Purely Functional Effect-Handling

Henry Blanchette

Contents

Chapter 1

Introduction

1.1 Outline

1. Definition of a simple, typed lambda calculus
2. Context and definition of *effects* in programming languages
3. Examples of common effects
   (a) IO
   (b) mutable data, state
   (c) exception
   (d) nondeterminism
   (e) non-termination / non-totality
   (f) reader, writer, output
   (g) continuation
4. Explanation of how effects are handled in ML's style
   (a) call-by-value at runtime
   (b) examples of effects: IO, exceptions
   (c) effects are untyped
   (d) simple exception-handling system (type/catch)
5. Explain why this style is not purely functional
   (a) exposes *side-effects*, which are implicit effects not accessible by the programming language

Chapter 2

Monadic Effects

2.1 Outline

1. Definition and context for monads in category theory and computer science
2. Explanation of how monads can model effects in general
3. Demonstration of constructing and using the stateful monad, building it up from scratch in Haskell or Haskell-like psuedocode
   (a) Explain the Functor, Applicative, and Monad typeclasses, and how they build up the mathematical definition of a monad
4. Outline some problems with monadic effects
   (a) different effects are not composable
   (b) paper: *The Awkward Squad*

Chapter 3

Algebraic Effect Handlers

3.1 Outline

1. Definition and context for algebraic effect handlers
2. Explanation of the Eff programming languages approach to implementation
   (a) subset of semantics
   (b) examples
3. Comparison of algebraic effect handlers to monadic effects approach
   (a) alebraic effect handlers allow for effects and handlers to be defined separately
   (b) algebraic effect handlers and generally composable, however they must be carefully handled to match their internal composition
   (c) Eff does not expressively type the effect system, and many features are untyped

Chapter 4

Monad Transformers

4.1 Outline

1. Definition and context for monad transformers
   (a) provides a framework for sequencing different monadic effects
   (b) introduces a higher-order monad that acts as an effect for sequencing lower monadic effects
2. Brief explanation of monad transformers implementation in Haskell
3. Discussion of advantages and disadvantages of monad transformers
   (a) facilitates general sequencing of different effects at the same level
   (b) rigorously typed
   (c) does not facilitate general composition of different effects; effects still must be handled at the level they are introduced
   (d) are clunky to program with

Chapter 5

Freer Monadic Effects

5.1 Outline

1. Consideration of the problem of *composing monads*; no systems so far have facilitated this while maintaining type-checking
2. Definition and context for freer monads in category theory and computer science, in relation to monads as previously discussed in chapter 2
3. Explanation of how freer monads can model effects
   (a) left Kan extension (Lan)
   (b) act as a sort of monadic implementation of algebraic effect handlers
4. Demonstration of stateful freer monad, paralleling monadic example from chapter 2
5. Discussion of advantages and disadvantages of freer monadic effects
   (a) fully-typed system for algebraic effect handlers; all the advantages of algebraic effect handlers
   (b) facilitates generally composable effects
   (c) not implementable in non-dependently typed languages like Haskell, OCaml, SML, etc. since it requires extensive type-level manipulation
   (d) potentially very user-unfriendly, incurring many of the original problems with monadic effects

Chapter 6

Effigy

1. Explanation of motivations for a new language, incorporating the ideas of previous chapters
   (a) provides fully-type freer monadic effects (implemented in a language like Coq)
   (b) user-friendly interfacing with effects, that degenerates to familiar monadic-effect syntax for single-level effects
2. Specification of Effigy's syntax
3. Correctness proofs for Effigy's implementation
4. Demonstration of familiar examples in Effigy programs