The Essence of the ITERATOR Pattern

Bruno Oliveira
Jeremy Gibbons
University of Oxford

INTRODUCTION

- The ITERATOR pattern gives a clean interface for element-by-element access to a collection.
- Imperative iterations using the pattern have two simultaneous aspects: mapping and accumulating.
- In this work we looked at various functional iteration models and found that McBride and Paterson's idioms is the only model that manages to nicely capture both aspects of (internal) iterations.
- We also provide fusion laws for iterations and argue that our iteration model can generalise ITERATORS in some senses.

Internal Iterators in C#

foreach construct for simplified iteration over collections; generics for parametric polymorphism.

```
public static int loop⟨MyObj⟩ (IEnumerable⟨MyObj⟩ coll){
  int n = 0;
  foreach (MyObj obj in coll){
    n = n + 1;
    obj.touch ();
  }
  return n;
}
```

Parametrically polymorphic in collection elements; datatype-generic (ad-hoc polytypic) in collection shape via *IEnumerable* interface.

IDIOMS (OR APPLICATIVE FUNCTORS)

A generalization of $Monads$ introduced by McBride and Paterson.

$$\begin{aligned}&\textbf{class } Functor\ m \Rightarrow Idiom\ m \textbf{ where}\\ &\quad pure :: a \to m\ a\\ &\quad (\otimes)\ \ :: m\ (a \to b) \to m\ a \to m\ b\end{aligned}$$

Idioms obey the following laws:

$$\begin{aligned}
pure\ id \otimes u &= u && \text{\{-identity -\}}\\
pure\ (\circ) \otimes u \otimes v \otimes w &= u \otimes (v \otimes w) && \text{\{-composition -\}}\\
pure\ f \otimes pure\ x &= pure\ (f\ x) && \text{\{-homomorphism -\}}\\
u \otimes pure\ x &= pure\ (\lambda f \to f\ x) \otimes u && \text{\{-interchange -\}}
\end{aligned}$$

TRAVERSALS

The idiomatic map ($traverse$) operation can be defined for a large number of data types.

$$
\begin{aligned}
&\textbf{class } Traversable\ t \textbf{ where}\\
&\quad traverse :: Idiom\ i \Rightarrow (a \to i\ b) \to t\ a \to i\ (t\ b)\\
&\quad dist\ \ \ \ \ \ :: Idiom\ i \Rightarrow t\ (i\ a) \to i\ (t\ a)\\
&\quad dist\ \ \ \ \ \ = traverse\ id\\
&\textbf{instance } Traversable\ [\,] \textbf{ where}\\
&\quad traverse\ f\ [\,]\ \ \ \ \ \ = pure\ [\,]\\
&\quad traverse\ f\ (x : xs) = pure\ (:) \otimes f\ x \otimes traverse\ f\ xs
\end{aligned}
$$

A number of operations come for free given $traverse$.

IDENTITY IDIOM - MAPPING

The mapping aspect of ITERATORS can be captured using the *Identity* idiom.

$$\textbf{newtype}\ Id\ a = Id\{\, runId :: a \,\}$$

$$\textbf{instance}\ Idiom\ Id\ \textbf{where}$$

$$\quad pure\ = Id$$

$$\quad f \otimes x = Id\ (runId\ f\ \$\ runId\ x)$$

$$tmap :: Traversable\ t \Rightarrow (a \rightarrow b) \rightarrow t\ a \rightarrow t\ b$$

$$tmap\ f = runId \circ traverse\ (Id \circ f)$$

MONOIDAL IDIOM - ACCUMULATION

The accumulation aspect of ITERATORS can be captured with the constant $Idiom$ arising from a $Monoid$.

$$\begin{aligned}
&\textbf{class } Monoid\ o \textbf{ where}\\
&\quad \emptyset \quad :: o\\
&\quad (\oplus) :: o \to o \to o\\
&\textbf{newtype } Acc\ b\ a = A\{runA :: b\}\\
&\textbf{instance } Monoid\ b \Rightarrow Idiom\ (Acc\ b) \textbf{ where}\\
&\quad pure\ _ = A\ \emptyset\\
&\quad f \otimes x \ = A\ (runA\ f \oplus runA\ x)
\end{aligned}$$

(This idiom does not form a $Monad$.)

MONOIDAL IDIOM - ACCUMULATION

$traverse$ specialized to the constant idiom:

$$accumulate :: (Monoid\ b, Traversable\ t) \Rightarrow (a \to b) \to t\ a \to b$$
$$accumulate\ f = runA \circ traverse\ (A \circ f)$$

The $crush$ operation (Meertens, 1996) can be defined for any $Traversable$ type constructor.

$$crush :: (Monoid\ a, Traversable\ t) \Rightarrow t\ a \to a$$
$$crush = accumulate\ id$$

Backwards Idiom

Normally an idiom will do a left-to-right traversal. However, it is possible to reverse the direction of traversal for any idiom.

$$\mathbf{newtype}\ Backwards\ i\ a = B\{runB :: i\ a\}$$

$$\mathbf{instance}\ Idiom\ i \Rightarrow Idiom\ (Backwards\ i)\ \mathbf{where}$$

$$pure\ = B \circ pure$$

$$f \otimes x = B\ (pure\ (flip\ (\$)) \otimes runB\ x \otimes runB\ f)$$

There is also a trivial $Forwards$ idiom adapter that has no effect on the traversal.

Idiom Transformers

It is possible to define an adapter based on an idiom transformer that can be used to parameterize our traversals by the direction of the traversal.

$$\begin{aligned}
&\mathbf{data}\ IAdapter\ m = \forall g.\ Idiom\ (g\ m) \Rightarrow IAdapter \\
&\qquad\qquad (\forall a.\ m\ a \rightarrow g\ m\ a)\ (\forall a.\ g\ m\ a \rightarrow m\ a)
\end{aligned}$$

$$backwards :: Idiom\ m \Rightarrow IAdapter\ m$$

$$backwards = IAdapter\ B\ runB$$

$$\begin{aligned}
&ptraverse :: (Traversable\ t, Idiom\ i) \Rightarrow IAdapter\ i \rightarrow \\
&\qquad (a \rightarrow i\ b) \rightarrow t\ a \rightarrow i\ (t\ b)
\end{aligned}$$

$$ptraverse\ (IAdapter\ w\ runW)\ f = runW \circ traverse\ (w \circ f)$$

The direction of a traversal is first-class!

More generic operations

Accumulations parameterized by direction:

$$paccum :: (Monoid\ b, Traversable\ t) \Rightarrow IAdapter\ (Acc\ b) \rightarrow (a \rightarrow b) \rightarrow t\ a \rightarrow b$$
$$paccum\ d\ f = runA \circ ptraverse\ d\ (A \circ f)$$

Generalizing Meertens $crush$ combinator using a monoid $(id, \circ)$ of endo-functions.

$$reduceL :: Traversable\ t \Rightarrow (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow t\ a \rightarrow b$$
$$reduceL\ f = flip\ (runE \circ paccum\ forwards\ (E \circ flip\ f))$$
$$reduceR :: Traversable\ t \Rightarrow (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow t\ a \rightarrow b$$
$$reduceR\ f = flip\ (runE \circ paccum\ backwards\ (E \circ f))$$

These two operations, when specialized to lists, correspond to $foldl$ and $foldr$.

Laws of *traverse*

We know laws about idioms, but what about $traverse$? Should the following be a valid instance of $Traversable$?

$$\textbf{data}\ Tree\ a = Leaf\ a \mid Bin\ (Tree\ a)\ (Tree\ a)$$

$$\begin{aligned}&\textbf{instance}\ Traversable\ Tree\ \textbf{where}\\ &\quad traverse\ f\ (Leaf\ x)\ \ = pure\ Leaf \otimes f\ x\\ &\quad traverse\ f\ (Bin\ t\ u) = pure\ Bin \otimes traverse\ f\ u \otimes traverse\ f\ t\end{aligned}$$

What about this one?

$$\begin{aligned}&\textbf{instance}\ Traversable\ Tree\ \textbf{where}\\ &\quad traverse\ f\ (Leaf\ x)\ \ = pure\ Leaf \otimes f\ x\\ &\quad traverse\ f\ (Bin\ t\ u) = pure\ (flip\ Bin) \otimes traverse\ f\ u \otimes traverse\ f\ t\end{aligned}$$

NATURALITY

Distributor $dist$ should be natural in the idiom: for idiom transformation ϕ:

$$\begin{aligned}
&\phi :: m\ a \to n\ a \\
&\phi\ (pure_m\ a) &&= pure_n\ a \\
&\phi\ (mf \otimes_m mx) &&= \phi\ mf \otimes_n \phi\ mx
\end{aligned}$$

we require $dist$ to satisfy the following naturality property:

$$dist_n \circ fmap\ \phi = \phi \circ dist_m$$

One consequence of this naturality property is a 'purity law':

$$traverse\ pure = pure$$

This rules out the first definition of $traverse$ in the previous slide — but not the second one.

Free theorems

The free theorem arising from the type of $dist$ is:

$$dist \circ fmap\ (fmap\ k) = fmap\ (fmap\ k) \circ dist$$

As corollaries, we get the following two free theorems of $traverse$:

$$\begin{aligned} traverse\ (g \circ h) &= traverse\ g \circ fmap\ h \\ traverse\ (fmap\ k \circ f) &= fmap\ (fmap\ k) \circ traverse\ f \end{aligned}$$

Fusion of Traversals

Unlike monads, idioms are closed under composition:

$$\mathbf{newtype}\ Comp\ m\ n\ a = Comp\{\, unComp :: m\ (n\ a)\,\}$$

$$\begin{aligned}&\mathbf{instance}\ (Idiom\ m, Idiom\ n) \Rightarrow Idiom\ (Comp\ m\ n)\ \mathbf{where}\\ &\quad pure\ x \quad = Comp\ (pure\ (pure\ x))\\ &\quad mf \otimes mx = Comp\ (pure\ (\otimes) \otimes unComp\ mf \otimes unComp\ mx)\end{aligned}$$

This idiom together with the free theorems gives us the following fusion law:

$$traverse\ (Comp \circ fmap\ f \circ g) = Comp \circ fmap\ (traverse\ f) \circ traverse\ g$$

RepMin

Using a monoid $(maxBound, min)$ we can define

$$
\begin{aligned}
&tmin \quad :: (Ord\ a, Bounded\ a) \Rightarrow a \rightarrow Writer\ (Min\ a)\ a \\
&tmin\ a = \mathbf{do}\ \{tell\ (Min\ a); return\ a\} \\
&trep \quad :: a \rightarrow Reader\ a\ a \\
&trep\ a\ = ask \\
&trepmin :: (Ord\ a, Bounded\ a) \Rightarrow Tree\ a \rightarrow Tree\ a \\
&trepmin\ t = \mathbf{let}\ (r, m) = runWriter\ iteration\ \mathbf{in}\ runReader\ r\ (unMin\ m) \\
&\quad \mathbf{where}\ iteration = fmap\ (traverse\ trep)\ (traverse\ tmin\ t)
\end{aligned}
$$

RepMin

These two traversals can be fused using the fusion law of traversals.

$$
\begin{aligned}
&trepmin :: (Ord\ a, Bounded\ a) \Rightarrow Tree\ a \rightarrow Tree\ a \\
&trepmin\ t = \mathbf{let}\ (r, m) = runWriter\ iteration\ \mathbf{in}\ runReader\ r\ (unMin\ m) \\
&\quad \mathbf{where}\ iteration = unComp\ \$\ traverse\ (Comp \circ fmap\ trep \circ tmin)\ t
\end{aligned}
$$

CONCLUSIONS

- Idioms and $traverse$ provide a very nice theory for iterations. We have axioms and fusion laws.
- The direction of the traversal is first-class.
- Idiomatic traversals are more general than object-oriented ITERATORs in at least one sense: it is trivial with our approach to change the type of the collection elements with a traversal.