FUNCTIONAL PEARL

Applicative programming with effects

CONOR MCBRIDE
University of Nottingham

ROSS PATERSON
City University, London

Abstract

In this paper, we introduce Applicative functors—an abstract characterisation of an applicative style of effectful programming, weaker than Monads and hence more widespread. Indeed, it is the ubiquity of this programming pattern that drew us to the abstraction. We retrace our steps in this paper, introducing the applicative pattern by diverse examples, then abstracting it to define the Applicative type class and introducing a bracket notation which interprets the normal application syntax in the idiom of an Applicative functor. Further, we develop the properties of applicative functors and the generic operations they support. We close by identifying the categorical structure of applicative functors and examining their relationship both with Monads and with Arrows.

1 Introduction

This is the story of a pattern that popped up time and again in our daily work, programming in Haskell (Peyton Jones, 2003), until the temptation to abstract it became irresistable. Let us illustrate with some examples.

Sequencing commands One often wants to execute a sequence of commands and collect the sequence of their responses, and indeed there is such a function in the Haskell Prelude (here specialised to IO):

```
sequence :: [IO a] → IO [a]
sequence []       = return []
sequence (c : cs) = do
  x  ← c
  xs ← sequence cs
  return (x : xs)
```

In the $(c : cs)$ case, we collect the values of some effectful computations, which we then use as the arguments to a pure function (:). We could avoid the need for names to wire these values through to their point of usage if we had a kind of 'effectful application'. Fortunately, exactly such a thing lives in the standard Monad library: