HERMIT: An Equational Reasoning Model to Implementation Rewrite System for Haskell*

Andrew Gill

Information Technology and Telecommunication Center
Department of Electrical Engineering and Computer Science
The University of Kansas, USA
`andygill@ku.edu`

Abstract

HERMIT is a rewrite system for Haskell. Haskell, a pure functional programming language, is an ideal candidate for performing equational reasoning. Equational reasoning, replacing equals with equals, is a tunneling mechanism between different, but equivalent, programs. The ability to be agile in representation and implementation, but retain equivalence, brings many benefits. Post-hoc optimization is one obvious application of representation agility.

What we want to explore is the mechanization of rewriting, inside real Haskell programs, enabling the prototyping of new optimizations, the explicit use of types to direct transformations, and perform larger data refinement tasks than are currently undertaken. Paper and pencil program transformations have been published that improve performance in a principled way; indeed some have turned the act of program transformation into an art form. But there is only so far a sheet of paper and a pencil can take you. There are also source code development environments that provide support for refactoring, such as HaRe. These work at the syntactical level, and Haskell is a large and complex language. What we want is mechanization, for examples that are currently undertaken by hand, and for examples that are challenging to perform using current development environments.

In this talk, we overview HERMIT, the Haskell equational reasoning model to implementation tunnel. HERMIT operates at the Glasgow Haskell compiler's Core level, deep inside GHC, where type information is easy to obtain, and the language being rewritten is smaller. HERMIT provides three levels of support for transformation and prototyping: a strategic programming base with many typed rewrite primitives, a simple shell that can used to interactively request rewrites and explore transformation possibilities, and a batch language that can mechanize focused, and optionally program specific, optimizations. We will demonstrate all three of these levels, and show how they cooperate.

The explicit aim of the HERMIT project is to explore the worker/wrapper transformation as a specific way of mechanizing data refinement. HERMIT has been successfully used on small examples, efficient `reverse`, tupling-`fib`, and many other examples from the literature. We will show two larger and more interesting examples of program transformation using HERMIT. Specifically, we will show the mechanization of the making a century program refinement pearl, originally by Richard Bird, and the exploration of datatype alternatives in Graham Hutton's implementation of John Conway's Game of Life.

1998 ACM Subject Classification D.3.4 Translator writing systems and compiler generators

Keywords and phrases Program Transformation, Equational Reasoning, Optimization

Digital Object Identifier 10.4230/OASIcs.WPTE.2014.1

Category Invited Talk

* This material is based upon work supported by the National Science Foundation under Grant No. 1117569.