Preliminary Thoughts On Injective Monadic Pure Type Systems

WJL

Introduction

I'm trying to extend a type checking algorithm for injective pure type systems to monadic type systems. The star and box types of both computations and values can be used to construct an injective type system. The axioms can reflect those of the lambda cube, but with one for values and one for computations. If the rules, accepting at any position a computation or value sort, have the second and third sort equal, then the type system is injective. The monadic operations are checkable in the typing rules, but the classification algorithm has issues. The operations are only shown typed while applied to an argument. I attempted to use the classification rules for application, lambdas, and dependent products to inform rules for the monadic operations. I didn't get anywhere with this. I was especially bother by the thought that I might need two PTS rules to type a constructor by itself, and not really need it in the applied form. I decided to proceed with monadic operations that are not already applied and see how that goes. My gut feeling is that they will be easier to work with. I suppose that each monad rule could just have the base sort of computations, but it feels weird. The other thing is that haskell does this in a similar way. The unsafe monads, namely ST and IO, are implemented like regular types and stuff. Then, it uses unsafe functions or the ffi or the interpreter idk to do special priviledged things.

Monad typing rules according to Barthe 1998

$$\frac{\Gamma \vdash A : *^x}{\Gamma \vdash \mathbf{M}A : *^c} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash \mathbf{M}A : s}{\Gamma \vdash \mathbf{unit}M : \mathbf{M}A}$$

$$\frac{\Gamma \vdash M : \mathbf{M}A \qquad \Gamma, x : A \vdash N : \mathbf{M}B}{\Gamma \vdash \mathbf{let} x : A, = M \mathbf{in} N : \mathbf{M}B}$$

Although the work describes these as type constructors they are only shown in typing rules applied to another term. So lets isolate them. The rule for the monad type constructor implies that:

$$\frac{}{\Gamma \vdash \mathbf{M} : \Pi\alpha : *^x.\alpha \to *^c}$$

I am not sure I can type this dependence on both sorts, computations and values. Split them up.

$$\frac{}{\Gamma \vdash \mathbf{M} : \Pi\alpha : *^v.\alpha \to *^c} \qquad \frac{}{\Gamma \vdash \mathbf{MT} : \Pi\alpha : *^c.\alpha \to *^c}$$

That looks like a regular monad, and a monad transformer. That's kinda neat. First class monad transformers. So, all chains of monad transformers could hypothetically be terminated in a value monad. Or not, which is fine but might be interesting. There is obviously an identity monad for both monad transformers and monads, but the one for monads might be more interesting. Whatever the last actions of a properly terminating program, they end with a pure function returning a value. Maybe this can be exploited as some "main function", maybe its loader, or whatever. I have no clue.

Anyway, we can now make operations for both types.