TYPE-CHANGING REFACTORINGS IN HASKELL

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF PHD.

Draft on June 9, 2016

By

Stephen Adams

June 2016

Abstract

This mini-thesis tells you all you need to know about...

Acknowledgements

I would like to thank...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This is probably going to be written last.

Chapter 2

Refactoring Haskell in HaRe

History of HaRe and the update to the GHC. Maybe some architectural details.

Chapter 3

Data refactoring in a functional context

Introduce some of the ideas of functional data refactorings. Talk about simple refactoring examples like Hughes lists and Maybe generalising to MonadPlus.

Chapter 4

Generalising Monads to Applicative

In their 2008 functional pearl "Applicative programming with effects" Conor McBride and Ross Paterson introduced a new typeclass that they called Idioms but are also known as Applicative Functors (Mcbride and Paterson 2008). Idioms provide a way to run effectful computations and collect them in some way. They are more expressive than functors but more general than Monads, further work was done in (Lindley, Wadler and Yallop 2011) to prove that Idioms are also less powerful than Arrows.

Applicative functors were implemented in the GHC as the typeclass *Applicative*. An interesting part of the history of the GHC is that despite McBride and Paterson proving in their original functional pearl that all monads are also applicative functors, however, the GHC did not actually require instances of monad to also be instances of Applicative until GHC's 7.10.1 release (GHC 2015). Now that every monad must also be an applicative functor there now exists a large amount of code which could be rewritten using the applicative operators rather than the monadic ones.

This chapter will discuss the design and implementation of a refactoring which

will automatically refactor code written in a monadic style to use the applicative operators instead. Section 4.1 is a brief overview of the *Applicative* typeclass's operators, section 4.2 will discuss the applicative programming style and, in general, how programs are constructed using the applicative operators, next, section 4.3 will cover some common applications of this refactoring, section 4.4 will specify the refactoring itself, section 4.5 covers the preconditions of the refactoring, finally section 4.6 outlines other refactorings that may be used in conjunction with the generalising monads to applicative refactoring and some possible variations of this refactoring.

4.1 The Applicative Typeclass

4.2 The Applicative Programming Style

4.3 Applications of the Refactoring

4.4 Refactoring Monadic Programs to Applicative

4.5 Preconditions of the Refactoring (When is a Monad actually a Monad?)

4.6 Variations and Related Refactorings

Chapter 5

Mysterious third chapter of contribution

More research goes here.

Chapter 6

Related work

Mention type and transform, Meng Wang's paper...

Chapter 7

Conclusion

Well its done..

Bibliography

GHC (2015). 1.5. release notes for version 7.10.1. `https://downloads.haskell.org/~ghc/7.10.1/docs/html/users_guide/release-7-10-1.html`.

Lindley, S., Wadler, P. and Yallop, J. (2011). Idioms are oblivious, arrows are meticulous, monads are promiscuous. *Electronic Notes in Theoretical Computer Science*, 229(5), pp. 97–117.

Mcbride, C. and Paterson, R. (2008). Applicative programming with effects. *J Funct Program*, 18(1), pp. 1–13.