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 $List^A \Rightarrow (A \Rightarrow Future^B) \Rightarrow Future^{List^B}$
- Generalize: $L^A \Rightarrow (A \Rightarrow F^B) \Rightarrow F^{L^B}$ 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
- ullet In Scala, we have Future.traverse() that assumes L to be a sequence
 - This is the iconic 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?

Deriving the sequence operation

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

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

- 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:

$$\operatorname{trav} f^{\stackrel{A \Rightarrow F^B}{=}} = \operatorname{fmap} f \circ \operatorname{seq}$$

$$\stackrel{\operatorname{fmap} f}{\longrightarrow} L^{F^B} \stackrel{\operatorname{seq}}{\longrightarrow} F^{L^B}$$

Here F is an arbitrary applicative functor

- Note: We cannot have the opposite transformation $F^{L^B} \Rightarrow L^{F^B}$
 - ightharpoonup Keep in mind the example Future.sequence : List Future ightharpoonup \Rightarrow Future List ightharpoonup
 - Examples: List, all "finite" polynomial functors (see Bird et al., 2013)
 - Non-traversable: $L^A \equiv R \Rightarrow A$; lazy lists ("infinite streams")

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,

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

Look for "identity" and "composition" laws:

- "Identity" as pure : $A \Rightarrow F^A$ must be lifted to pure : $L^A \Rightarrow F^{L^A}$
- ② "Identity" as $id^{A\Rightarrow A}$ with $F^A\equiv A$ (identity functor) lifted to $id^{L^A\Rightarrow L^A}$
- **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 somehow equal to the composition of traversals with f and then with g Questions:

Questions.

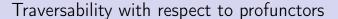
- Are the laws for the sequence operation simpler?
- Are all these laws independent?
- What functors L satisfy these laws for all applicative functors F?

Formulation of the laws for traverse

Derivation of the laws for sequence

Constructions of traversable functors

Traversable profunctors



Foldable functors

Exercises

Show that any traversable functor L admits a method

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

for any applicative functor F. Show that traverse and consume are equivalent.