Fun with type functions

Oleg Kiselyov Simon Peyton Jones Chung-chieh Shan

May 3, 2010

Abstract

Tony Hoare has always been a leader in writing down and proving properties of programs. To prove properties of programs automatically, the most widely used technology today is by far the ubiquitous type checker. Alas, static type systems inevitably exclude some good programs and allow some bad ones. Thus motivated, we describe some fun we have been having with Haskell, by making the type system more expressive without losing the benefits of automatic proof and compact expression. Specifically, we offer a programmer's tour of so-called *type families*, a recent extension to Haskell that allows functions on types to be expressed as straightforwardly as functions on values. This facility makes it easier for programmers to effectively extend the compiler by writing functional programs that execute during type-checking.

This paper gives a programmer's tour of type families as they are supported in GHC.

Source code for all the examples is available at `http://research.microsoft.com/~simonpj/papers/assoc-types/fun-with-type-funs/fun-with-type-funs.zip`

1 Introduction

The type of a function specifies (partially) what it does. Although weak as a specification language, static types have compensating virtues: they are

- *lightweight*, so programmers use them;
- *machine-checked* with minimal programmer assistance;
- *ubiquitous*, so programmers cannot avoid them.

As a result, static type checking is by far the most widely used verification technology today.

Every type system excludes some "good" programs, and permits some "bad" ones. For example, a language that lacks polymorphism will reject this "good" program:

```
f :: [Int] -> [Bool] -> Int
f is bs = length is + length bs
```