+ \\
& map\ (second\ (k+))\ (map\ (second\ (1+))\ (depths\ u)) \\
= \quad & \{\ map\text{-fusion}\ \} \\
& map\ (second\ (k+) \cdot second\ (1+))\ (depths\ t) +\!\!+ [(x,k)] +\!\!+ \\
& map\ (second\ (k+) \cdot second\ (1+))\ (depths\ u) \\
= \quad & \{\ second\text{-fusion}, (k+) \cdot (1+) = ((k+1)+)\ \} \\
& map\ (second\ ((k+1)+))\ (depths\ t) +\!\!+ [(x,k)] +\!\!+ \\
& map\ (second\ ((k+1)+))\ (depths\ u) \\
= \quad & \{\ \text{definition of } depthsAcc\ \} \\
& depthsAcc\ t\ (k+1) +\!\!+ [(x,k)] +\!\!+ depthsAcc\ u\ (k+1)\ .
\end{aligned}
$$

By $second$-fusion we meant that for all f and g, $second\ f \cdot second\ g = second\ (f \cdot g)$.

Therefore,

$$
\begin{aligned}
& depthsAcc :: \mathsf{ITree}\ a \to \mathsf{Nat} \to \mathsf{List}\ (a, \mathsf{Nat}) \\
& depthsAcc\ \mathsf{Null} \qquad\qquad k = [\,] \\
& depthsAcc\ (\mathsf{Node}\ t\ x\ u)\ k = \\
& \quad depthsAcc\ t\ (k+1) +\!\!+ [(x,k)] +\!\!+ depthsAcc\ u\ (k+1)\ .
\end{aligned}
$$

(b) Due to the use of $(+\!\!+)$, the solution of the previous subproblem still uses $O(n^2)$ time. Define a more general function and redefine $depthsAcc$ with it, and derive a version that runs in $O(n)$ time. **Note**: again, a complete soltuion should contain the specification of the function, a derivation, and a new defintion of $depthsAcc$.

Solution: Let $depthS\ t\ k\ ys = depthsAcc\ t\ k +\!\!+ ys$. If it has an alternative solution, we may let $depthsAcc\ t\ k = depthS\ t\ k\ [\,]$.

In the base case, $depthS$ Null $k\ ys = depthsAcc$ Null $k +\!\!+ ys = ys$. Consider the

inductive case:

$$
\begin{aligned}
& \mathit{depthS}\ (\mathsf{Node}\ t\ x\ u)\ k\ ys \\
= \quad & \{\ \text{definition of } \mathit{depthS}\ \} \\
& \mathit{depthsAcc}\ (\mathsf{Node}\ t\ x\ u)\ k +\!\!+ ys \\
= \quad & \{\ \text{definition of } \mathit{depthsAcc}\ \} \\
& \mathit{depthsAcc}\ t\ (k+1) +\!\!+ [(x,k)] +\!\!+ \mathit{depthsAcc}\ u\ (k+1) +\!\!+ ys \\
= \quad & \{\ (+\!\!+) \text{ associative, definition of } \mathit{depthS}\ \} \\
& \mathit{depthS}\ t\ (k+1)\ ([(x,k)] +\!\!+ \mathit{depthsAcc}\ u\ (k+1) +\!\!+ ys) \\
= \quad & \{\ \text{definitions of } (+\!\!+) \text{ and } \mathit{depthS}\ \} \\
& \mathit{depthS}\ t\ (k+1)\ ((x,k) : \mathit{depthS}\ u\ (k+1)\ ys)\ \ .
\end{aligned}
$$

Therefore we have

$$
\begin{aligned}
& \mathit{depthS} :: \mathsf{ITree}\ a \to \mathsf{Nat} \to \mathsf{List}\ (a, \mathsf{Nat}) \to \mathsf{List}\ (a, \mathsf{Nat}) \\
& \mathit{depthS}\ \mathsf{Null} \qquad\qquad k\ ys = ys \\
& \mathit{depthS}\ (\mathsf{Node}\ t\ x\ u)\ k\ ys = \\
& \quad \mathit{depthS}\ t\ (k+1)\ ((x,k) : \mathit{depthS}\ u\ (k+1)\ ys)\ \ .
\end{aligned}
$$