Towards Open Type Functions for Haskell
September 3, 2007

Tom Schrijvers*[1], Martin Sulzmann[2], Simon Peyton Jones[3], and Manuel Chakravarty[4]

[1] K.U.Leuven, Belgium (`tom.schrijvers@cs.kuleuven.be`)
[2] National University of Singapore (`sulzmann@comp.nus.edu.sg`)
[3] Microsoft Research Cambridge, UK (`simonpj@microsoft.com`)
[4] University of New South Wales (`chak@cse.unsw.edu.au`)

Abstract. We report on an extension of Haskell with type(-level) functions and equality constraints. We illustrate their usefulness in the context of phantom types, GADTs and type classes. Problems in the context of type checking are identified and we sketch our solution: a decidable type checking algorithm for a restricted class of type functions. Moreover, functional dependencies are now obsolete: we show how they can be encoded as type functions.
This paper is submitted to the Implementing Functional Languages workshop, Sept 2007 (IFL07).

1 Introduction

Experimental languages such as ATS [6], Cayenne [1], Chameleon [25], Epigram [15] and Omega [21] equip the programmer with various forms of "type functions" to write entire programs on the level of types. In the context of Haskell, there are two distinct languages extensions that that support such type-level computation: *functional dependencies* which are well established [12], and *associated types* which are a more recent experiment [5]. In this paper, we make the following contributions:

- We generalise the so-called "associated type synonyms" [5] by decoupling them from `class` declarations, thereby allowing us to define stand-alone type functions (Section 2). We give examples which show the usefulness of stand-alone type functions in combination with GADTs and phantom types.
- It turns out that pure type *inference* for our extended language is very easy. However, in the presence of user-supplied type signatures (which are ubiquitous in Haskell) and GADTs, the type *checking* problem becomes unexpectedly hard. We identify the problem and sketch our solution (Section 3). This is the main technical contribution of the paper.
- We show that type functions are enough to express all programs involving functional dependencies, although the reverse is problematic (Section 4). Other related work is discussed in Section 5.

* Post-doctoral researcher of the Fund for Scientific Research - Flanders.