Chapter 9: Traversable functors and contrafunctors

Sergei Winitzki

Academy by the Bay

2018-08-08

Motivation for the `traverse` operation

- Consider data of type List^A and processing $f : A \Rightarrow \text{Future}^B$
- Typically, we want to wait until the entire data set is processed
- What we need is $\text{List}^A \Rightarrow \left(A \Rightarrow \text{Future}^B\right) \Rightarrow \text{Future}^{\text{List}^B}$
- Generalize: $L^A \Rightarrow \left(A \Rightarrow F^B\right) \Rightarrow F^{L^B}$ for some type constructors F, L
- This operation is called `traverse`
  - How to implement it: for example, a 3-element list is $A \times A \times A$
  - Consider $L^A \equiv A \times A \times A$, apply map f and get $F^B \times F^B \times F^B$
  - We will get $F^{L^B} \equiv F^{B \times B \times B}$ if we can apply `zip` as $F^B \times F^B \Rightarrow F^{B \times B}$
- So we need to assume that F is applicative
- In Scala, we have `Future.traverse()` that assumes L to be a sequence
  - This is the easy-to-remember example that fixes the requirements
- Questions:
  - Which functors L can have this operation?
  - Can we express `traverse` through a simpler operation?
  - What are the required laws for `traverse`?
  - What about contrafunctors or profunctors?

- The type signature of `traverse` is a complicated "lifting"
  - A "lifting" is often equivalent to a simpler natural transformation
- To derive it, ask: what is missing from `fmap` to do the job of `traverse`?

$$\mathrm{fmap}_L : (A \Rightarrow F^B) \Rightarrow L^A \Rightarrow L^{F^B}$$

- We need F^{L^B}, but the `traverse` operation gives us L^{F^B} instead
  - What's missing is a natural transformation `sequence` : $L^{F^B} \Rightarrow F^{L^B}$
- The functions `traverse` and `sequence` are computationally equivalent:

$$\mathrm{trav}\ f^{\underline{A\Rightarrow F^B}} = \mathrm{fmap}_L\ f \circ \mathrm{seq}$$

$$\begin{tikzcd}
 & L^{F^B} \arrow[dr, "\mathrm{seq}"] & \\
L^A \arrow[ur, "\mathrm{fmap}_L\ f^{\underline{A\Rightarrow F^B}}"] \arrow[rr, "\mathrm{trav}\ f^{\underline{A\Rightarrow F^B}}"'] & & F^{L^B}
\end{tikzcd}$$

Here F is an *arbitrary* applicative functor

- Keep in mind the example `Future.sequence` : $\mathrm{List}^{\mathrm{Future}^X} \Rightarrow \mathrm{Future}^{\mathrm{List}^X}$
- Examples: $L^A \equiv A \times A \times A$; $L^A = \mathrm{List}^A$; finite trees
- Non-traversable: $L^A \equiv R \Rightarrow A$; lazy list ("infinite product")
  - ★ Note: We *cannot have* the opposite transformation $F^{L^B} \Rightarrow L^{F^B}$

Polynomial functors are traversable

- Generalize from the example $L^A \equiv A \times A \times A$ to other polynomials
- Polynomial functors have the form

$$L^A \equiv Z \times A \times ... \times A + Y \times A \times ... \times A + ... + Q \times A + P$$

- To implement $\text{seq} : L^{F^B} \Rightarrow F^{L^B}$, consider monomial $L^A \equiv Z \times A \times ... \times A$
- We have $L^{F^B} = Z \times F^B \times ... \times F^B$; apply `zip` and get $Z \times F^{B \times ... \times B}$
- Lift Z into the functor F using $Z \Rightarrow F^A \Rightarrow F^{Z \times A}$ (or with $F.\text{pure}$)
- The result is $F^{Z \times B \times ... \times B} \equiv F^{L^B}$
  - For a polynomial L^A, do this to each monomial, then lift to F^{L^B}
  - Note that we could apply `zip` in various different orders
- The traversal order is arbitrary, may be application-specific
- Non-polynomial functors are not traversable (see Bird et al., 2013)
  - Example: $L^A \equiv E \Rightarrow A$; $F^A \equiv 1 + A$; can't have $\text{seq} : L^{F^B} \Rightarrow F^{L^B}$
- All polynomial functors are traversable, and usually in several ways
  - It is still useful to have a type class for traversable functors

Motivation for the laws of the `traverse` operation

- The "law of traversals" paper (2012) argues that `traverse` should "visit each element" of the container L^A exactly once, and evaluate each corresponding "effect" F^B exactly once; then they formulate the laws
- To derive the laws, use the "lifting" intuition for `traverse`,

$$\text{trav} : (A \Rightarrow F^B) \Rightarrow L^A \Rightarrow F^{L^B}$$

Look for "identity" and "composition" laws:

1. "Identity" as `pure` $: A \Rightarrow F^A$ must be lifted to `pure` $: L^A \Rightarrow F^{L^A}$
2. "Identity" as $\text{id}^{\underline{A \Rightarrow A}}$ with $F^A \equiv A$ (identity functor) lifted to $\text{id}^{\underline{L^A \Rightarrow L^A}}$
3. "Compose" $f : A \Rightarrow F^B$ and $g : B \Rightarrow G^C$ to get $h : A \Rightarrow F^{G^C}$, where F, G are applicative; a traversal with h maps L^A to $F^{G^{L^C}}$ and must be equal to the composition of traversals with f and then with $g^{F\uparrow}$

Questions:

- Are the laws for the `sequence` operation simpler?
- Are all these laws independent?
- What functors L satisfy these laws *for all* applicative functors F?

Formulation of the laws for `traverse`

- Identity law: For any applicative functor F,

$$\text{trav}\,(\text{pure}) = \text{pure}$$

$$L^A \xrightarrow[\text{trav}\,(\text{pure}^{\underline{A\Rightarrow F^A}})]{\text{pure}^{\underline{L^A\Rightarrow F^{L^A}}}} F^{L^A}$$

  - Second identity law: $\text{trav}^{\text{Id}}(\text{id}^A) = \text{id}^{L^A}$ is a consequence with $F = \text{Id}$
    - ★ So, we need only one identity law

- Composition law: For any $f^{\underline{A\Rightarrow F^B}}$ and $g^{\underline{B\Rightarrow G^C}}$, & applicative F and G,

$$\text{trav}\,f \circ (\text{trav}\,g)^{F\uparrow} = \text{trav}\left(f \circ g^{F\uparrow}\right)$$

$$\begin{tikzcd}
 & F^{L^B} \arrow[dr, "\text{fmap}_F\left(\text{trav}^G\, g\right)^{\underline{L^B\Rightarrow G^{L^C}}}"] & \\
L^A \arrow[ur, "\text{trav}^F\, f^{\underline{A\Rightarrow F^B}}"] \arrow[rr, "\text{trav}^{F^G}\, h^{\underline{A\Rightarrow F^{G^C}}}"'] & & F^{G^{L^C}}
\end{tikzcd}$$

where $h^{\underline{A\Rightarrow F^{G^C}}} \equiv f \circ g^{F\uparrow}$. (Note: $H^A \equiv F^{G^A}$ is applicative!)

Derivation of the laws for `sequence`

Express $\text{trav}\, f = f^{L\uparrow} \circ \text{seq}$ and substitute into the laws for trav:

- Identity law: $\text{trav}\,(\text{pure}) = \text{pure}^{L\uparrow} \circ \text{seq} = \text{pure}$

$$\begin{tikzcd} & L^{F^A} \arrow[dr, "\text{seq}"] & \\ L^A \arrow[ur, "\text{fmap}_L\ \text{pure}^A"] \arrow[rr, "\text{pure}^{L^A}"'] & & F^{L^A} \end{tikzcd}$$

Naturality law: $\text{seq} \circ g^{F\uparrow L\uparrow} = g^{L\uparrow F\uparrow} \circ \text{seq}$ with $g^{\underline{A \Rightarrow B}}$, mapping $L^{F^A} \Rightarrow F^{L^B}$

- Composition law:

$$\text{trav}\, f \circ (\text{trav}\, g)^{F\uparrow} = f^{L\uparrow} \circ \text{seq} \circ \left(g^{L\uparrow} \circ seq\right)^{F\uparrow}$$

$$= f^{L\uparrow} \circ \text{seq} \circ g^{L\uparrow F\uparrow} \circ \text{seq}^{F\uparrow} = f^{L\uparrow} \circ g^{F\uparrow L\uparrow} \circ \text{seq} \circ \text{seq}^{F\uparrow}$$

$$\text{trav}\left(f \circ g^{F\uparrow}\right) = \left(f \circ g^{F\uparrow}\right)^{L\uparrow} \circ \text{seq} = f^{L\uparrow} \circ g^{F\uparrow L\uparrow} \circ \text{seq}$$

Now omit the common prefix $f^{\cdots} \circ g^{\cdots}$ and obtain: $\text{seq} \circ \text{seq}^{F\uparrow} = \text{seq}$

$$\begin{tikzcd} & F^{L^{G^C}} \arrow[dr, "(\text{seq}^G)^{F\uparrow}"] & \\ L^{F^{G^C}} \arrow[ur, "\text{seq}^F"] \arrow[rr, "\text{seq}^{F^G}"'] & & F^{G^{L^C}} \end{tikzcd}$$

Constructions of traversable and bitraversable functors

Constructions of traversable functors:

1. $L^A \equiv Z$ (constant functor) and $L^A \equiv A$ (identity functor)
2. $L^A \equiv G^A \times H^A$ for any traversable G^A and H^A
3. $L^A \equiv G^A + H^A$ for any traversable G^A and H^A
4. $L^A \equiv S^{A,L^A}$ (recursive) for a bitraversable bifunctor $S^{A,B}$
    - If L^A is infinite, laws will appear to hold but `seq` will not terminate

A bifunctor $S^{A,B}$ is **bitraversable** if `bisequence` exists such that

$$\text{biseq} : S^{F^A,F^B} \Rightarrow F^{S^{A,B}}$$

for any applicative functor F; the analogous laws must hold

Constructions of bitraversable bifunctors:

1. $S^{A,B} \equiv Z$, $S^{A,B} \equiv A$, and $S^{A,B} = B$
2. $S^{A,B} \equiv G^{A,B} \times H^{A,B}$ for any bitraversable G and H
3. $S^{A,B} \equiv G^{A,B} + H^{A,B}$ for any bitraversable G and H

- All polynomial bifunctors are bitraversable
- All polynomial functors, including recursive functors, are traversable

Foldable functors: traversing with respect to a monoid

- Take $F^A \equiv Z$ where Z is a monoid
  - The `zip` operation is the monoid operation $\oplus$
- The type signature of `traverse` becomes $(A \Rightarrow Z) \Rightarrow L^A \Rightarrow Z$
  - This method is called `foldMap`
- The type signature of `seq` becomes $L^Z \Rightarrow Z$
  - This is called `mconcat` – combines all values in L^Z with Z's $\oplus$
- It is convenient to define the `Foldable` type class
  - But it has no laws any more
  - All traversable functors are also foldable
- The `foldLeft` method can be defined via `foldMap` with $Z \equiv (B \Rightarrow B)$:

$$\text{foldl} : (A \Rightarrow B \Rightarrow B) \Rightarrow L^A \Rightarrow B \Rightarrow B$$

Traversable contrafunctors and profunctors are not useful

Traversing profunctors with respect to functors F: effects of F are ignored

- All contrafunctors C^A are traversable w.r.t. applicative profunctors F^A,

$$\text{seq} : C^{F^A} \Rightarrow F^{C^A} \equiv \text{pure}^{C\downarrow} \circ \text{pure}$$

$$C^{F^A} \xrightarrow{\text{cmap}_C \text{pure}_F^A} C^A \xrightarrow{\text{pure}_F^{C^A}} F^{C^A}$$

- But not profunctors that are neither functors not contrafunctors
- Counterexample: $P^A \equiv A \Rightarrow A$; need $\text{seq} : \left(F^A \Rightarrow F^A\right) \Rightarrow F^{A \Rightarrow A}$; we can't get an $A \Rightarrow A$, so the only implementation is to return $\text{pure}_F(\text{id})$, which ignores its argument and so will fail the identity law

Traversing profunctors L with respect to profunctors F: effects are ignored

- Counterexample 1: contrafunctor $L^A \equiv A \Rightarrow R$ and contrafunctor $F^A \equiv A \Rightarrow S$, a `seq` of type $L^{F^A} \Rightarrow F^{L^A}$ must return $1 + 0$
- Counterexample 2: contrafunctor $F^A \equiv (R \Rightarrow A) \Rightarrow S$ and functor $L^A \equiv 1 + A$; `seq` must return $1 + 0$
- So, the result is trivial and probably not useful
  - Laws of traversables allow ignoring the effects of F

Examples of usage

1. Convert foldable data structures to list
2. Fold a tree to aggregate data
3. Decorate a tree with traversal order labels; DFS, BFS
4. Implement scan on a list, as `traverse` with a state monad
5. Use "reverse state monad" to change traversal direction

Naturality with respect to applicative functor as parameter

- The `traverse` method must be "generic in the functor F":

$$\text{trav}^{F,A,B} : (A \Rightarrow F^B) \Rightarrow L^A \Rightarrow F^{L^B}$$

  Which means: The code of `traverse` can only use `pure` and `zip` from F
- A functor F^A is "generic in A": have $\text{fmap} : (A \Rightarrow B) \Rightarrow F^A \Rightarrow F^B$
- "Generic in F" means mapping $(F \Rightarrow G) \Rightarrow \text{trav}^F \Rightarrow \text{trav}^G$ in some way

Mathematical formulation:

- For any natural transformation $F^A \Rightarrow G^A$ between applicative functors F and G such that F.pure and F.zip are mapped into G.pure and G.zip, the result of transforming trav^F is trav^G
  - Such a natural transformation is a morphism of applicative functors
  - Category theory can describe $(F \Rightarrow G) \Rightarrow \text{trav}^F \Rightarrow \text{trav}^G$ as a "lifting"
  - Use a more general definition of category than what we had so far

Exercises

1. Show that any traversable functor L admits a method
$$\text{consume} : (L^A \Rightarrow B) \Rightarrow L^{F^A} \Rightarrow F^B$$
for any applicative functor F. Show that `traverse` and `consume` are equivalent.
2. Show that all the bitraversable laws hold for the bifunctor $S^{A,B} \equiv A \times B$.