Functional Program Derivation
Exercises for Day 2

Shin-Cheng Mu   Shu-Chun Weng (TA)

2007 Formosan Summer School on
Logic, Language, and Computation
July 5th, 2007

This exam sheet is worth 20 points in total.

2.1 Folds and Fold-Fusion

1. **(1 points)** The function $\mathit{filter}\ p$ selects from a list all elements satisfying a predicate p. For example, $\mathit{filter}\ \mathit{even}\ [1,2,3,4] = [2,4]$.

   (a) Give a recursive definition of filter:

   $$\begin{array}{lcl} \mathit{filter}\ p\ [\,] & = & \ldots \\ \mathit{filter}\ p\ (x{:}xs) & = & \ldots \end{array}$$

   (b) Define $\mathit{filter}\ p$ in terms of foldr.

2. **(2 points)** Prove, by fold-fusion, that

   $$\mathit{filter}\ p \cdot \mathit{map}\ f \quad = \quad \mathit{map}\ f \cdot \mathit{filter}\ (p \cdot f).$$

   Hint: apply fold-fusion on both sides, and show that they are equal to the same fold.

3. **(3 points)** Given functions $f :: \alpha \to \beta$ and $g :: \alpha \to \gamma$, $\langle f, g \rangle :: \alpha \to (\beta, \gamma)$ is a function defined by:

   $$\langle f, g \rangle\ a = (f\ a, g\ a).$$

   Recall the definition of steep and sum. The definition of $\mathit{steepsum}$ can be re-written as:

   $$\mathit{steepsum} \quad = \quad \langle \mathit{steep}, \mathit{sum} \rangle.$$

   Also recall that the identity function id on lists is a fold: $\mathit{id} = \mathit{foldr}\ (:)\ [\,]$. Use the fold-fusion theorem to fuse $\mathit{steepsum} \cdot \mathit{id}$ into one fold.

4. **(4 points)** Recall the definition of *scanr* from the lecture:

$$scanr\ f\ e \quad = \quad map\ (foldr\ f\ e) \cdot tails$$

and its implementation as a fold:

$$\begin{array}{lcl} scanr\ f\ e & = & foldr\ (sc\ f)\ [e] \\ sc\ f\ x\ (y{:}ys) & = & f\ x\ y : y : ys \end{array}$$

   (a) Expand $scanr\ (+)\ 0\ [1, 2, 3]$ step by step:

$$\begin{array}{cl} & scanr\ (+)\ 0\ [1, 2, 3] \\ = & foldr\ (sc\ (+))\ [0]\ [1, 2, 3] \\ = & \ldots \end{array}$$

   (b) Derive the implementation of $scanr\ f\ e$ by fusing $map\ (foldr\ f\ e) \cdot tails$ into one fold.

5. **(4 points)** Given two functions h_1 and h_2, the function $\langle h_1, h_2 \rangle$ (pronounced "split of h_1 and h_2") computes the pair of their results:

$$\langle h_1, h_2 \rangle\ xs \quad = \quad (h_1\ xs, h_2\ xs).$$

In the special case when both h_1 and h_2 are defined by *foldr*:

$$\begin{array}{lcl} h_1 & = & foldr\ f_1\ e_1, \\ h_2 & = & foldr\ f_2\ e_2, \end{array}$$

the following "banana-split" rule allows us to express $\langle h_1, h_2 \rangle$ using one single *foldr*:

$$\begin{array}{lcl} \langle h_1, h_2 \rangle & = & foldr\ g\ (e_1, e_2), \\ g\ x\ (y, z) & = & (f_1\ x\ y, f_2\ x\ z). \end{array}$$

It optimises two traversal through the list to only one traversal. It is called "banana-split" because folds used to be written using a notation called "banana brackets".

   (a) The function $\langle sum, length \rangle$ return the pair of sum and length of the input list. Use the banana-split rule to express $\langle sum, length \rangle$ by a fold.

   (b) Prove the banana-split rule by fold fusion. Hint: $\langle h_1, h_2 \rangle = \langle h_1, h_2 \rangle \cdot id$, and id is a fold.

2.2 Unfolds and Hylomorphism

1. **(2 points)** Let $\mathit{hyloeT}\ f\ g\ p\ h\ k = \mathit{foldeT}\ f\ g \cdot \mathit{unfoldeT}\ p\ h\ k$.

   (a) Express *msort* using hyloeT.

   (b) Given a recursive definition of hyloeT, like that of hyloiT in the lecture.

2. **(4 points)**

   (a) The function $\mathit{indexFrom} :: (N, [\alpha]) \rightarrow [(N, \alpha)]$ assigns an index to each element in the give list. E.g.

   $$\mathit{indexFrom}\,(0, [a, b, c]) \quad = \quad [(0, a), (1, b), (2, c)].$$

   Define *indexFrom* by *unfoldr*. Hint: the answer may probably look like:

   $$\begin{array}{lcl} \mathit{indexFrom} & = & \mathit{unfoldr}\ p\ \mathit{idn}, \\ p\,(?, ?) & = & \ldots \\ \mathit{idn}\,(n, x{:}xs) & = & \ldots \end{array}$$

   (b) The function call $\mathit{lsearch}\ x\ xs$ performs a linear search for x in the list xs and returns its index. If the x is not in xs, it returns -1. E.g.

   $$\begin{array}{lcl} \mathit{lsearch}\ b\,[(0, a), (1, b), (2, c)] & = & 1, \\ \mathit{lsearch}\ d\,[(0, a), (1, b), (2, c)] & = & -1. \end{array}$$

   Define *lsearch* by a *foldr*.

   If you are able to complete (a) and (b), you have constructed, as a hylomorphism, a function $\mathit{posx} = \mathit{lsearchx} \cdot \mathit{indexFrom}$ searching for the position of x in xs.