Generic Programming in Scala
Project Proposal

Carlos Tomé Cortiñas Matthew Swart Renate Eilers

December 8, 2016

1 Introduction

Oftentimes, programmers find themselves writing very similar programs for various datatypes. Consider, for instance, functions for calculating the size of a datastructure such as a list or a tree. On the surface these programs will look quite different, as they are designed to work on different datatypes. On a closer look, however, they are actually very similar. From the wish to quit repeatedly writing more-or-less the same functions, the idea of *datatype-generic programming* was born. Datatype-generic programming entails the writing of functions on the *structure* of datatypes rather than on concrete instantiations. This allows programmers to write a single function to work on an entire class of datatypes. The main idea here is that most datastructures can be translated to a combination of basic structural elements, such as sums and products. Abstracting over the specifics of a datastructure allows us to see that many computations simply consists of transforming one structure into another or breaking it down into a single value. If such functions are then defined as operations on the structures of these aforementioned *standard* types, a lot of code duplication can be foregone.

According to Gibbons[**?**], a language should allow parameterisation in three forms in order to support datatype-generic programming, :

- By *element type*: for instance, allow lists datatypes to parameterise over the items they can contain rather than having dedicated integer lists and character lists.
- By the *body* of the computation: consider higher order functions such as `map` and `fold`, allowing a multitude of functions to be applied to a single datatype.
- By *shape* of the computation: folding over a tree or a list structure is essentially a process of recursing over a structure following the datatype definition, and replacing type constructors by given arguments along the way. In order to support datatype-generic programming, a language should provide the functionality to parameterise over these shapes.

Historically, Haskell has been the popular choice for datatype-generic programming, as it supports all the above forms of parameterisation (Oliveira and

Gibbons,[?]). More recently it has been argued that the Scala programming language is a fairly reasonable choice as well.

Scala[?] is a language incorporating features from both the functional and object-oriented paradigm. It runs on the Java platform, and operates with all Java libraries seamlessly. In Scala, every value is an object, and every operation is a method call. Novel constructs such as *traits* and *mixin composition* allow for more advanced object composition than most other languages according to its creator, Martin Odersky[?]. Scala facilitates the parameterisation of shape through higher-kinded types, generics take care of parameterisation by type, and parameterisation of computation can be handled using higher-order functions[?].

In the following sections we present a problem within the area of generic programming in Scala and propose a methodology for tackling it. Finally, a tentative planning for the proposed work is supplied.

2 Problem

The aim of this project is to investigate Scala's potential in the domain of datatype-generic programming. For a long time, Haskell has been the go-to language for the exploration of datatype-generic programming[?], which has resulted in the development of a considerable number of Haskell libraries for this purpose (e.g. GHC.Generics and Uniplate). Each of these libraries comes with its own pros and cons.

Scala has only recently been suggested as a language for the domain of datatype-generic programming, which leaves the area less developed: only one library supporting datatype-generic programming exists. This libary, *Shapeless*[1][?], was originally written by Miles Sabin. According to its author, "Shapeless is a typeclass and dependent-type based generic programming library for Scala[2]". Similar to the existing Haskell libraries, we expect Shapeless to come with its own set of up- and downsides. We aim to investigate these strong and weak points, see where there is room for improvement in the domain of datatype-generic programming in Scala, and ultimately to implement these findings.

3 Methodology

A thorough comparison of various Haskell libraries for datatype-generic programming has been done by Rodriguez et al[?]. In this work, the authors collected from literature a series of typical scenarios where datatype-generic programming is used. Based on these findings, they identified the features that are needed in a library in order to be able to tackle such a scenario. Finally, they used these features as criteria for evaluating each of the Haskell libraries.

The starting point for this project is to examine how well the current approach for datatype-generic programming in Scala, and its implementation in

[1] `https://github.com/milessabin/shapeless`

[2] In the context of Scala dependent types refer to path-dependent types, which are more limited than full-blown dependent types

Types

- Universe size
- Subuniverse

Expressiveness

- First-class generic functions
- Abstraction over type constructors
- Separate compilation
- Ad-hoc definitions for datatypes
- Ad-hoc definitions for constructors
- Extensibility
- Multiple arguments
- Multiple type representation arguments
- Constructor names
- Consumers, transformers and producers

Usability

- Performance
- Portability
- Overhead of library use
- Ease of use and learning

Figure 1: Criteria overview

the library Shapeless, scores on the criteria defined by [**?**] (Figure **??**, a precise definition can be found in the paper). In order to do so, we will closely follow the methodology presented by the authors, and port the collection of programs they use for benchmarking to Scala.

Once we have clearly identified the approach to datatype-generic programming taken by Shapeless, and have established its strengths and weaknesses, we can proceed to investigate how this work can be improved upon. Even in Haskell, for which a great amount of research in the area has been done, there is no clear agreement on how a library has to be designed in order to fulfill all the proposed criteria. Different approaches embrace different trade-offs. We do not expect that the only existing approach in Scala will fit all criteria perfectly. Upon figuring out which criteria Shapeless lacks, we will implement an approach providing exactly these criteria.

However, if it is the case that the Shapeless library does fit all the criteria, our contingency plan is to investigate alternative approaches to datatype-generic programming in Scala, and implement one, even though it will not necessarily improve on the existing one.

Regardless of Shapeless' scores on the criteria, the expected result of this project is a new Scala library for datatype-generic programming.

A good reference for starting the investigation, can be found in [**?**], where the authors explore how some of the main approaches to datatype-generic programming in Haskell can be translated to Scala.

4 Planning

Task	**Deadline**
Draft research proposal	24-11-2016
Research proposal	2-12-2016
Scala proficiency, Getting acquainted with the internals of the Shapeless library	9-12-2016
Benchmarking Shapeless, Report	15-12-2016
Prototype, Report	12-1-2017
Finish report	19-1-2017
Presentation	26-1-2017