System F with Type-Equality Coercions (Invited TFP/Types 2006 Talk)

Simon Peyton Jones[1]
joint work with
Martin Sulzmann[2] and Manuel Chakravarty[3]

[1] Microsoft Research
[2] University of Singapore
[3] University of New South Wales

Abstract

GHC has a problem. It uses System F, extended with algebraic data types and case-expressions, as its typed intermediate language. But Haskell has incubated a variety of extensions that are hard to translate into this language: in particular, functional dependencies, generalised algebraic data types (GADTs), and associated types. When I added GADTs to GHC, I found I had to extend GHC's intermediate language with GADTs as well, although GADTs are arguably rather a sophisticated addition. But when it came to associated types, it was clear that I could not throw them in too without making the intermediate language unbearably baroque.

In this talk I'll present a solution to this dilemma. There is no need to add GADTs to System F. Instead, we add explicit type-equality coercions, and use them as witnesses to justify explicit type-cast operations. These coercions are treated just like types: we have coercion abstraction and application just like type abstraction and application, and we have an erasure result that allows us to discard all coercions and casts before execution.

This single mechanism allows a very direct encoding of associated types and GADTs, and allows us to deal with some exotic functional-dependency programs that GHC currently rejects on the grounds that they have no System-F translation. The language is rather general, though, and may have other applications.

Like many good ideas, it all seems quite obvious in retrospect; we are simply deploying an approach that is well-studied in the TYPES community. However, it's a language with very immediate practical application and many of the details are interesting in their own right.