Modelling Haskell Programs using the Freer Monad

Jan Christiansen
Hochschule Flensburg
Fachbereich 3
jan.christiansen@hs-flensburg.de

Sandra Dylus
CAU Kiel
Institut für Informatik
sad@informatik.uni-kiel.de

When we want to prove statements about a Haskell program in an interactive theorem prover like Agda or Coq, we have to transform the Haskell program into an Agda/Coq program. As Agda/Coq programs have to be total and Haskell programs are often not, we have to model partiality explicitly in the target language. A natural way of modeling partiality is the maybe monad. Instead of using the maybe monad explicitly, it is quite appealing to use a generic monadic embedding. By instantiating the monadic program with a concrete monad instance we can choose the model we would like to prove the statement in. For example, when we use the identity monad, we are working in a total world while the maybe monad allows us to model partiality. In preceding work by Abel et al. [2005] this kind of model has been used to reason about existing Haskell programs in Agda. However, this approach does not work in Coq as Coq rejects the definitions of monadic data types that are required. Similarly we cannot use this approach in Agda these days because Agda has added the same restriction that is enforced by Coq.

As a solution to this problem we present an approach that is based on the free monad — more specifically the freer monad. The free monad extracts the essence of a monad into a data type and allows for specific monadic effects by parametrising over a type constructor. By using the free monad we get function and data type definitions that can be used to model both worlds. While the definition of the free monad is not allowed as well, a derivative, the freer monad, is accepted by current versions of Agda and Coq. In contrast to preceding work our model even allows to prove statements that hold for all monads with a single proof.

References

Andreas Abel, Marcin Benke, Ana Bove, John Hughes, and Ulf Norell. Verifying haskell programs using constructive type theory. In *In Haskell'05*. ACM Press, 2005.