On Structuring Pure Functional Programs with Monoidal Profunctors

Alexandre Garcia de Oliveira

MSFP 2022

April 2, 2022

Outline

Monoidal Category

- Used to reason about monoids.
- Monoids give a structure with a binary operation that is associative and has a unit.
- Many monoids in a monoidal category represent a notion of computation [Rivas and Jaskelioff, 2017].
- This work presents another one.

Definition

- A profunctor generalizes the notion of function relations and bimodules [Leinster, 2003].
- Lifts two functions: one "covariantly" and the other "contravariantly".
- Same laws as a functor.

Profunctors in Haskell

From the `profunctors` package.

$$
\begin{array}{l}
\textbf{class}\ \mathit{Profunctor}\ p\ \textbf{where} \\
\quad \mathit{dimap} :: (a \to b) \to (c \to d) \to p\ b\ c \to p\ a\ d
\end{array}
$$

Satisfying:

$$
\begin{aligned}
\mathit{dimap}\ \mathit{id}\ \mathit{id} &= \mathit{id} \\
\mathit{dimap}\ (f \circ g)\ (h \circ i) &= \mathit{dimap}\ g\ h \circ \mathit{dimap}\ f\ i
\end{aligned}
$$

Profunctors in Haskell

$$\textbf{instance}\ \mathit{Profunctor}\ (\rightarrow)\ \textbf{where}$$
$$\quad \mathit{dimap}\ \mathit{ab}\ \mathit{cd}\ \mathit{bc} = \mathit{cd} \circ \mathit{bc} \circ \mathit{ab}$$

$$\textbf{data}\ \mathit{SISO}\ f\ g\ a\ b = \mathit{SISO}\ \{\mathit{unSISO} :: f\ a \rightarrow g\ b\}$$
$$\textbf{instance}\ (\mathit{Functor}\ f, \mathit{Functor}\ g) \Rightarrow$$
$$\quad \mathit{Profunctor}\ (\mathit{SISO}\ f\ g)\ \textbf{where}$$
$$\quad \mathit{dimap}\ \mathit{ab}\ \mathit{cd}\ (\mathit{SISO}\ \mathit{bc}) = \mathit{SISO}\ (\mathit{fmap}\ \mathit{cd} \circ \mathit{bc} \circ \mathit{fmap}\ \mathit{ab})$$

Day convolution - Profunctor case

- Gives the notion of a tensor;
- Desired properties to work with monoidal categories.
- In the Functor case, allows to reason about monoidal (applicative) functors.
- In the Profunctor case, allows to reason about monoidal profunctors.

Day convolution as coends

For Functors ($\mathcal{C}$ small category):
$(F \star G)(S, T) = \int^{XY} FX \times GY \times \mathcal{C}(X \otimes Y, T)$

Day convolution as coends

For Prounctors ($\mathcal{C}$ small category):
$(P \star Q)(S,T) = \int^{ABCD} P(A,B) \times Q(C,D) \times \mathcal{C}(S, A \otimes C) \times \mathcal{C}(B \otimes D, T)$

Day convolution datatype

$$\textbf{data}\ Day\ p\ q\ s\ t = \forall a\ b\ c\ d. Day\ (p\ a\ b)\ (q\ c\ d)\ (s \to (a, c))\ ((b, d) \to t)$$

A Monoid in the profunctor case

- The monoidal profunctor slogan is: "A monoid in the monoidal category of profunctors with day convolution as its tensor."

Examples

- Given $(\mathcal{C}, \otimes, I)$ any monoidal category and the *Hom* profunctor $P(A, B) = A \to B$.
- Relations: $\mathcal{C} = \mathcal{D} = (\mathbb{N}, \leqslant)$, the category of natural numbers viewed as a poset. The Kronecker product is a monoidal profunctor in this setting.
- The Kronecker product between a matrix A $p \times q$ matrix and B is a $m \times n$ matrix, is a matrix $A \otimes B$ with $pm \times qn$ entries given by $(A \otimes B) = a_{ij} B$.

Monoidal profunctors in Haskell

Appears in `product-profuntors` as `ProductProfunctor` [Tom Ellis, 2015], and as `Monoidal` in the work of Pickering, Gibbons and Wu.

$$
\begin{array}{l}
\textbf{class}\ \mathit{Profunctor}\ p \Rightarrow \mathit{MonoPro}\ p\ \textbf{where} \\
\quad \mathit{mpempty} :: p\ ()\ () \\
\quad (\star) :: p\ a\ b \rightarrow p\ c\ d \rightarrow p\ (a, c)\ (b, d)
\end{array}
$$

Satisfying monoidal laws.

- Left identity:

$$dimap\ diag\ snd\ (mpempty \star f) = f$$

- Right identity:

$$dimap\ diag\ fst\ (f \star mpempty) = f$$

- Associativity:

$$dimap\ \alpha^{-1}\ \alpha\ (f \star (g \star h)) = (f \star g) \star h$$

Examples

instance *MonoPro* $(\to)$ **where**
  $mpempty = id$
  $f \star g = \lambda(x, y) \to (f\ x, g\ y)$

Examples

$$
\begin{array}{l}
\textbf{instance}\ (\mathit{Functor}\ f, \mathit{Applicative}\ g) \Rightarrow \\
\qquad\qquad \mathit{MonoPro}\ (\mathit{SISO}\ f\ g)\ \textbf{where} \\
\quad \mathit{mpempty} = \mathit{SISO}\ (\lambda\_ \rightarrow \mathit{pure}\ ()) \\
\quad \mathit{SISO}\ f \star \mathit{SISO}\ g = \mathit{SISO}\ (\mathit{zip}' \circ (f \star g) \circ \mathit{unzip}')
\end{array}
$$

$$
\begin{array}{l}
\mathit{unzip}' :: \mathit{Functor}\ f \Rightarrow f\ (a, b) \rightarrow (f\ a, f\ b) \\
\mathit{unzip}' = (\mathit{fmap}\ \mathit{fst} \star \mathit{fmap}\ \mathit{snd}) \circ \mathit{diag}
\end{array}
$$

$$
\begin{array}{l}
\mathit{zip}' :: \mathit{Applicative}\ f \Rightarrow (f\ a, f\ b) \rightarrow f\ (a, b) \\
\mathit{zip}'\ (\mathit{fa}, \mathit{fb}) = \mathit{pure}\ (,) \otimes \mathit{fa} \otimes \mathit{fb}
\end{array}
$$

- There is a free construction for a monoidal profunctor [Rivas and Jaskelioff, 2017] [Milewski, 2017].
- $Prof(\mathcal{C}^{op}, \mathcal{C})$, when $\mathcal{C}$ is a small monoidal category, is monoidal with the Day convolution $\star$ and the profunctor J as its unit, and also have binary products and exponentials.
- Least fixed point of $\mu X.J + P \star X$.

$$
\begin{array}{l}
\textbf{data}\ \mathit{FreeMP}\ p\ s\ t\ \textbf{where} \\
\quad \mathit{MPempty} :: t \rightarrow \mathit{FreeMP}\ p\ s\ t \\
\quad \mathit{FreeMP} :: (s \rightarrow (x, z)) \rightarrow ((y, w) \rightarrow t) \\
\qquad\qquad\quad \rightarrow p\ x\ y \\
\qquad\qquad\quad \rightarrow \mathit{FreeMP}\ p\ z\ w \\
\qquad\qquad\quad \rightarrow \mathit{FreeMP}\ p\ s\ t
\end{array}
$$

Build a free monoidal profunctor.

– compare with singleton

$toFreeMP :: Profunctor\ p \Rightarrow p\ s\ t \rightarrow FreeMP\ p\ s\ t$
$toFreeMP\ p = FreeMP\ diag\ fst\ p\ (MPempty\ ())$

Append a profunctor.

$$
\begin{array}{l}
\quad \text{-- compare with (:)} \\
\mathit{consMP} :: \mathit{Profunctor}\ p \Rightarrow p\ a\ b \\
\quad \rightarrow \mathit{FreeMP}\ p\ s\ t \\
\quad \rightarrow \mathit{FreeMP}\ p\ (a, s)\ (b, t) \\
\mathit{consMP}\ \mathit{pab}\ (\mathit{MPEmpty}\ t) = \mathit{FreeMP}\ \mathit{id}\ \mathit{id}\ \mathit{pab}\ (\mathit{Arr}\ t) \\
\mathit{consMP}\ \mathit{pab}\ (\mathit{FreeMP}\ f\ g\ p\ \mathit{fp}) = \\
\quad \mathit{FreeMP}\ (\mathit{id} \star f)\ (\mathit{id} \star g)\ \mathit{pab}\ (\mathit{consMP}\ p\ \mathit{fp})
\end{array}
$$

Evaluate a monoidal profunctor.

– avoid impredicative polymorphism

$$
\begin{array}{l}
\textbf{data}\ \mathit{Prof}\ p\ q = \mathit{Prof}\ (\forall x\ y . p\ x\ y \to q\ x\ y) \\
\mathit{foldFreeMP} :: (\mathit{Profunctor}\ p, \mathit{MonoPro}\ q) \Rightarrow \\
\quad \mathit{Prof}\ p\ q \to \mathit{FreeMP}\ p\ s\ t \to q\ s\ t \\
\mathit{foldFreeMP}\ \_\ (\mathit{Arr}\ t) = \\
\quad \mathit{dimap}\ (\backslash\_ \to ())\ (\lambda() \to t)\ \mathit{arrr} \\
\mathit{foldFreeMP}\ (\mathit{Prof}\ h)\ (\mathit{FreeMP}\ f\ g\ p\ \mathit{mp}) = \\
\quad \mathit{dimap}\ f\ g\ ((h\ p) \star \mathit{foldFreeMP}\ (\mathit{Prof}\ h)\ \mathit{mp})
\end{array}
$$

The free monoidal profunctor is a monoidal profunctor.

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Profunctor}\ p \Rightarrow \mathit{MonoPro}\ (\mathit{FreeMP}\ p)\ \textbf{where}\\
&\quad \mathit{mpempty} = \mathit{MPempty}\ ()\\
&\quad \mathit{MPempty}\ t \qquad\qquad \star\ q \qquad\qquad\qquad\quad =\\
&\qquad \mathit{dimap}\ \mathit{snd}\ (\lambda x \to (t, x))\ q\\
&\quad q \qquad\qquad\qquad\qquad \star\ \mathit{MPempty}\ t \qquad\quad =\\
&\qquad \mathit{dimap}\ \mathit{fst}\ (\lambda x \to (x, t))\ q\\
&\quad (\mathit{FreeMP}\ f\ g\ p\ \mathit{fp}) \star (\mathit{FreeMP}\ k\ l\ \mathit{pp}\ \mathit{fq}) = \mathit{dimap}\ \mathit{t1}\ \mathit{t2}\ \mathit{t3}\\
&\qquad \textbf{where}\\
&\qquad\quad \mathit{t1} = (\mathit{assoc}' \circ (f \star k))\\
&\qquad\quad \mathit{t2} = (\mathit{sw} \circ (l \star g) \circ \mathit{associnv})\\
&\qquad\quad \mathit{t3} = (\mathit{consMP}\ p\ (\mathit{consMP}\ \mathit{pp}\ (\mathit{fp} \star \mathit{fq})))
\end{aligned}
$$

When p is an Arrow, `FreeMp p` is also an Arrow. To check this, one needes to collapse all monoidal profunctor in this structure to obtain the sequential composition.

$$
\begin{aligned}
&\textbf{instance}\ (\mathit{MonoPro}\ p, \mathit{Arrow}\ p) \Rightarrow \\
&\qquad\qquad \mathit{Category}\ (\mathit{FreeMP}\ p)\ \textbf{where} \\
&\quad \mathit{id} = \mathit{FreeMP}\ (\lambda x \to (x, ()))\ \mathit{fst}\ (\mathit{arr}\ \mathit{id})\ (\mathit{MPempty}\ ()) \\
&\quad \mathit{mp} \circ \mathit{mq} = \mathit{toFreeMP}\ (\mathit{fromFreeMP}\ \mathit{mp} \circ \mathit{fromFreeMP}\ \mathit{mq})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance}\ (\mathit{MonoPro}\ p, \mathit{Arrow}\ p) \Rightarrow \\
&\qquad\qquad \mathit{Arrow}\ (\mathit{FreeMP}\ p)\ \textbf{where} \\
&\quad \mathit{arr}\ f = \mathit{FreeMP}\ (\lambda x \to (x, ()))\ \mathit{fst}\ (\mathit{arr}\ f)\ (\mathit{MPempty}\ ()) \\
&\quad (\star\star\star) = (\star)
\end{aligned}
$$

$$\begin{array}{l}\textbf{class}\ \mathit{Profunctor}\ p \Rightarrow \mathit{MonoPro}\ p\ \textbf{where} \\ \quad \mathit{mpempty} :: p\ ()\ () \\ \quad (\star) :: p\ a\ b \to p\ c\ d \to p\ (a, c)\ (b, d)\end{array}$$

or,

$$\begin{array}{l}\quad \text{– compare with Day} \\ \textbf{class}\ \mathit{Profunctor}\ p \Rightarrow \mathit{MonoPro}\ p\ \textbf{where} \\ \quad \mathit{mpempty} :: (s \to ()) \to (() \to t) \to p\ s\ t \\ \quad (\star) :: (s \to (a, c)) \to ((b, d) \to t) \to p\ a\ b \to p\ c\ d \to p\ s\ t\end{array}$$

What if we change the pure functions? (Use a Kleisli category for $\mathcal{C}$).

$\textbf{class } \mathit{Category}\ k\ \textbf{where}$
$\quad \mathit{id}' :: k\ a\ a$
$\quad (<.>) :: k\ b\ c \to k\ a\ b \to k\ a\ c$

$\textbf{class } K.\mathit{Category}\ k \Rightarrow \mathit{CatProfunctor}\ k\ p\ \textbf{where}$
$\quad \mathit{catdimap} :: k\ a\ b \to k\ c\ d \to p\ b\ c \to p\ a\ d$

$\textbf{class } (\mathit{Category}\ k, \mathit{Profunctor}\ p) \Rightarrow \mathit{CatMonoPro}\ k\ p \mid p \to k\ \textbf{where}$
$\quad \mathit{cmpunit} :: k\ s\ () \to k\ ()\ t \to p\ s\ t$
$\quad \mathit{mpZipWith} :: k\ s\ (a, c) \to k\ (b, d)\ t \to p\ a\ b \to p\ c\ d \to p\ s\ t$
$\quad \mathit{lconvolve} :: k\ s\ (a, c) \to p\ a\ b \to p\ c\ d \to p\ s\ (b, d)$
$\quad \mathit{lconvolve}\ f = \mathit{mpZipWith}\ f\ \mathit{id}'$
$\quad \mathit{rconvolve} :: k\ (b, d)\ t \to p\ a\ b \to p\ c\ d \to p\ (a, c)\ t$
$\quad \mathit{rconvolve}\ g = \mathit{mpZipWith}\ \mathit{id}'\ g$

Same rules.

An example

– lift when a b

$$\textbf{newtype}\ \mathit{Lift}\ t\ m\ a\ b = \mathit{Lift}\ \{\mathit{runLift} :: m\ a \rightarrow t\ m\ b\}$$

– Lift have a CatMonoPro instance

– provided that there is a function

$$\mathit{comm} :: (\mathit{Monad}\ m, \mathit{Traversable}\ m) \Rightarrow t\ m\ (m\ a) \rightarrow t\ m\ a$$

– and (t m) is a monad. Works with MaybeT Writer.

Quicksort with CatMonoPro

$$
\begin{array}{l}
\mathit{lift'} :: \mathit{Lift}\ \mathit{MaybeT}\ (\mathit{Writer}\ [\mathit{String}])\ b\ b \\
\mathit{lift'} = \mathit{Lift}\ \mathit{lift} \\[4pt]
\mathit{qsort} :: [\mathit{String}] \to \mathit{MaybeT}\ (\mathit{Writer}\ [\mathit{String}])\ [\mathit{String}] \\
\mathit{qsort}\ [\,] = \mathit{return}\ [\,] \\
\mathit{qsort}\ \mathit{xs} = \mathbf{do} \\
\quad \mathit{guard}\ (\mathit{head}\ \mathit{xs} \not\equiv \texttt{""}) \\
\qquad \text{-- effectful divide} \\
\quad (\mathit{ls}, \mathit{rs}) \leftarrow \mathit{runLift}\ (\mathit{lconvolve}\ (\mathit{Kleisli}\ \mathit{lsplit})\ \mathit{lift'}\ \mathit{lift'})\ (\mathit{return}\ \mathit{xs}) \\
\quad (\mathit{ls'}, \mathit{rs'}) \leftarrow \mathit{runKleisli}\ ((\mathit{Kleisli}\ \mathit{qsort}) \star (\mathit{Kleisli}\ \mathit{qsort}))\ (\mathit{ls}, \mathit{rs}) \\
\qquad \text{-- effectful conquer} \\
\quad \mathit{ss} \leftarrow \mathit{runLift}\ (\mathit{rconvolve}\ (\mathit{Kleisli}\ (\mathit{rsplit}\ (\mathit{head}\ \mathit{xs})))\ \mathit{lift'}\ \mathit{lift'}) \\
\qquad (\mathit{return}\ (\mathit{ls'}, \mathit{rs'})) \\
\quad \mathit{return}\ \mathit{ss}
\end{array}
$$

Quicksort with CatMonoPro

```
lsplit :: [String] → Writer [String] ([String], [String])
lsplit (z : zs) = do
  xs ← return (filter (<z) zs)
  ys ← return (filter (⩾ z) zs)
  tell ["Splitting: " ++ show zs
    ++ " into "
    ++ show xs
    ++ ", "
    ++ show ys]
  return (xs, ys)
```

$$
\begin{array}{l}
\mathit{rsplit} :: \mathit{String} \to ([\,\mathit{String}\,], [\,\mathit{String}\,]) \to \mathit{Writer}\ [\,\mathit{String}\,]\ [\,\mathit{String}\,] \\
\mathit{rsplit}\ l\ (\mathit{xs}, \mathit{ys}) = \mathbf{do} \\
\quad \mathit{tell}\ [\,\texttt{"Merging: "} +\!\!+\ \mathit{show}\ \mathit{xs} \\
\qquad +\!\!+\ \texttt{", "} \\
\qquad +\!\!+\ l \\
\qquad +\!\!+\ \texttt{", and "} \\
\qquad +\!\!+\ \mathit{show}\ \mathit{ys}\,] \\
\quad \mathit{return}\ (\mathit{xs} +\!\!+ [\,l\,] +\!\!+ \mathit{ys})
\end{array}
$$

- Pure functional data accessors (compositional) [Pickering, Gibbons and Wu, 2017];
- Generalize getters and setters for ADTs;
- An optic is a general denotation to locate parts (whole) of a data structure in which some action needs to be performed.
- Each optic deals with different ADTs;
- Lenses are to product types, prisms to sum types, traversals with traversable containers, grates with function types, etc. Isos works with any type but cannot do much with the shape of data.

Profunctor optics

- Getters $s \to a$ and setters $s \to b \to t$ can be amalgamated in a single type $\forall p.c\ p \Rightarrow p\ a\ b \to p\ s\ t$ where p is a profunctor and c is a typeclass constraint.
- When p is a profunctor, we have an iso.
- When p is strong, we have a lens [O'Connor, 2015].
- When p is closed, we have a grate [O'Connor, 2015].
- When p is a monoidal profunctor, we get a mix of grates and traversals.
- The monoidal profunctor optic is given by
  type $\mathit{Mono}\ s\ t\ a\ b = \forall p.\mathit{MonoPro}\ p \Rightarrow p\ a\ b \to p\ s\ t$.

  class $\mathit{Profunctor}\ p \Rightarrow \mathit{Closed}\ p$ **where**
  $\quad \mathit{closed} :: p\ a\ b \to p\ (x \to a)\ (x \to b)$

Some monos

$$
\begin{aligned}
&\mathit{each2} :: \mathit{MonoPro}\ p \Rightarrow p\ a\ b \rightarrow p\ (a, a)\ (b, b) \\
&\mathit{each2}\ p = p \star p \\
&\mathit{each3} :: \mathit{MonoPro}\ p \Rightarrow p\ a\ b \rightarrow p\ (a, a, a)\ (b, b, b) \\
&\mathit{each3}\ p = \mathit{dimap}\ \mathit{flat3i}\ \mathit{flat3l}\ (p \star p \star p) \\
&\mathit{convolve2} :: (\mathit{Functor}\ f, \mathit{Applicative}\ g, \mathit{MonoPro}\ p) \Rightarrow \\
&\quad p\ (f\ a)\ (g\ b) \rightarrow p\ (f\ (a, a))\ (g\ (b, b)) \\
&\mathit{convolve2}\ p = \mathit{dimap}\ \mathit{unzip}'\ \mathit{zip}'\ (p \star p)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{foldMapOf} :: \mathit{Monoid}\ r \Rightarrow \\
&\qquad \mathit{Mono}\ s\ t\ a\ b \to (a \to r) \to s \to r \\
&\mathit{foldMapOf}\ \mathit{lens}\ f = \mathit{runForget}\ (\mathit{lens}\ (\mathit{Forget}\ f))
\end{aligned}
$$

$$
\mathit{foldMapOf}\ \mathit{each3} :: \mathit{Monoid}\ r \Rightarrow (a \to r) \to (a, a, a) \to r
$$

Every profunctorial optic has a van Laarhoven [O'Connor,2015], functorial representation. For monoidal profunctors:

$$
\begin{aligned}
&\mathit{convolute} :: (\mathit{Applicative}\ g, \mathit{Functor}\ f) \Rightarrow \\
&\qquad\qquad \mathit{Mono}\ s\ t\ a\ b \to (f\ a \to g\ b) \to f\ s \to g\ t \\
&\mathit{convolute}\ \mathit{mono}\ f = \mathit{unSISO}\ (\mathit{mono}\ (\mathit{SISO}\ f))
\end{aligned}
$$

Also called as a `FiniteGrate` by O'Connor but without the monoidal profunctor semantics.

If we specialize *convolute* to the identity functor $f = Id$ to get a traversal [Kmett, 2011]:

$$
\begin{aligned}
&traverseOf :: Applicative\ g \Rightarrow \\
&\qquad Mono\ s\ t\ a\ b \to (Id\ a \to g\ b) \to (Id\ s \to g\ t) \\
&traverseOf\ mono = convolute\ mono
\end{aligned}
$$

Specializing *convolute* to the applicative functor $g = Id$ gives a grate [O'Connor, 2015]:

$$
\begin{aligned}
&zipFWithOf :: Functor\ f \Rightarrow \\
&\qquad Mono\ s\ t\ a\ b \to (f\ a \to Id\ b) \to (f\ s \to Id\ t) \\
&zipFWithOf\ mono = convolute\ mono
\end{aligned}
$$

About monoidal profunctors

- The Arrow interface provides the same parallel operation of a monoidal (product) profunctor, but also provides a way to create a trivial computation basing on a pure function, and a notion of sequential composition;
- Monoidal profunctors provides the parallel composition and a trivial (unit) computation. Monoidal profunctor is, in this sense, a weaker interface than Arrow;
- Monoidal profunctors are stronger than a plain profunctor since it can lift a pure function with several parameters;

$$
\begin{aligned}
&lmap :: Profunctor\ p \Rightarrow (a \to b) \to p\ b\ c \to p\ a\ c \\
&lmap2 :: MonoPro\ p \Rightarrow ((b, bb) \to b') \to p\ a\ b \\
&\quad \to p\ c\ bb \to p\ (a, c)\ b' \\
&lmap2\ f\ pa\ pc = dimap\ id\ f\ (pa \star pc)
\end{aligned}
$$

About monoidal profunctors

- Provides a nice way to think about optics;
- Free monoidal profunctors have a proper nature to deal with parallel computations (computations cannot interleave);
- Kan Extensions can help to derive more Monoidal Profunctor examples;
- It can shine in linear typesetting, can be used to model contexts, and also can be used to parser libraries.
- Can be used to do matrix programming;
- In this work, we discussed only the case when $\otimes = \times$, i.e., the product case of a monoidal profunctor. Using this with other tensors can be very useful. Othe tensors can provide other optics (even mixed optics with a smart $\otimes$ choice).
- Enhance the monoidal profunctor inerface with this one.

class *SemiCat p* **where**
  $(\circ) :: p\ a\ b \to p\ b\ c \to p\ a\ c$

Thank you for your attention.