# Profunctor Optics

## Modular Data Accessors

Matthew Pickering[a], Jeremy Gibbons[b], and Nicolas Wu[a]

a University of Bristol

b University of Oxford

**Abstract** Data accessors allow one to read and write components of a data structure, such as the fields of a record, the variants of a union, or the elements of a container. These data accessors are collectively known as *optics*; they are fundamental to programs that manipulate complex data. Individual data accessors for simple data structures are easy to write, for example as pairs of 'getter' and 'setter' methods. However, it is not obvious how to combine data accessors, in such a way that data accessors for a compound data structure are composed out of smaller data accessors for the parts of that structure. Generally, one has to write a sequence of statements or declarations that navigate step by step through the data structure, accessing one level at a time—which is to say, data accessors are traditionally not first-class citizens, combinable in their own right.

We present a framework for *modular data access*, in which individual data accessors for simple data structures may be freely combined to obtain more complex data accessors for compound data structures. Data accessors become first-class citizens. The framework is based around the notion of *profunctors*, a flexible generalization of functions. The language features required are higher-order functions ('lambdas' or 'closures'), parametrized types ('generics' or 'abstract types') of higher kind, and some mechanism for separating interfaces from implementations ('abstract classes' or 'modules'). We use Haskell as a vehicle in which to present our constructions, but other languages such as Scala that provide the necessary features should work just as well. We provide implementations of all our constructions, in the form of a literate program: the manuscript file for the paper is also the source code for the program, and the extracted code is available separately for evaluation. We also prove the essential properties, demonstrating that our profunctor-based representations are precisely equivalent to the more familiar concrete representations. Our results should pave the way to simpler ways of writing programs that access the components of compound data structures.

**ACM CCS 2012**

- **Software and its engineering → Abstract data types; Patterns; Polymorphism;**

**Keywords** lens, traversal, compositionality

The Art, Science, and Engineering of Programming

**Submitted** November 30, 2016

**Published** April 1, 2017

**DOI** 10.22152/programming-journal.org/2017/1/7

© Matthew Pickering, Jeremy Gibbons, and Nicolas Wu
This work is licensed under a "CC BY 4.0" license.
In *The Art, Science, and Engineering of Programming*, vol. 1, no. 2, 2017, article 7; 51 pages.