# Folds, Scans, and Moore Machines as Monoidal Profunctor Homomorphisms

Alexandre Garcia de Oliveira

MSFP 2024 - Tallinn, Estonia

July 8, 2024

# Outline

# The idea

- This paper is a theoretical application of monoidal profunctors, since every key idea depends on it.

# The idea

- This paper is a theoretical application of monoidal profunctors, since every key idea depends on it.
- Functional programming always relies on mathematical laws to describe, for example, that an evaluation of an expression is well-behaved.

# The idea

- This paper is a theoretical application of monoidal profunctors, since every key idea depends on it.
- Functional programming always relies on mathematical laws to describe, for example, that an evaluation of an expression is well-behaved.
- For example, a map over a list needs to preserve composition, and if we map the identity function, nothing happens.

# The idea

- This paper is a theoretical application of monoidal profunctors, since every key idea depends on it.
- Functional programming always relies on mathematical laws to describe, for example, that an evaluation of an expression is well-behaved.
- For example, a map over a list needs to preserve composition, and if we map the identity function, nothing happens.
- With monads, when chaining multiple functions with bind, the way we group the functions does not affect the result. Also, if we have a monadic value and bind it to the return function, the result is the same as the original monadic value.

# Map for Lists Law in Haskell

- **Identity:**
  - $map\ id\ xs \equiv xs$
- **Preserves Composition:**
  - $map\ (f \circ g)\ xs \equiv map\ f\ (map\ g\ xs)$

# Monad Laws in Haskell

- **Left identity:** $\mathit{return}\ a \gg\!= k \equiv k\ a$
- **Right identity:** $m \gg\!= \mathit{return} \equiv m$
- **Associativity:** $(m \gg\!= k) \gg\!= h \equiv m \gg\!= (\lambda x \rightarrow k\ x \gg\!= h)$

# Functor laws

- The law for lists generalizes to every Functor instance.

# Functor laws

- The law for lists generalizes to every Functor instance.
- **Identity:**
  - $fmap\ id\ xs \equiv xs$
- **Preserves Composition:**
  - $fmap\ (f \circ g)\ xs \equiv fmap\ f\ (fmap\ g\ xs)$

# Monoidal Category laws

A *monoidal category* is a sextuple $(\mathcal{C}, \otimes, I, \alpha, \rho, \lambda)$ where

- $\mathcal{C}$ is a category;
- $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is a bifunctor;
- $I$ is an object of $\mathcal{C}$ called unit;
- $\rho_A : A \otimes I \to A$, $\lambda_A : I \otimes A \to A$ and $\alpha_{ABC} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ are three natural isomorphisms such that the diagrams below commute.

$$
\begin{tikzcd}
A \otimes (B \otimes (C \otimes D)) & (A \otimes B) \otimes (C \otimes D) \arrow[l, "\alpha"'] & ((A \otimes B) \otimes C) \otimes D \arrow[l, "\alpha"'] \\
A \otimes ((B \otimes C) \otimes D) \arrow[u, "id \otimes \alpha"'] & & (A \otimes (B \otimes C)) \otimes D \arrow[ll, "\alpha"'] \arrow[u, "\alpha \otimes id"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
A \otimes (I \otimes B) \arrow[rr, "\alpha"] \arrow[dr, "id \otimes \lambda"'] & & (A \otimes I) \otimes B \arrow[dl, "\rho \otimes id"] \\
& A \otimes B. &
\end{tikzcd}
$$

# Monoidal Category laws

A *monoid* in a monoidal category $\mathcal{C}$ is a tuple $(M, e, m)$ where $M$ is an object of $\mathcal{C}$, $e : I \to M$ is the unit morphism and $m : M \otimes M \to M$ is the multiplication morphism, satisfying

- Right unit: $m \circ (id \otimes e) = \rho_M$
- Left unit: $m \circ (e \otimes id) = \lambda_M$
- Associativity: $m \circ (m \otimes id) = m \circ (id \otimes m) \circ \alpha_M$

# Law for folds

- (Fusion law) $foldl\ op\ e \circ map\ f = foldl\ (\lambda s\ a \rightarrow op\ s\ (f\ \ a))\ e$
- (Parallel-bifold law)
  $bifoldl\ f\ \ g\ (e, u)\ (zip\ as\ bs) = (foldl\ f\ \ e\ as, foldl\ g\ u\ bs)$

# Definition of bifoldl

$$
\begin{aligned}
&\mathit{bifoldl} :: (c \to a \to c) \to (d \to b \to d) \to (c, d) \to [(a, b)] \to (c, d)\\
&\mathit{bifoldl}\ f\ g\ (c, d)\ [\,] = (c, d)\\
&\mathit{bifoldl}\ f\ g\ (c, d)\ ((a, b) : \mathit{xs}) = \mathit{bifoldl}\ f\ g\ (f\ c\ a, g\ d\ b)\ \mathit{xs}
\end{aligned}
$$

# Abstracting the Pattern

- The first law represents the naturality condition of a natural transformation between two profunctors.

# Abstracting the Pattern

- The first law represents the naturality condition of a natural transformation between two profunctors.
- The second law derives from a homomorphism that preserves the structure between two monoidal profunctors.

# Monoidal Profunctor in Haskell

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

# Monoidal Profunctor in Haskell

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

- Left identity: $\mathit{dimap}\ \mathit{diagr}\ \mathit{snd}\ (\mathit{mpempty} \star f) = f$
- Right identity: $\mathit{dimap}\ \mathit{diagl}\ \mathit{fst}\ (f \star \mathit{mpempty}) = f$
- Associativity: $\mathit{dimap}\ \alpha^{-1}\ \alpha\ (f \star (g \star h)) = (f \star g) \star h$

# First example

$$
\begin{aligned}
&\textbf{instance}\ \mathit{MonoPro}\ (\to)\ \textbf{where} \\
&\quad \mathit{mpempty} = \lambda() \to () \\
&\quad f \star g = \lambda(a, b) \to (f\ a, g\ b)
\end{aligned}
$$

# Second example

$$\textbf{data}\ \mathit{Moore}\ a\ b = \mathit{Moore}\ b\ (a \to \mathit{Moore}\ a\ b)$$

## Second example

$$\textbf{data}\ \mathit{Moore}\ a\ b = \mathit{Moore}\ b\ (a \to \mathit{Moore}\ a\ b)$$

$$\begin{array}{l}
\textbf{instance}\ \mathit{Profunctor}\ \mathit{Moore}\ \textbf{where} \\
\quad \mathit{dimap}\ f\ g\ (\mathit{Moore}\ c\ \mathit{bm}) = \mathit{Moore}\ (g\ c)\ (\mathit{dimap}\ f\ g \circ \mathit{bm} \circ f) \\
\textbf{instance}\ \mathit{MonoPro}\ \mathit{Moore}\ \textbf{where} \\
\quad \mathit{mpempty} = \mathit{Moore}\ ()\ (\backslash_ \to \mathit{mpempty}) \\
\quad (\mathit{Moore}\ b\ \mathit{am}) \star (\mathit{Moore}\ d\ \mathit{cm}) = \\
\qquad \mathit{Moore}\ (b, d)\ (\lambda(a, c) \to \mathit{am}\ a \star \mathit{cm}\ c)
\end{array}$$

# Third example

$$\textbf{data}\ SISO\ f\ g\ a\ b = SISO\ \{\ unSISO :: f\ a \to g\ b\ \}$$

# Third example

$$\textbf{data}\ SISO\ f\ g\ a\ b = SISO\ \{unSISO :: f\ a \to g\ b\}$$

$$\begin{aligned}
&\textbf{instance}\ (Functor\ f, Functor\ g) \Rightarrow Profunctor\ (SISO\ f\ g)\ \textbf{where}\\
&\quad dimap\ ab\ cd\ (SISO\ bc) = SISO\ (fmap\ cd \circ bc \circ fmap\ ab)\\
&\textbf{instance}\ (Functor\ f, Applicative\ g) \Rightarrow MonoPro\ (SISO\ f\ g)\ \textbf{where}\\
&\quad mpempty = SISO\ (\lambda_ \to pure\ ())\\
&\quad SISO\ f \star SISO\ g = SISO\ (zip' \circ (f \star g) \circ unzip')
\end{aligned}$$

# Moore machines from coalgebras

$\textbf{data}\ \mathit{MooreCoalg}\ s\ a\ b = \mathit{MooreCoalg}\ (s \to b)\ (s \to a \to s)$

# Moore machines from coalgebras

$$\textbf{data}\ \mathit{MooreCoalg}\ s\ a\ b = \mathit{MooreCoalg}\ (s \to b)\ (s \to a \to s)$$

$$\begin{aligned}
&\mathit{buildMoore} :: \mathit{MooreCoalg}\ s\ a\ b \to s \to \mathit{Moore}\ a\ b \\
&\mathit{buildMoore}\ (\mathit{MooreCoalg}\ \mathit{out}\ \mathit{next})\ s = \\
&\quad \mathit{Moore}\ (\mathit{out}\ s)\ (\mathit{buildMoore}\ (\mathit{MooreCoalg}\ \mathit{out}\ \mathit{next}) \circ \mathit{next}\ s)
\end{aligned}$$

# Moore machines and folds

- $foldl :: (s \to a \to s) \to s \to [\,a\,] \to s$

# Moore machines and folds

- $\mathit{foldl} :: (s \to a \to s) \to s \to [\,a\,] \to s$
- Expand: $\mathit{ofoldl} :: (s \to a \to s) \to s \to (s \to b) \to [\,a\,] \to b$

# Moore machines and folds

- $\mathit{foldl} :: (s \to a \to s) \to s \to [\,a\,] \to s$
- Expand: $\mathit{ofoldl} :: (s \to a \to s) \to s \to (s \to b) \to [\,a\,] \to b$
- Use Moore: $\mathit{mfoldl} :: \mathit{Moore}\ a\ b \to [\,a\,] \to b$

# Moore machines and folds

- $foldl :: (s \to a \to s) \to s \to [\,a\,] \to s$
- Expand: $ofoldl :: (s \to a \to s) \to s \to (s \to b) \to [\,a\,] \to b$
- Use Moore: $mfoldl :: Moore\ a\ b \to [\,a\,] \to b$
- Pack into SISO $(f\ a \to g\ b)$:
  $mfoldl :: Moore\ a\ b \to SISO\ [\,]\ Identity\ a\ b$

# A natural transformation emerges

- *mfoldl* is a natural transformation: for every $h :: a \to b$, $i :: c \to d$ the following diagram commutes.

$$\begin{tikzcd}
\text{Moore } b\ c \arrow[r, "dimap\ h\ i"] \arrow[d, "mfoldl"'] & \text{Moore } a\ d \arrow[d, "mfoldl"] \\
\text{Fold } b\ c \arrow[r, "dimap\ h\ i"'] & \text{Fold } a\ d
\end{tikzcd}$$

# Monoidal profunctor homomorphism

- *mfoldl* is a monoidal homomorphism between the profunctors *Moore* and *SISO* [ ] *Identity*.

# Monoidal profunctor homomorphism

- *mfoldl* is a monoidal homomorphism between the profunctors *Moore* and *SISO* [ ] *Identity*.
- *mfoldl* preserves unit *mpempty*: *mfoldl mpempty* = *mpempty*

# Monoidal profunctor homomorphism

- *mfoldl* is a monoidal homomorphism between the profunctors *Moore* and $SISO\ [\ ]\ Identity$.
- *mfoldl* preserves unit *mpempty*: $mfoldl\ mpempty = mpempty$
- *mfoldl* preserves the monoidal profunctor multiplication $\star$: $mfoldl\ (m \star n) = mfoldl\ m \star mfoldl\ n$. The lhs is the parallel composition of Moore machines, the rhs is the same composition in a *SISO*.

# Laws and monoidal profunctor homomorphism

- Naturality, gives us the foldl fusion law: $\mathit{foldl}\ \mathit{op}\ e \circ \mathit{map}\ g = \mathit{foldl}\ (\lambda s\ a \to \mathit{op}\ s\ (g\ a))\ e$. This is a special case of the naturality condition: $\mathit{dimap}\ g\ \mathit{id} \circ \mathit{mfoldl} = \mathit{mfoldl} \circ \mathit{dimap}\ g\ \mathit{id}$.

# Laws and monoidal profunctor homomorphism

- Naturality, gives us the foldl fusion law: $\mathit{foldl}\ \mathit{op}\ e \circ \mathit{map}\ g = \mathit{foldl}\ (\lambda s\ a \rightarrow \mathit{op}\ s\ (g\ a))\ e$. This is a special case of the naturality condition: $\mathit{dimap}\ g\ \mathit{id} \circ \mathit{mfoldl} = \mathit{mfoldl} \circ \mathit{dimap}\ g\ \mathit{id}$.
- The preservation of identity tells us that folding over a list of () is just ().

# Laws and monoidal profunctor homomorphism

- Naturality, gives us the foldl fusion law: $foldl\ op\ e \circ map\ g = foldl\ (\lambda s\ a \to op\ s\ (g\ a))\ e$. This is a special case of the naturality condition: $dimap\ g\ id \circ mfoldl = mfoldl \circ dimap\ g\ id$.
- The preservation of identity tells us that folding over a list of () is just ().
- The preservation of monoidal multiplication tells us that $foldl\ (curry\ (lmap\ (\lambda((a,b),(c,d)) \to ((a,c),(b,d)))\ (uncurry\ f \star uncurry\ g)))\ (e,u)\ (zip\ as\ bs) \equiv (foldl\ f\ e\ as, foldl\ g\ u\ bs)$.

# Laws and monoidal profunctor homomorphism

- Naturality, gives us the foldl fusion law:
  $\mathit{foldl}\ \mathit{op}\ e \circ \mathit{map}\ g = \mathit{foldl}\ (\lambda s\ a \rightarrow \mathit{op}\ s\ (g\ a))\ e$. This is a special case of the naturality condition: $\mathit{dimap}\ g\ \mathit{id} \circ \mathit{mfoldl} = \mathit{mfoldl} \circ \mathit{dimap}\ g\ \mathit{id}$.
- The preservation of identity tells us that folding over a list of () is just ().
- The preservation of monoidal multiplication tells us that
  $\mathit{foldl}\ (\mathit{curry}\ (\mathit{lmap}\ (\lambda((a,b),(c,d)) \rightarrow ((a,c),(b,d)))\ (\mathit{uncurry}\ f \star \mathit{uncurry}\ g)))\ (e,u)\ (\mathit{zip}\ \mathit{as}\ \mathit{bs}) \equiv (\mathit{foldl}\ f\ e\ \mathit{as}, \mathit{foldl}\ g\ u\ \mathit{bs})$.
- Or using bifoldl:
  $\mathit{bifoldl}\ f\ g\ (e,u)\ (\mathit{zip}\ \mathit{as}\ \mathit{bs}) = (\mathit{foldl}\ f\ e\ \mathit{as}, \mathit{foldl}\ g\ u\ \mathit{bs})$

# The same is valid for scans

$$\textbf{type}\ \mathit{Scan}\ a\ b = \mathit{SISO}\ [\,]\ \mathit{ZipNonEmpty}\ a\ b$$

$$\mathit{mscanl} :: \mathit{Moore}\ a\ b \to \mathit{Scan}\ a\ b$$
$$\mathit{mscanl}\ m = \mathit{SISO}\ (\lambda \mathit{as} \to \mathit{ZipNonEmpty}\ (\mathit{runMoore}\ m\ \mathit{as}))$$

$$
\begin{array}{l}
\textbf{data}\ \mathit{NonEmpty}\ a = a \triangleright [\,a\,] \\
\textbf{data}\ \mathit{ZipNonEmpty}\ a = \mathit{ZipNonEmpty}\ \{\mathit{unzipne} :: \mathit{NonEmpty}\ a\} \\
\quad \textbf{deriving}\ \mathit{Show} \\
\textbf{instance}\ \mathit{Functor}\ \mathit{ZipNonEmpty}\ \textbf{where} \\
\quad \mathit{fmap}\ f\ (\mathit{ZipNonEmpty}\ (a \triangleright \mathit{as})) = \\
\quad\quad \mathit{ZipNonEmpty}\ ((f\ a) \triangleright \mathit{fmap}\ f\ \mathit{as}) \\
\textbf{instance}\ \mathit{Applicative}\ \mathit{ZipNonEmpty}\ \textbf{where} \\
\quad \mathit{pure}\ a = \mathit{ZipNonEmpty}\ (a \triangleright \mathit{repeat}\ a) \\
\quad (\mathit{ZipNonEmpty}\ (f \triangleright \mathit{fs})) \otimes (\mathit{ZipNonEmpty}\ (x \triangleright \mathit{xs})) \\
\quad\quad = \mathit{ZipNonEmpty}\ ((f\ x) \triangleright \mathit{zipWith}\ (\$)\ \mathit{fs}\ \mathit{xs})
\end{array}
$$

# Contributions

- Investigating *Moore* type and *SISO* as instances of monoidal profunctors has provided valuable insights..

# Contributions

- Investigating *Moore* type and *SISO* as instances of monoidal profunctors has provided valuable insights..
- Establishing a significant mathematical connection: a natural transformation links Moore machines with folds and scans, and this transformation preserves the monoidal profunctor operations.

# Contributions

- Investigating *Moore* type and *SISO* as instances of monoidal profunctors has provided valuable insights..
- Establishing a significant mathematical connection: a natural transformation links Moore machines with folds and scans, and this transformation preserves the monoidal profunctor operations.
- Laws in the monoidal profunctor domain directly translate to laws governing folds and scans.

# Contributions

- Investigating *Moore* type and *SISO* as instances of monoidal profunctors has provided valuable insights..
- Establishing a significant mathematical connection: a natural transformation links Moore machines with folds and scans, and this transformation preserves the monoidal profunctor operations.
- Laws in the monoidal profunctor domain directly translate to laws governing folds and scans.
- Applying this pattern of reasoning can help identify lawful computations using monoidal profunctors.

# Paper and Future Work

- Provide comprehensive proofs for all claims made.

# Paper and Future Work

- Provide comprehensive proofs for all claims made.
- Include detailed examples to illustrate the rules in their final form.

# Paper and Future Work

- Provide comprehensive proofs for all claims made.
- Include detailed examples to illustrate the rules in their final form.
- Definitions of the structures presented.

# Paper and Future Work

- Provide comprehensive proofs for all claims made.
- Include detailed examples to illustrate the rules in their final form.
- Definitions of the structures presented.
- Explore whether other monoidal profunctors yield similar results. Unfolds? Traversals?