The Practical Guide to Levitation

Ahmad Salim Al-Sibahi

Advisors:
Dr. Peter Sestoft
& David R. Christiansen
Submitted: September 1, 2014

IT University
of Copenhagen

Abstract

Goal: Implementation of levitation in a realistic setting, with practical performance benefits.

Contents

Chapter 1

Introduction

1.1 Context

- Generic Programming
- Descriped Types in Dependently-Typed Programming Languages
- The Gentle Art of Levitation
- Leveling-up dependent types

1.2 Problem Definition

- Lack of efficiency in type checking and run-time due to resulting large, data structures
- Lack of use in a practical programming setting which has type-classes
- Need for matching on the level of types (type-casing)

1.3 Aim and Scope

- Aim
  - To provide an efficient and practical implementation of described types/levitation in Idris that is in synergy with existing language features such as type classes.

- Scope
  - To not extend underlying type theory, such as to increase support for further inductive-datatypes
  - To not focus on improving the existing representation significantly, and to choose the best fitting description there is
  - To not try and support all existing Idris language features, such as implicits, codata or automated proof search
  - To find an efficient run-time representation of the existing data-structures via partial compilation
  - To specialise generic functions over a description, to yield functions with good performance metrics (near hand-written)
  - To show realistic examples using generic functions such as some form for type-class deriving and traversal/querying

1.4 Significance

- To show the applicability of generic programming using described types to real-world programs
- To show that generic programming using described types can be a viable option to reduce boiler-plate without significant cost in performance
- To highlight possible problems in the type-class deriving and generic traversal/querying examples which can be considered future work

1.5 Overview

- Chapter 2 highlights the recent developments in generic programming, specifically focusing on described types in dependently-typed programming languages.
- Chapter 3 investigates techniques for partial evaluation of functions and specialisation of data-type constructors.

- Chapter 4 discusses further on how the general ideas of levitation, are implemented specifically in an Idris context. Furthermore it discusses, how some of the existing structures, can be changed to have descriptions.
- Chapter 5 continues the discussion from Chapter 4, this time concentrating on how optimizations are made to ensure that described types do not carry a large run-time overhead.
- Chapter 6 shows two realistic examples for using levitation in Idris, namely type-class deriving and generic traversal/querying.
- Chapter 7 compares the performance of hand-written data types and functions, with the performance of generic programs on described types.
- Chapter 8 discusses what challenges still lie ahead for practical usage of generic programming using described types, and concludes on the results of this work.

Chapter 2

Generic Programming

2.1 The Generic Structure of Inductive Data Types

How Generic Programming generally works.

2.2 The Importance of Genericity in Dependently-typed Languages

The similarity of structure and various slightly-different indexing of types.

2.3 The (Mostly) Gentle Art of Levitation

The elegance of a complete theorem for both ordinary and generic programming. Highlighting of possible issues with performance.

Chapter 3

Partial Evaluation

3.1 Functions and Constant Inputs

General introduction about partial evaluation.

3.2 Binding-time Analyses of Programs

Finding the relevant constant parts of the program.

3.3 Specialisation as a Form of Optimization

Performance benefits of program specialisation. Pitfalls.

Chapter 4

Levitating Idris

4.1 A Pragmatic Implementation of Levitation

How the general concept of levitation was transferred to Idris.

4.2 Description Synthesis from Ordinary Data Declarations

How ordinary data-declarations are synthesized to levitational descriptions.

Chapter 5

Optimizing Idris for Flight

5.1 Specialising Constructors for Specific Types

How generalized constructors of described types, are specialised as ordinary data structures.

5.2 Online Erasure of Unused Arguments

How some type infromation is to be erased at compile time to reduce elaboration overhead. Very hypothetical.

5.3 Static Initialization of Generic Functions

How algorithms that are dependent on the generic structure of a data-type are optimized. Discuss benefits of having a JIT/Profiling information for future work.

Chapter 6

Practical Examples

6.1 Generic Deriving

Examples of generic deriving of algorithms like decidable equality, pretty printing and possibly eliminators via generic structure.

6.2 Uniplate for Idris

A version of the Uniplate library for Idris based on `http://community.haskell.org/~ndm/uniplate/` *and* `http://www-ps.informatik.uni-kiel.de/~sebf/projects/traversal.html` *. This is useful for traversing structures in a generic fashion and especially when dealing with small changes in large data structures (such as compiler ADTs)*

Chapter 7

Evaluation

Chapter 8

Discussion

8.1 Future Work

8.2 Conclusion