Functional Programming
Practicals 1

Shin-Cheng Mu

FLOLAC 2022

Folds and Fold-Fusion

1. Express the following functions by *foldr*:

    1. $all\ p :: \text{List}\ a \to \text{Bool}$, where $p :: a \to \text{Bool}$.
    2. $elem\ z :: \text{List}\ a \to \text{Bool}$, where $z :: a$.
    3. $concat :: \text{List}\ (\text{List}\ a) \to \text{List}\ a$.
    4. $filter\ p :: \text{List}\ a \to \text{List}\ a$, where $p :: a \to \text{Bool}$.
    5. $takeWhile\ p :: \text{List}\ a \to \text{List}\ a$, where $p :: a \to \text{Bool}$.
    6. $id :: \text{List}\ a \to \text{List}\ a$.

    In case you haven't seen them, *all p xs* is True iff. all elements in *xs* satisfy *p*, and *elem z xs* is True iff. *x* is a member of *xs*.

    Solution:

    1. $all\ p = foldr\ (\lambda x\ b \to p\ x \wedge b)\ \text{True}$.
    2. $elem\ x = foldr\ (\lambda y\ b \to x == y \vee b)\ \text{False}$,
    3. $concat = foldr\ (+\!\!+)\ [\,]$.
    4. $filter\ p = foldr\ (\lambda x\ xs \to \textbf{if}\ p\ x\ \textbf{then}\ x : xs\ \textbf{else}\ xs)\ [\,]$,
    5. $takeWhile\ p = foldr\ (\lambda x\ xs \to \textbf{if}\ p\ x\ \textbf{then}\ x : xs\ \textbf{else}\ [\,])\ [\,]$,
    6. $id = foldr\ (:)\ [\,]$.

2. Given $p :: a \to \text{Bool}$, can $dropWhile\ p :: \text{List}\ a \to \text{List}\ a$ be written as a *foldr*?

Solution: No. Consider *dropWhile even* $[5, 4, 2, 1]$, which ought to be $[5, 4, 1, 1]$. Meanwhile, *dropWhile even* $[4, 2, 1] = [1]$, and the lost elements cannot be recovered.

3. Express the following functions by *foldr*:

   1. $\mathit{inits} :: \mathsf{List}\ a \rightarrow \mathsf{List}\ (\mathsf{List}\ a)$.
   2. $\mathit{tails} :: \mathsf{List}\ a \rightarrow \mathsf{List}\ (\mathsf{List}\ a)$.
   3. $\mathit{perms} :: \mathsf{List}\ a \rightarrow \mathsf{List}\ (\mathsf{List}\ a)$.
   4. $\mathit{sublists} :: \mathsf{List}\ a \rightarrow \mathsf{List}\ (\mathsf{List}\ a)$.
   5. $\mathit{splits} :: \mathsf{List}\ a \rightarrow \mathsf{List}\ (\mathsf{List}\ a, \mathsf{List}\ a)$.

Solution:

1. $\mathit{inits} = \mathit{foldr}\ (\lambda x\ \mathit{xss} \rightarrow [\,] : \mathit{map}\ (x{:})\ \mathit{xss})\ [[\,]]$.

2. $\mathit{tails} = \mathit{foldr}\ (\lambda x\ \mathit{xss} \rightarrow (x : \mathit{head}\ \mathit{xss}) : \mathit{xss})\ [[\,]]$,

3. $\mathit{perms} = \mathit{foldr}\ (\lambda x\ \mathit{xss} \rightarrow \mathit{concat}\ (\mathit{map}\ (\mathit{fan}\ x)\ \mathit{xss}))\ [[\,]]$

4. $\mathit{sublists} = \mathit{foldr}\ (\lambda x\ \mathit{xss} \rightarrow \mathit{xss} \mathbin{+\!\!+} \mathit{map}\ (x{:})\ \mathit{xss})\ [[\,]]$

5. *splits* can be defined by:

$$
\begin{aligned}
&\mathit{splits} = \mathit{foldr}\ \mathit{spl}\ [([\,],[\,])]\ , \\
&\quad \textbf{where}\ \mathit{spl}\ x\ ((\mathit{xs},\mathit{ys}) : \mathit{zss}) = \\
&\qquad\qquad ([\,], x : \mathit{xs} \mathbin{+\!\!+} \mathit{ys}) : \mathit{map}\ ((x{:}) \times \mathit{id})\ ((\mathit{xs},\mathit{ys}) : \mathit{zss})\ .
\end{aligned}
$$

   where $(f \times g)\ (x, y) = (f\ x, g\ y)$.

4. Prove the *foldr*-fusion theorem. To recite the theorem: given $f :: a \rightarrow b \rightarrow b$, $e :: b$, $h :: b \rightarrow c$ and $g :: a \rightarrow c \rightarrow c$, we have

$$h \cdot \mathit{foldr}\ f\ e = \mathit{foldr}\ g\ (h\ e)\ ,$$

   if $h\ (f\ x\ y) = g\ x\ (h\ y)$ for all x and y.

Solution: The aim is to prove that $h\ (\mathit{foldr}\ f\ e\ \mathit{xs}) = \mathit{foldr}\ g\ (h\ e)\ \mathit{xs}$ for all *xs*, assuming that $h\ (f\ x\ y) = g\ x\ (h\ y)$.

Case $\mathit{xs} := [\,]$:

$$
\begin{array}{ll}
& h\ (foldr\ f\ e\ [\,])\\
= & h\ e\\
= & foldr\ g\ (h\ e)\ [\,]\ \ .
\end{array}
$$

Case $xs := x : xs$:

$$
\begin{array}{ll}
& h\ (foldr\ f\ e\ (x : xs))\\
= & \quad \{\ \text{definition of } foldr\ \}\\
& h\ (f\ x\ (foldr\ f\ e\ xs))\\
= & \quad \{\ \text{fusion condition: } h\ (f\ x\ y) = g\ x\ (h\ y)\ \}\\
& g\ x\ (h\ (foldr\ f\ e\ xs))\\
= & \quad \{\ \text{induction}\ \}\\
& g\ x\ (foldr\ g\ (h\ e)\ xs)\\
= & \quad \{\ \text{definition of } foldr\ \}\\
& foldr\ g\ (h\ e)\ (x : xs)\ \ .
\end{array}
$$

5. Prove the *map*-fusion rule $map\ f \cdot map\ g = map\ (f \cdot g)$ by *foldr*-fusion.

Solution: Since $map\ g$ is a *foldr*, we proceed as follows:

$$
\begin{array}{ll}
& map\ f \cdot map\ g\\
= & \quad \{\ map\ g \text{ is a } foldr\ \}\\
& map\ f \cdot foldr\ (\lambda x\ ys \rightarrow g\ x : ys)\ [\,]\\
= & \quad \{\ foldr\text{-fusion}\ \}\\
& foldr\ (\lambda x\ ys \rightarrow f\ (g\ x) : ys)\ [\,]\\
= & \quad \{\ \text{definition of } map \text{ as a } foldr\ \}\\
& map\ (f \cdot g)\ \ .
\end{array}
$$

The fusion condition is proved below:

$$
\begin{array}{ll}
& map\ f\ (g\ x : ys)\\
= & \quad \{\ \text{definition } map\ \}\\
& f\ (g\ x) : map\ f\ ys\ \ .
\end{array}
$$

6. Prove that $sum \cdot concat = sum \cdot map\ sum$ by *foldr*-fusion, twice. Compare the proof with you previous proof in earlier parts of this course.

Solution:

$$\begin{aligned}
&\quad \mathit{sum} \cdot \mathit{concat}\\
&= \mathit{sum} \cdot \mathit{foldr}\ (+\!\!+)\ [\,]\\
&= \quad \{\ \mathit{foldr}\text{-fusion}\ \}\\
&\quad \mathit{foldr}\ (\lambda \mathit{xs}\ n \to \mathit{sum}\ \mathit{xs} + n)\ 0\\
&= \quad \{\ \mathit{foldr}\text{-}\mathit{map}\ \text{fusion, see Exercise 7}\ \}\\
&\quad \mathit{foldr}\ (+)\ 0 \cdot \mathit{map}\ \mathit{sum}\\
&= \mathit{sum} \cdot \mathit{map}\ \mathit{sum}\ \ .
\end{aligned}$$

Fusion conditions for the *foldr*-fusion is

$$\mathit{sum}\ (\mathit{xs} +\!\!+ \mathit{ys}) = \mathit{sum}\ \mathit{xs} + \mathit{sum}\ \mathit{ys}\ \ ,$$

which is the key property we needed in the early part of this term to prove the same property. We have proved the property before, by induction on *xs*. We omit the proof here. (Note that we can also prove it by two more *foldr*-fusion, noting that $(+\!\!+\mathit{ys})$ is a *foldr*, and so is *sum*.)

See Exercise 7 for *foldr*-*map* fusion. The penultimate equality holds because $(+) \cdot \mathit{sum} = (\lambda \mathit{xs}\ n \to \mathit{sum}\ \mathit{xs} + n)$. Instead of *foldr*-*map* fusion we cal also use *foldr* fusion alone. The fusion condition is $\mathit{sum}\ (\mathit{sum}\ \mathit{xs} : \mathit{xss}) = \mathit{sum}\ \mathit{xs} + \mathit{sum}\ \mathit{xss}$.

The *foldr*-fusion theorem captures the common pattern in these proofs. We only need to fill in the problem-dependent proofs.

7. The *map* fusion theorem is an instance of the *foldr*-*map* fusion theorem: $\mathit{foldr}\ f\ e \cdot \mathit{map}\ g = \mathit{foldr}\ (f \cdot g)\ e$.

   (a) Prove the theorem.

Solution: Since *map g* is a *foldr*, we proceed as follows:

$$\begin{aligned}
&\quad \mathit{foldr}\ f\ e \cdot \mathit{map}\ g\\
&= \quad \{\ \mathit{map}\ g\ \text{is a}\ \mathit{foldr}\ \}\\
&\quad \mathit{foldr}\ f\ e \cdot \mathit{foldr}\ (\lambda x\ \mathit{ys} \to g\ x : \mathit{ys})\ [\,]\\
&= \quad \{\ \mathit{foldr}\text{-fusion}\ \}\\
&\quad \mathit{foldr}\ (f \cdot g)\ (\mathit{foldr}\ f\ e\ [\,])\\
&= \quad \{\ \text{definition of}\ \mathit{foldr}\ \}\\
&\quad \mathit{foldr}\ (f \cdot g)\ e\ \ .
\end{aligned}$$

The fusion condition is proved below:

$$\begin{aligned}
&\quad \mathit{foldr}\ f\ e\ (g\ x : \mathit{ys})\\
&= \quad \{\ \text{definition}\ \mathit{foldr}\ \}\\
&\quad f\ (g\ x)\ (\mathit{foldr}\ f\ e\ \mathit{ys})\ \ .
\end{aligned}$$

(b) Express $sum \cdot map\ (2\times)$ as a *foldr*.

Solution:

$$
\begin{aligned}
& sum \cdot map\ (2\times) \\
=\ & foldr\ (+)\ 0 \cdot map\ (2\times) \\
=\ & \quad \{\ foldr\text{-}map \text{ fusion } \} \\
& foldr\ ((+) \cdot (2\times))\ 0 \quad .
\end{aligned}
$$

(c) Show that $(2\times) \cdot sum$ reduces to the same *foldr* as the one above.

Solution:

$$
\begin{aligned}
& (2\times) \cdot sum \\
=\ & (2\times) \cdot foldr\ (+)\ 0 \\
=\ & \quad \{\ foldr \text{ fusion } \} \\
& foldr\ ((+) \cdot (2\times))\ 0 \quad .
\end{aligned}
$$

The fusion condition is

$$
\begin{aligned}
& 2 \times (x + y) \\
=\ & \quad \{ \text{ distributivity } \} \\
& 2 \times x + 2 \times y \\
=\ & \quad \{ \text{ definition of } (\cdot) \} \\
& ((+) \cdot (2\times))\ x\ (2 \times y) \quad .
\end{aligned}
$$

8. Prove that $map\ f\ (xs \mathbin{+\!\!+} ys) = map\ f\ xs \mathbin{+\!\!+} map\ f\ ys$ by *foldr*-fusion. **Hint**: this is equivalent to $map\ f \cdot (\mathbin{+\!\!+} ys) = (\mathbin{+\!\!+} map\ f\ ys) \cdot map\ f$. You may need to do (any kinds of) fusion twice.

Solution: Recall that $(\mathbin{+\!\!+} ys)$ is a *foldr*. Use *foldr* fusion and *foldr*-*map* fusion:

$$
\begin{aligned}
& (\mathbin{+\!\!+} map\ f\ ys) \cdot map\ f \\
=\ & \quad \{\ foldr\text{-}map \text{ fusion } \} \\
& foldr\ ((:) \cdot f)\ (map\ f\ ys) \\
=\ & \quad \{\ foldr \text{ fusion } \} \\
& map\ f \cdot (\mathbin{+\!\!+} ys) \quad .
\end{aligned}
$$

The fusion condition of the last step is:

$$
\begin{aligned}
& map\ f\ (x : zs) \\
=\ & \quad \{ \text{ definition of } map \ \}
\end{aligned}
$$

$$
\begin{aligned}
& f\ x : \mathit{map}\ f\ \mathit{zs} \\
= & \quad \{\ \text{definition of } (\cdot)\ \} \\
& ((:) \cdot f)\ x\ (\mathit{map}\ f\ \mathit{zs}) \quad .
\end{aligned}
$$

9. Prove that *length* · *concat* = *sum* · *map length* by fusion.

Solution: We caculate

$$
\begin{aligned}
& \mathit{length} \cdot \mathit{concat} \\
= & \ \mathit{length} \cdot \mathit{foldr}\ (+\!\!+)\ [\,] \\
= & \quad \{\ \mathit{foldr}\text{-fusion}\ \} \\
& \mathit{foldr}\ ((+) \cdot \mathit{length})\ 0 \\
= & \ \{ \mid \mathit{sum} = \mathit{foldr}\ (+)\ 0 \mid, \mid \mathit{foldr} \mid - \mid \mathit{map} \mid \mathit{fusion} \} \\
& \mathit{sum} \cdot \mathit{map}\ \mathit{length} \quad .
\end{aligned}
$$

The fusion condition is proved below:

$$
\begin{aligned}
& \mathit{length}\ (\mathit{xs} +\!\!+ \mathit{ys}) \\
= & \quad \{\ (+\!\!+) \text{ and } (+) \text{ homorphic}\ \} \\
& \mathit{length}\ \mathit{xs} + \mathit{length}\ \mathit{ys} \\
= & \quad \{\ \text{definition of } (\cdot)\ \} \\
& ((+) \cdot \mathit{length})\ \mathit{xs}\ (\mathit{length}\ \mathit{ys}) \quad .
\end{aligned}
$$

10. Let *scanr f e* = *map* (*foldr f e*) · *tails*. Construct, by *foldr*-fusion, an implementation of *scanr* whose number of calls to *f* is proportional to the length of the input list.

Solution: Recall that *tails* is a *foldr*:

$$\mathit{tails} = \mathit{foldr}\ (\lambda x\ \mathit{xss} \to (x : \mathit{head}\ \mathit{xss}) : \mathit{xss})\ [[\,]] \quad ,$$

We try to fuse *map* (*foldr f e*) into *tails*. For the base value, notice that

$$\mathit{map}\ (\mathit{foldr}\ f\ e)\ [[\,]] = [e] \quad .$$

To construct the step function, we work on the fusion condition:

$$
\begin{aligned}
& \mathit{map}\ (\mathit{foldr}\ f\ e)\ ((x : \mathit{head}\ \mathit{xss}) : \mathit{xss}) \\
= & \quad \{\ \text{definition of } \mathit{map}\ \} \\
& \mathit{foldr}\ f\ e\ (x : \mathit{head}\ \mathit{xss}) : \mathit{map}\ (\mathit{foldr}\ f\ e)\ \mathit{xss}
\end{aligned}
$$

$$
\begin{aligned}
&= \quad \{ \text{definition of } \mathit{foldr} \} \\
&\quad f\ x\ (\mathit{foldr}\ f\ e\ (\mathit{head}\ \mathit{xss})) : \mathit{map}\ (\mathit{foldr}\ f\ e)\ \mathit{xss} \\
&= \quad \{ \mathit{foldr}\ f\ e\ (\mathit{head}\ \mathit{xss}) = \mathit{head}\ (\mathit{map}\ (\mathit{foldr}\ f\ e)\ \mathit{xss}) \} \\
&\quad \textbf{let}\ \mathit{ys} = \mathit{map}\ (\mathit{foldr}\ f\ e)\ \mathit{xss} \\
&\quad \textbf{in}\ f\ x\ (\mathit{head}\ \mathit{ys}) : \mathit{ys} \quad .
\end{aligned}
$$

We have therefore constructed:

$$\mathit{scanr}\ f\ e = \mathit{foldr}\ (\lambda x\ \mathit{ys} \to f\ x\ (\mathit{head}\ \mathit{ys}) : \mathit{ys})\ [e] \quad .$$

You may find the inductive definition easier to comprehend:

$$
\begin{aligned}
&\mathit{scanr}\ f\ e\ [\,] &&= [e] \\
&\mathit{scanr}\ f\ e\ (x : \mathit{xs}) &&= f\ x\ (\mathit{head}\ \mathit{ys}) : \mathit{ys} \quad , \\
&\quad \textbf{where}\ \mathit{ys} = \mathit{scanr}\ f\ e\ \mathit{xs} \quad .
\end{aligned}
$$

11. Recall the function *binary* :: Nat $\to$ [Nat] that returns the *reversed* binary representation of a natural number, for example *binary* $4 = [0,0,1]$. Also, we talked about a function *decimal* :: [Nat] $\to$ Nat that converts the representation back to a natural number.

(a) This time, express *decimal* using a *foldr*.

Solution:

$$\mathit{decimal} = \mathit{foldr}\ (\lambda d\ n \to d + 2 \times n)\ 0 \quad .$$

(b) Recall the function $\mathit{exp}\ m\ n = m^n$. Use *foldr*-fusion to construct *step* and *base* such that

$$\mathit{exp}\ m \cdot \mathit{decimal} = \mathit{foldr}\ \mathit{step}\ \mathit{base} \quad .$$

If the fusion succeeds, we have derived a hylomorphism computing m^n:

$$\mathit{fastexp}\ m = \mathit{foldr}\ \mathit{step}\ \mathit{base} \cdot \mathit{binary} \quad .$$

Solution: For the base value, we have $\mathit{base} = \mathit{exp}\ m\ 0 = 1$.
For the step function, we calculate

$$
\begin{aligned}
&\quad \mathit{exp}\ m\ (d + 2 \times n) \\
&= \quad \{ \text{since } m^{x+y} = m^x \times m^y \} \\
&\quad \mathit{exp}\ m\ d \times \mathit{exp}\ m\ (2 \times n)
\end{aligned}
$$

$$
\begin{aligned}
&= \quad \{\ \text{since } m^{2n} = (m^n)^2\text{, let } \mathit{square}\ x = x \times x\ \} \\
&\quad\ \mathit{exp}\ m\ d \times \mathit{square}\ (\mathit{exp}\ m\ n) \\
&= \quad \{\ d \text{ is either 0 or 1. Expand the definition}\ \} \\
&\quad\ \textbf{if}\ d == 0\ \textbf{then}\ \mathit{square}\ (\mathit{exp}\ m\ n)\ \textbf{else}\ m \times \mathit{square}\ (\mathit{exp}\ m\ n) \enspace .
\end{aligned}
$$

Therefore we conclude

$$
\begin{aligned}
\mathit{exp}\ m \cdot \mathit{decimal} = \mathit{foldr}\ (\lambda d\ x \rightarrow \textbf{if}\ d == 0\ &\textbf{then}\ \mathit{square}\ x \\
&\textbf{else}\ m \times \mathit{square}\ x)\ 1 \enspace .
\end{aligned}
$$

12. Express $\mathit{reverse} :: \mathsf{List}\ a \rightarrow \mathsf{List}\ a$ by a foldr. Let $\mathit{revcat} = (+\!\!+) \cdot \mathit{reverse}$. Express revcat as a foldr.

Solution: $\mathit{reverse} = \mathit{foldr}\ (\lambda x\ \mathit{xs} \rightarrow \mathit{xs} +\!\!+ [x])\ [\,]$.

To fuse $(+\!\!+)$ into $\mathit{reverse}$, the base value is $(+\!\!+)\ [\,] = \mathit{id}$. To construct the step function, we try to meet the fusion condition:

$$
(+\!\!+)\ ((\lambda x\ \mathit{xs} \rightarrow \mathit{xs} +\!\!+ [x])\ x\ \mathit{xs}) = \mathit{step}\ x\ ((+\!\!+)\ \mathit{xs}) \enspace .
$$

If we calculate:

$$
\begin{aligned}
&(+\!\!+)\ ((\lambda x\ \mathit{xs} \rightarrow \mathit{xs} +\!\!+ [x])\ x\ \mathit{xs}) \\
&\ = (+\!\!+)\ (\mathit{xs} +\!\!+ [x]) \enspace ,
\end{aligned}
$$

it is hard to figure out how to proceed, since $(+\!\!+)$ expects another argument. It is easier to calculate if we supply it another argument ys. We restart and calculate:

$$
\begin{aligned}
&(+\!\!+)\ ((\lambda x\ \mathit{xs} \rightarrow \mathit{xs} +\!\!+ [x])\ x\ \mathit{xs})\ \mathit{ys} \\
&\ = (+\!\!+)\ (\mathit{xs} +\!\!+ [x])\ \mathit{ys} \\
&\ = (\mathit{xs} +\!\!+ [x]) +\!\!+ \mathit{ys} \\
&\ = \quad \{\ (+\!\!+) \text{ associative}\ \} \\
&\quad\ \mathit{xs} +\!\!+ ([x] +\!\!+ \mathit{ys}) \\
&\ = \quad \{\ \text{definition of } (\cdot)\ \} \\
&\quad\ (((+\!\!+)\ \mathit{xs}) \cdot (x{:}))\ \mathit{ys} \\
&\ = \quad \{\ \text{factor out } x\text{, } ((+\!\!+)\ \mathit{xs})\text{, and } \mathit{ys}\ \} \\
&\quad\ (\lambda x\ f \rightarrow f \cdot (x{:}))\ x\ ((+\!\!+)\ \mathit{xs})\ \mathit{ys} \enspace .
\end{aligned}
$$

We conclude that

$$
\mathit{revcat} = \mathit{foldr}\ (\lambda x\ f \rightarrow f \cdot (x{:}))\ \mathit{id} \enspace .
$$

13. Fold on natural numbers.

    (a) The predicate $even :: \mathrm{Nat} \to \mathrm{Bool}$ yields True iff. the input is an even number. Define *even* in terms of *foldN*.

    Solution:

    $$even = foldN\ not\ \mathrm{True}\ \ .$$

    (b) Express the identity function on natural numbers $id\ n = n$ in terms of *foldN*.

    Solution:

    $$id = foldN\ \mathbf{1}_{+}\ 0\ \ .$$

14. Fuse *even* into $(+n)$. This way we get a function that checks whether a natural number is even after adding n.

    Solution: Recall that $(+n) = foldN\ \mathbf{1}_{+}\ n$. To fuse $even \cdot (+n)$ into one *foldN*, the base value is $even\ n$. To find out the step function, recall that $even\ (\mathbf{1}_{+}\ n) = not\ (even\ n)$. We may then conclude:

    $$even \cdot (+n) = foldN\ not\ (even\ n)\ \ .$$

15. The famous Fibonacci number is defined by:

    $$\begin{aligned} fib\ 0 &= 0 \\ fib\ 1 &= 1 \\ fib\ (2+n) &= fib\ (1+n) + fib\ n\ \ . \end{aligned}$$

    The definition above, when taken directly as an algorithm, is rather slow. Define $fib2\ n = (fib\ (1+n), fib\ n)$. Derive an $O(n)$ implementation of *fib2* by fusing it with $id :: \mathrm{Nat} \to \mathrm{Nat}$.

    Solution: Recall that $id = foldN\ (\mathbf{1}_{+})\ 0$. Fusing *fib2* into *id*, the base value is $fib2\ 0 = (1, 0)$. To construct the step function we calculate

    $$\begin{aligned} & fib2\ (\mathbf{1}_{+}\ n) \\ =\ & (fib\ (\mathbf{1}_{+}\ (\mathbf{1}_{+}\ n)), fib\ (\mathbf{1}_{+}\ n)) \\ =\ & \quad \{\ \text{definition of } fib\ \} \end{aligned}$$

$$
\begin{aligned}
&(\mathit{fib}\ (\mathbf{1}_{+}\ n) + \mathit{fib}\ n, \mathit{fib}\ (\mathbf{1}_{+}\ n)) \\
={}& (\lambda(x, y) \to (x + y, x))\ (\mathit{fib2}\ n) \ \ .
\end{aligned}
$$

Therefore we conclude that

$$
\mathit{fib2} = \mathit{foldN}\ (\lambda(x, y) \to (x + y, x))\ (1, 0) \ \ .
$$

16. What are the fold fusion theorems for ETree and ITree?

Solution:

$$
\begin{aligned}
h \cdot \mathit{foldIT}\ f\ \ e = \mathit{foldIT}\ g\ (h\ e) &\ \Leftarrow\ h\ (f\ \ x\ \ y\ \ z) = g\ x\ (h\ y)\ (h\ z) \ \ , \\
h \cdot \mathit{foldET}\ f\ \ k = \mathit{foldET}\ g\ (h \cdot k) &\ \Leftarrow\ h\ (f\ \ x\ \ y) = g\ (h\ x)\ (h\ y) \ \ .
\end{aligned}
$$