

Modular Modelling with Monads

Christoph Lüth

FB 3 — Mathematik und Informatik

Universität Bremen

<http://www.informatik.uni-bremen.de/~cx1>

November 26, 2005

One useful application of category theory in computer science are *computational monads*, which are used to model diverse computational effects (such as stateful computations, exceptional behaviour or non-determinism). This goes back to Moggi [6], and is for example used to great effect in the pure functional language Haskell [8, 11]. In this talk, we use the same technique to model computational effects in higher-order logic, and in particular the theorem prover Isabelle [7], thus shallowly embedding a small imperative language embedded shallow into Isabelle. The talk consists of three parts: we first recall the basic notions of monads, then show how the imperative language is built from three basic ingredients, and finally discuss the implementation.

Monads

A *monad* is the categorical modelling of an algebraic theory (that is, a set of operations and equations on them). One attraction of monads is that they come with a rich categorical model theory, giving a uniform treatment of both algebras and terms. The latter are given by the *Kleisli category* of a monad; this is where the computations modelled by the monad live. An adjunction between the base category and the Kleisli category gives us a notion of lifting every function from the base to a pure computation.

To combine monads, we can (under some mild preconditions on the base category) use *colimits* [3]. However, as the construction of colimits in general is quite intricate, two special cases of combinations of monads which have been considered: the coproduct, in particular of layered monads [5], and the tensor [1]. The former specifies that the sequential order of computations from the monads in question must be maintained, the latter specifies that they do not interact at all and hence their sequential order can be exchanged freely.

Modelling an Imperative Language

Our language is made up from three ingredients. Firstly, we have a core language with *iteration* and *case distinction*. The case distinction is given by coproducts from the base, and the iteration requires the existence of fixpoints.

Stateful computations can be added in two ways, either by an axiomatic description [9] or constructively by state threads [4]. The former allows us to model typed references and is more general, but may lead to arguments about

consistency; the latter is more restrictively typed, but constructive and admits more equations.

Finally, exceptional behaviour is modelled by the exception monad, which is just the coproduct with a (fixed) set of exceptions.

Modelling in Isabelle

The modelling of monads in Isabelle uses a recent extension of Isabelle with parameterised theories and theory morphisms [2]. With these, we can model the monads as described above as parameterised theories and their relation by theory morphisms between them.

Syntax plays an important rôle when it comes to interactive theorem proving, as a good concise notation can make complicated proofs tractable. Therefore, we introduce a number of syntactic conventions (some as in Haskell) for monadic expressions. Moreover, we need a logic in which to reason about these; instead of e.g. Hoare logic [10] we choose a direct approach which lets us use Isabelle's automatic proof procedures.

Conclusions

We have shown how to concisely model a core imperative language which is short but, we argue, covers the essential features of most imperative languages. This can be implemented in a theorem prover to yield a shallow embedding of the core imperative language into higher-order logic; this in turn can be used to e.g. derive verification conditions, or prove program transformations. We plan to use our model in real-live program verification projects in the nearer future.

References

- [1] M. Hyland, G. Plotkin, and J. Power. Combining computational effects: Commutativity and sum. In *TCS 2002, 2nd IFIP International Conference on Computer Science*, Montreal, 2002.
- [2] E. B. Johnsen and C. Lüth. Theorem reuse by proof term transformation. In K. Slind, A. Bunker, and G. Gopalakrishnan, editors, *International Conference on Theorem Proving in Higher-Order Logics TPHOLs 2004*, volume 3223 of *Lecture Notes in Computer Science*, pages 152–167. Springer Verlag, Sept. 2004.
- [3] G. M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves and so on. *Bulletins of the Australian Mathematical Society*, 22:1– 83, 1980.
- [4] J. Launchbury and S. P. Jones. Lazy functional state threads. In *Proc. ACM Conference on Programming Languages Design and Implementation*, 1994.
- [5] C. Lüth and N. Ghani. Composing monads using coproducts. In *International Conference on Functional Programming ICFP'02*, pages 133– 144. ACM Press, Sept. 2002.

- [6] E. Moggi. Computational lambda-calculus and monads. In *Fourth Annual Symposium on Logic in Computer Science*. IEEE, Computer Society Press, June 1989.
- [7] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.
- [8] S. Peyton Jones, editor. *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press, 2003.
- [9] G. Plotkin and J. Power. Notions of computation determine monads. In M. Nielsen and U. Engberg, editors, *Proc. FOSSACS 2002*, Lecture Notes in Computer Science 2303, pages 342– 356, 2002.
- [10] L. Schröder and T. Mossakowski. Monad-independent Hoare logic in Haskell. In M. Pezze, editor, *Fundamental Approaches to Software Engineering (FASE 2003)*, volume 2621 of *Lecture Notes in Computer Science*, pages 261–277. Springer Verlag, 2003.
- [11] P. Wadler. How to declare an imperative. *ACM Computing Surveys*, 29(3):240– 263, 1997.