Folds and unfolds all around us

Conal Elliott

Tabula

Spring, 2013

Preliminaries

This talk is a literate Haskell program.

$$\textbf{module}\ \mathit{FoldsAndUnfolds}\ \textbf{where}$$

I'll use some non-standard (for Haskell) type notation:

$$\begin{aligned}
&\textbf{type}\ \mathbf{1}\ \ \ \ = ()\\
&\textbf{type}\ (+) = \mathit{Either}\\
&\textbf{type}\ (\times) = (,)
\end{aligned}$$

$$\begin{aligned}
&\textbf{infixl}\ 7\ \times\\
&\textbf{infixl}\ 6\ +
\end{aligned}$$

Recursive functional programming

On numbers:

$$
\begin{aligned}
&\mathit{fact}_0\ 0 = 1 \\
&\mathit{fact}_0\ n = n \times \mathit{fact}_0\ (n-1)
\end{aligned}
$$

On lists:

$$\mathbf{data}\ [\,a\,] = [\,] \mid a : [\,a\,]$$

$$
\begin{aligned}
&\mathit{product}_L :: [\,\mathit{Integer}\,] \to \mathit{Integer} \\
&\mathit{product}_L\ [\,] \qquad\quad = 1 \\
&\mathit{product}_L\ (a : as) = a \times \mathit{product}_L\ as
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{range}_L :: \mathit{Integer} \to \mathit{Integer} \to [\,\mathit{Integer}\,] \\
&\mathit{range}_L\ l\ h \mid l > h \qquad\quad = [\,] \\
&\qquad\qquad\ \mid \mathit{otherwise} = l : \mathit{range}_L\ (\mathit{succ}\ l)\ h
\end{aligned}
$$

Recursive functional programming

On (binary leaf) trees:

$$\textbf{data}\ T\ a = L\ a \mid B\ (T\ a)\ (T\ a)\ \textbf{deriving}\ \mathit{Show}$$

$$\begin{aligned}
&\mathit{product}_T :: T\ \mathit{Integer} \to \mathit{Integer} \\
&\mathit{product}_T\ (L\ a) \quad = a \\
&\mathit{product}_T\ (B\ s\ t) = \mathit{product}_T\ s \times \mathit{product}_T\ t
\end{aligned}$$

$$\begin{aligned}
&\mathit{range}_T :: \mathit{Integer} \to \mathit{Integer} \to T\ \mathit{Integer} \\
&\mathit{range}_T\ l\ h \mid l \equiv h \qquad\quad = L\ l \\
&\qquad\qquad\ \ \mid \mathit{otherwise} = B\ (\mathit{range}_T\ l\ m)\ (\mathit{range}_T\ (m+1)\ h) \\
&\quad \textbf{where}\ m = (l+h)\ `\mathit{div}`\ 2
\end{aligned}$$

Recursive functional programming?

Structured functional programming

> *… recursive equations are the "assembly language" of functional programming, and direct recursion the* `goto`*.*

Jeremy Gibbons, *Origami programming*

A structured alternative:

- identify commonly useful patterns,
- determine their properties, and
- apply the patterns and properties.

Folds ("catamorphisms")

Contract a structure *down to* a single value.

For lists:

$$
\begin{aligned}
&fold_L :: (a \to b \to b) \to b \to ([\,a\,] \to b) \\
&fold_L \ \_ \ b \ [\,] \qquad\quad = b \\
&fold_L \ f \ b \ (a : as) = f \ a \ (fold_L \ f \ b \ as)
\end{aligned}
$$

$$
\begin{aligned}
sum_L \quad &= fold_L \ (+) \ 0 \\
product_L &= fold_L \ (\times) \ 1 \\
reverse_L &= fold_L \ (\lambda a \ r \to r +\!\!+ [\,a\,]) \ [\,]
\end{aligned}
$$

For trees:

$$
\begin{aligned}
&fold_T :: (b \to b \to b) \to (a \to b) \to (T \ a \to b) \\
&fold_T \ \_ \ l \ (L \ a) \quad = l \ a \\
&fold_T \ b \ l \ (B \ s \ t) = b \ (fold_T \ b \ l \ s) \ (fold_T \ b \ l \ t)
\end{aligned}
$$

$$
product_T = fold_T \ (\times) \ id
$$

Unfolds ("anamorphisms")

Expand a structure *up from* a single value.

Lists:

$$
\begin{array}{l}
unfold_L :: (b \to Maybe\ (a \times b)) \to (b \to [a]) \\
unfold_L\ f\ b = \textbf{case}\ f\ b\ \textbf{of} \\
\quad Just\ (a, b') \to a : unfold_L\ f\ b' \\
\quad Nothing \quad\;\; \to [\,]
\end{array}
$$

$$
\begin{array}{l}
rangeL' :: Integer \times Integer \to [Integer] \\
rangeL' = unfold_L\ g \\
\quad \textbf{where} \\
\qquad g\ (l, h) \mid l > h \qquad\quad = Nothing \\
\qquad\qquad\quad\;\; \mid otherwise = Just\ (l, (succ\ l, h))
\end{array}
$$

Unfolds ("anamorphisms")

Trees:

$$
\begin{array}{l}
\mathit{unfold}_T :: (b \to a + b \times b) \to (b \to T\ a) \\
\mathit{unfold}_T\ g\ x = \mathbf{case}\ g\ x\ \mathbf{of} \\
\quad \mathit{Left}\ a \qquad\quad \to L\ a \\
\quad \mathit{Right}\ (c, d) \to B\ (\mathit{unfold}_T\ g\ c)\ (\mathit{unfold}_T\ g\ d)
\end{array}
$$

$$
\begin{array}{l}
\mathit{range}_{TP} :: \mathit{Integer} \times \mathit{Integer} \to T\ \mathit{Integer} \\
\mathit{range}_{TP} = \mathit{unfold}_T\ g \\
\quad \mathbf{where} \\
\qquad g\ (l, h) \mid l \equiv h \qquad\quad = \mathit{Left}\ l \\
\qquad\qquad\qquad \mid \mathit{otherwise} = \mathit{Right}\ ((l, m), (m + 1, h)) \\
\qquad\quad \mathbf{where}\ m = (l + h)\ \text{`}\mathit{div}\text{`}\ 2
\end{array}
$$

Factorial again

Assembly language:

$$\begin{aligned} &\mathit{fact_0}\ 0 = 1 \\ &\mathit{fact_0}\ n = n \times \mathit{fact_0}\ (n - 1) \end{aligned}$$

You may have seen this Haskelly definition:

$$\mathit{fact_1}\ n = \mathit{product}\ [1 \mathinner{..} n]$$

Theme: replace control structures by data structures and standard combining forms.

Carry this theme further.

Combining *unfold* and *fold*

Equivalently,

$$fact_1 = product_L \circ range_L\ 1$$

Note: composition of unfold ($range_L$) and fold $product_L$.

More explicit:

$$\begin{array}{l} fact_2 = fold_L\ (\times)\ 1 \circ unfold_L\ g \\ \quad \textbf{where} \\ \qquad g\ 0 = Nothing \\ \qquad g\ n = Just\ (n, n-1) \end{array}$$

This combination of *unfold* and *fold* is called a "hylomorphism".

Fibonacci

Assembly language:

$$\begin{aligned}
\mathit{fib}_0\ 0 &= 0 \\
\mathit{fib}_0\ 1 &= 1 \\
\mathit{fib}_0\ n &= \mathit{fib}_0\ (n-1) + \mathit{fib}_0\ (n-2)
\end{aligned}$$

Via trees:

$$\begin{aligned}
&\mathit{fib}_T :: \mathit{Integer} \rightarrow T\ \mathit{Integer} \\
&\mathit{fib}_T\ 0 = L\ 0 \\
&\mathit{fib}_T\ 1 = L\ 1 \\
&\mathit{fib}_T\ n = B\ (\mathit{fib}_T\ (n-1))\ (\mathit{fib}_T\ (n-2))
\end{aligned}$$

$$\begin{aligned}
&\mathit{sum}_T :: T\ \mathit{Integer} \rightarrow \mathit{Integer} \\
&\mathit{sum}_T = \mathit{fold}_T\ (+)\ \mathit{id}
\end{aligned}$$

$$\begin{aligned}
&\mathit{fib}_1 :: \mathit{Integer} \rightarrow \mathit{Integer} \\
&\mathit{fib}_1 = \mathit{sum}_T \circ \mathit{fib}_T
\end{aligned}$$

Fibonacci

More explicitly hylomorphic:

$$unfold_T :: (b \to a + b \times b) \to (b \to T\ a)$$

$$\begin{aligned}
&fib_2 :: Integer \to Integer \\
&fib_2 = fold_T\ (+)\ id \circ unfold_T\ g \\
&\quad \textbf{where} \\
&\qquad g\ 0 = Left\ 0 \\
&\qquad g\ 1 = Left\ 1 \\
&\qquad g\ n = Right\ (n-1, n-2)
\end{aligned}$$

Generalizing folds and unfolds

Summary of *fold* and *unfold*:

$$
\begin{aligned}
fold_L &:: (a \to b \to b) \to b \to ([\,a\,] \to b) \\
unfold_L &:: (b \to Maybe\ (a \times b)) \to (b \to [\,a\,])
\end{aligned}
$$

$$
\begin{aligned}
fold_T &:: (b \to b \to b) \to (a \to b) \to (T\ a \to b) \\
unfold_T &:: (b \to a + b \times b) \to (b \to T\ a)
\end{aligned}
$$

Why the asymmetry?

Playing with type isomorphisms

$$
\begin{aligned}
\mathit{fold}_L :: (a \to b \to b) \to b \qquad\qquad &\to ([\,a\,] \to b) \\
\simeq (a \times b \to b) \to b \qquad\qquad &\to ([\,a\,] \to b) \\
\simeq (a \times b \to b) \to (\mathbf{1} \to b) &\to ([\,a\,] \to b) \\
\simeq (a \times b \to b) \times (\mathbf{1} \to b) &\to ([\,a\,] \to b) \\
\simeq ((a \times b + \mathbf{1}) \to b) \qquad &\to ([\,a\,] \to b) \\
\simeq (\mathit{Maybe}\ (a \times b) \to b) \quad &\to ([\,a\,] \to b)
\end{aligned}
$$

Why $\mathit{Maybe}\ (a \times b)$?

Because

$$
\begin{aligned}
[\,a\,] &\simeq \mathit{Maybe}\ (a \times (\mathit{Maybe}\ (a \times (\mathit{Maybe}\ (a \times (...)))))) \\
&\simeq \mathit{Fix}\ (\Lambda b \to \mathit{Maybe}\ (a \times b))
\end{aligned}
$$

Regularizing

Recall:

$$fold_L :: (a \to b \to b) \to b \to ([\,a\,] \to b)$$

A more standard interface:

$$\begin{aligned}
&fold_{LF} :: (Maybe\ (a \times b) \to b) \to ([\,a\,] \to b) \\
&fold_{LF}\ h = fold_L\ (curry\ (h \circ Just))\ (h\ Nothing)
\end{aligned}$$

Now the duality emerges:

$$\begin{aligned}
unfold_L &:: (b \to Maybe\ (a \times b)) \to (b \to [\,a\,]) \\
fold_{LF} &:: (Maybe\ (a \times b) \to b) \to ([\,a\,] \to b)
\end{aligned}$$

Similarly for tree fold and unfold.

List and tree *unfold* and *fold* – pictures

$$\begin{tikzcd} Maybe\,(a \times b) \\ b \arrow[u, "g"] \arrow[r, "unfold_L\,g"'] & {[a]} \end{tikzcd} \qquad \begin{tikzcd} & Maybe\,(a \times b) \arrow[d, "h"] \\ {[a]} \arrow[r, "fold_L\,h"'] & b \end{tikzcd}$$

$$\begin{tikzcd} a + b \times b \\ b \arrow[u, "g"'] \arrow[r, "unfold_T\,g"] & T\,a \end{tikzcd} \qquad \begin{tikzcd} & a + b \times b \arrow[d, "h"] \\ T\,a \arrow[r, "fold_T\,h"'] & b \end{tikzcd}$$

General regular algebraic data types – pictures

Build up from "base functor" F to fixpoint μF:

$$\begin{tikzcd} F\,b & \\ b \arrow[u, "g"] \arrow[r, "\mathit{unfold}\ g"'] & \mu F \end{tikzcd} \qquad \begin{tikzcd} & F\,b \arrow[d, "h"] \\ \mu F \arrow[r, "\mathit{fold}\ h"'] & b \end{tikzcd}$$

General regular algebraic data types – Haskell

Build up from "base functor" f:

$$\textbf{newtype}\ \mathit{Fix}\ f = \mathit{Roll}\ \{\mathit{unRoll} :: f\ (\mathit{Fix}\ f)\}$$

$$\begin{aligned}
&\mathit{fold} :: \mathit{Functor}\ f \Rightarrow (f\ b \to b) \to (\mathit{Fix}\ f \to b)\\
&\mathit{fold}\ h = h \circ \mathit{fmap}\ (\mathit{fold}\ h) \circ \mathit{unRoll}
\end{aligned}$$

$$\begin{aligned}
&\mathit{unfold} :: \mathit{Functor}\ f \Rightarrow (a \to f\ a) \to (a \to \mathit{Fix}\ f)\\
&\mathit{unfold}\ g = \mathit{Roll} \circ \mathit{fmap}\ (\mathit{unfold}\ g) \circ g
\end{aligned}$$

$$\begin{aligned}
&\mathit{hylo} :: \mathit{Functor}\ f \Rightarrow (f\ b \to b) \to (a \to f\ a) \to (a \to b)\\
&\mathit{hylo}\ h\ g = \mathit{fold}\ h \circ \mathit{unfold}\ g
\end{aligned}$$

Let's revisit our examples.

Factorial via list *hylo*

$$\textbf{data}\ \mathit{LF}\ a\ t = \mathit{NilF} \mid \mathit{ConsF}\ a\ t\ \textbf{deriving}\ \mathit{Functor}$$
$$\textbf{type}\ L'\ a = \mathit{Fix}\ (\mathit{LF}\ a)$$

$$\begin{array}{l}
\mathit{fact_3} :: \mathit{Integer} \to \mathit{Integer} \\
\mathit{fact_3} = \mathit{hylo}\ h\ g \\
\quad \textbf{where} \\
\qquad g :: \mathit{Integer} \to \mathit{LF}\ \mathit{Integer}\ \mathit{Integer} \\
\qquad g\ 0\ = \mathit{NilF} \\
\qquad g\ n = \mathit{ConsF}\ n\ (n-1) \\
\qquad h :: \mathit{LF}\ \mathit{Integer}\ \mathit{Integer} \to \mathit{Integer} \\
\qquad h\ \mathit{NilF} \qquad\qquad = 1 \\
\qquad h\ (\mathit{ConsF}\ n\ u) = n \times u
\end{array}$$

Fibonacci via tree *hylo*

$$\textbf{data}\ \mathit{TF}\ a\ t = \mathit{LF}\ a \mid \mathit{BF}\ t\ t\ \textbf{deriving}\ \mathit{Functor}$$
$$\textbf{type}\ T'\ a = \mathit{Fix}\ (\mathit{TF}\ a)$$

$$
\begin{aligned}
&\mathit{fib}_3 :: \mathit{Integer} \rightarrow \mathit{Integer}\\
&\mathit{fib}_3 = \mathit{hylo}\ h\ g\\
&\quad \textbf{where}\\
&\qquad g :: \mathit{Integer} \rightarrow \mathit{TF}\ \mathit{Integer}\ \mathit{Integer}\\
&\qquad g\ 0 = \mathit{LF}\ 0\\
&\qquad g\ 1 = \mathit{LF}\ 1\\
&\qquad g\ n = \mathit{BF}\ (n-1)\ (n-2)\\
&\qquad h :: \mathit{TF}\ \mathit{Integer}\ \mathit{Integer} \rightarrow \mathit{Integer}\\
&\qquad h\ (\mathit{LF}\ n) \quad\ \ = n\\
&\qquad h\ (\mathit{BF}\ u\ v) = u + v
\end{aligned}
$$

Factorial via tree hylo

$$\textbf{type}\ \mathit{Range} = \mathit{Integer} \times \mathit{Integer}$$

$$
\begin{array}{l}
\mathit{fact}_4 :: \mathit{Integer} \to \mathit{Integer} \\
\mathit{fact}_4\ n = \mathit{hylo}\ h\ g\ (1, n) \\
\quad \textbf{where} \\
\qquad g :: \mathit{Range} \to \mathit{TF}\ \mathit{Integer}\ \mathit{Range} \\
\qquad g\ (\mathit{lo}, \mathit{hi}) = \textbf{case}\ \mathit{lo}\ \text{`}\mathit{compare}\text{`}\ \mathit{hi}\ \textbf{of} \\
\qquad\qquad\qquad\qquad GT \to \mathit{LF}\ 1 \\
\qquad\qquad\qquad\qquad EQ \to \mathit{LF}\ \mathit{lo} \\
\qquad\qquad\qquad\qquad LT \to \textbf{let}\ \mathit{mid} = (\mathit{lo} + \mathit{hi})\ \text{`}\mathit{div}\text{`}\ 2\ \textbf{in} \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{BF}\ (\mathit{lo}, \mathit{mid})\ (\mathit{mid} + 1, \mathit{hi}) \\
\qquad h :: \mathit{TF}\ \mathit{Integer}\ \mathit{Integer} \to \mathit{Integer} \\
\qquad h\ (\mathit{LF}\ i) \qquad = i \\
\qquad h\ (\mathit{BF}\ u\ v) = u \times v
\end{array}
$$

Parallel-friendly!

$$
\begin{tikzcd}
F\,a \arrow[r, "F\,(unfold\;g)"] & F\,(\mu F) \arrow[d, "Roll"] \\
a \arrow[u, "g"] \arrow[r, dashed, "unfold\;g"'] & \mu F
\end{tikzcd}
\qquad
\begin{tikzcd}
F\,(\mu F) \arrow[r, "F\,(fold\;h)"] & F\,b \arrow[d, "h"] \\
\mu F \arrow[u, "unRoll"] \arrow[r, dashed, "fold\;h"'] & b
\end{tikzcd}
$$

$$\textbf{newtype}\; Fix\; f = Roll\; \{unRoll :: f\; (Fix\; f)\}$$

$$unfold :: Functor\; f \Rightarrow (a \to f\; a) \to (a \to Fix\; f)$$
$$unfold\; g = Roll \circ fmap\; (unfold\; g) \circ g$$

$$fold :: Functor\; f \Rightarrow (f\; b \to b) \to (Fix\; f \to b)$$
$$fold\; h = h \circ fmap\; (fold\; h) \circ unRoll$$

Another look and *hylo*

$$a \xrightarrow{hylo\ h\ g} b$$

Another look and *hylo*

$$a \xrightarrow{\mathit{unfold}\ g} \mu F \xrightarrow{\mathit{fold}\ h} b$$

Definition of *hylo*.

Another look and *hylo*

$$\begin{tikzcd}
F\,a \arrow[r, "F\,(unfold\,g)"] & F\,(\mu F) \arrow[r, "F\,(fold\,h)"] \arrow[d, shift left, "Roll"] & F\,b \arrow[d, "h"] \\
a \arrow[u, "g"] \arrow[r, dashed, "unfold\,g"'] & \mu F \arrow[u, shift left, "unRoll"] \arrow[r, dashed, "fold\,h"'] & b
\end{tikzcd}$$

By definitions of *fold* and *unfold*.

Another look and *hylo*

$$\begin{tikzcd}[column sep=huge, row sep=large]
F\,a \arrow[r, "F\,(unfold\,g)"] & F\,(\mu F) \arrow[r, "F\,(fold\,h)"] & F\,b \arrow[d, "h"] \\
a \arrow[u, "g"] \arrow[r, dashed, "unfold\,g"'] & \mu F \arrow[r, dashed, "fold\,h"'] & b
\end{tikzcd}$$

Since *unRoll* and *Roll* are inverses.

Another look and *hylo*

$$
\begin{tikzcd}
F\,a \arrow[r, "F\,(\mathit{fold}\;h \circ \mathit{unfold}\;g)"] & F\,b \arrow[d, "h"] \\
a \arrow[u, "g"] \arrow[r, dashed, "\mathit{fold}\;h \circ \mathit{unfold}\;g"'] & b
\end{tikzcd}
$$

By the *Functor* law: $\mathit{fmap}\;v \circ \mathit{fmap}\;u \equiv \mathit{fmap}\;(v \circ u)$.

Another look and *hylo*

$$\begin{tikzcd} F\,a \arrow[r, "F\,(hylo\,h\,g)"] & F\,b \arrow[d, "h"] \\ a \arrow[u, "g"] \arrow[r, dashed, "hylo\,h\,g"'] & b \end{tikzcd}$$

Definition of *hylo*. Directly recursive!

All together

$$
\begin{tikzcd}
F\;a \arrow[r, "F\;(\mathit{unfold}\;g)"] \arrow[rr, bend left=30, "F\;(\mathit{fold}\;h \circ \mathit{unfold}\;g)"] \arrow[rr, bend left=50, "F\;(\mathit{hylo}\;h\;g)"] & F\;(\mu F) \arrow[r, "F\;(\mathit{fold}\;h)"] \arrow[d, shift left, "\mathit{Roll}"] & F\;b \arrow[d, "h"] \\
a \arrow[u, "g"] \arrow[r, dashed, "\mathit{unfold}\;g"'] \arrow[rr, dashed, bend right=50, "\mathit{hylo}\;h\;g"'] & \mu F \arrow[u, shift left, "\mathit{unRoll}"] \arrow[r, dashed, "\mathit{fold}\;h"'] & b
\end{tikzcd}
$$

Reversed

$$
\begin{tikzcd}[column sep=huge, row sep=large]
F\,b & F\,(\mu F) \arrow[l, "F\,(fold\,h)"'] \arrow[d, shift left, "Roll"] & F\,a \arrow[l, "F\,(unfold\,g)"'] \arrow[d, "h"] \arrow[ll, bend right=30, "F\,(fold\,h \circ unfold\,g)"'] \arrow[ll, bend right=50, "F\,(hylo\,h\,g)"'] \\
b \arrow[u, "g"] & \mu F \arrow[l, dashed, "unfold\,g"] \arrow[u, shift left, "unRoll"] & a \arrow[l, dashed, "fold\,h"] \arrow[ll, dashed, bend left=45, "hylo\,h\,g"]
\end{tikzcd}
$$

fold and *unfold* via *hylo*

hylo subsumes both *fold* and *unfold*:

$$\begin{aligned} \mathit{unfold}\ g &= \mathit{hylo}\ \mathit{Roll}\ g \\ \mathit{fold}\ h &= \mathit{hylo}\ h\ \mathit{unRoll} \end{aligned}$$

since

$$\mathit{hylo}\ h\ g \equiv \mathit{fold}\ h \circ \mathit{unfold}\ g$$

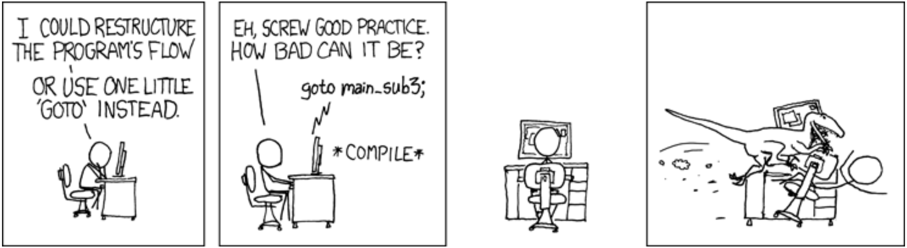
and

$$\mathit{fold}\ \mathit{Roll} \equiv \mathit{id} \equiv \mathit{unfold}\ \mathit{unRoll}$$

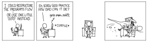
Summary

- *Fold* and *unfold* are structured replacements for the "assembly language" of recursive definitions.
- Unifying view of *fold* & *unfold* across data types via *functor fixpoints*.
- Recursive programs have a systematic translation to *unfold* and *fold*.
- The translation reveals parallelism clearly and simply.

A cautionary tale



Picture credits

-  Robert Lang's Origami BiCurve Pot 13
-  Maine Organic Farmers
-  unknown
- 

 Randall Munroe (xkcd)