Monadic reflection in Lax logic

Tristan Crolard
LACL – University of East Paris
crolard@u-pec.fr

Abstract

We revisit, from a logical standpoint, Filinski's implementation of Moggi's monadic reflection using delimited control operators. We show that monadic reflection can be applied in Lax logic to provide some computational content to various axioms of the form $\Diamond\varphi \Rightarrow \varphi$ where $\Diamond$ is a representable monad. We also discuss the meaning of this axiom for some standard monads.

In the quest for a logical understanding of delimited control, we revisit Filinski's encoding of Moggi's monadic reflection [9] using **shift**/**reset** [5]. Indeed, since representing monads is a major application of delimited control, it seems reasonable to look first for a the logical interpretation in this framework (where the full type system from [3] is not required).

As studied in [1], Moggi's computational types correspond, through the formulas-as-types interpretation, to formulas of a modal logic called Lax logic [4] (originally due to Curry [2]) . We show in this paper that monadic reflection can also be applied to provide some computational content to axioms of the form $\Diamond\varphi \Rightarrow \varphi$ where $\Diamond$ is a representable monad. Note that we shall not consider here the equational laws of monads, so $\Diamond$ is, strictly speaking, just a definable modality together with two proof-terms $\mathit{unit} : \varphi \Rightarrow \Diamond\varphi$ and $\mathit{bind} : (\varphi \Rightarrow \Diamond\psi) \Rightarrow \Diamond\varphi \Rightarrow \Diamond\psi$. However, we shall see that correctness can still be enforced by the presence of dependent types, even if the three monad equations do not hold.

As in [5], the source language is the typed λ-calculus with two operators $[\cdot]$ and $\mu(\cdot)$, called *reify* and *reflect*, and typed as follows:

$$\frac{\Gamma \vdash t : \varphi}{\Gamma \vdash [t] : \Diamond\varphi}\ (\mathit{reify}) \qquad\qquad \frac{\Gamma \vdash t : \Diamond\varphi}{\Gamma \vdash \mu(t) : \varphi}\ (\mathit{reflect})$$

As shown by Filinski [5], monadic reflection is validated both by the monadic translation and the *cps*-translation. We shall consider only the latter, and focus on the continuation monad ∇ defined as $\nabla\varphi = \forall\alpha\,((\varphi \Rightarrow \Diamond\alpha) \Rightarrow \Diamond\alpha)$. Note that this continuation monad, which suggested in a footnote of [5], is polymorphic in the answer type.

To be more specific, let us assume that our target system is a formulation of second-order arithmetic (such as Leivant's **M2L** [8] or, equivalently, Krivine's $\mathbf{AF}_2$ [7]). However, for simplicity, we shall only consider source terms typable is the first-order fragment. The translation of formulas is defined inductively as follows (where σ is atomic):

$$\sigma^{\nabla} \equiv \sigma \qquad (\varphi \wedge \psi)^{\nabla} \equiv \varphi^{\nabla} \wedge \psi^{\nabla} \qquad \forall\vec{t}(\varphi \Rightarrow \psi)^{\nabla} \equiv \forall\vec{t}(\varphi^{\nabla} \Rightarrow \nabla\psi^{\nabla})$$

Now, if t^{∇} denotes the call-by-value *cps*-translation of a term t [5], we have the following property: if $\Gamma \vdash t : \varphi$ is derivable then $\Gamma^{\nabla} \vdash t^{\nabla} : \nabla\varphi^{\nabla}$ is derivable. Moreover, a direct style implementation of the *reflect*/*reify* operators using delimited control operators **shift** and **reset** can be derived from the continuation semantics:

$$[t] \equiv \mathbf{reset}\ (\mathit{unit}\ t) \qquad\qquad \mu(t) \equiv \mathbf{shift}\ (\lambda k.\mathit{bind}\ k\ t)$$

where the above instances of **shift** and **reset** are typable as follows:

$$\mathbf{reset} : \Diamond\varphi \Rightarrow \Diamond\varphi \qquad\qquad \mathbf{shift} : \forall\alpha\,((\varphi \Rightarrow \Diamond\alpha) \Rightarrow \Diamond\alpha) \Rightarrow \varphi$$

Note that $\forall\alpha(\varphi \Rightarrow \Diamond\alpha) \Rightarrow \Diamond\alpha$ is equivalent to $\Diamond\varphi$. Indeed, $f : \forall\alpha(\varphi \Rightarrow \Diamond\alpha) \Rightarrow \Diamond\alpha \vdash (f\ unit) : \Diamond\varphi$ is derivable and, conversely, we also have $x : \Diamond\varphi \vdash \lambda k.bind\ k\ x : (\varphi \Rightarrow \Diamond\alpha) \Rightarrow \Diamond\alpha$. As a consequence, the above type of **shift** is equivalent to the type of μ, which is $\Diamond\varphi \Rightarrow \varphi$.

From a provability standpoint, monadic reflection is thus completely equivalent to assuming the existence of operators **shift**/**reset** (with the above types). Let us now consider this axiom $\Diamond\varphi \Rightarrow \varphi$ for some standard monads:

- for the continuation monad $\Diamond\varphi \equiv (\varphi \Rightarrow \psi) \Rightarrow \psi$, for some formula ψ, we get $\neg\neg\varphi \Rightarrow \varphi$ which extends the logic to classical logic if we take $\psi = \bot$, but this axiom is incoherent if ψ is a theorem.
- for the state monad $\Diamond\varphi \equiv \psi \Rightarrow (\varphi \wedge \psi)$, for some formula ψ, we get $(\psi \Rightarrow (\varphi \wedge \psi)) \Rightarrow \varphi$ which is not valid in general. This axiom is derivable if ψ is a theorem, but it is incoherent if we take $\psi = \bot$;
- for the exception monad, $\Diamond\varphi \equiv \varphi \vee \psi$, for some formula ψ, we get $(\varphi \vee \psi) \Rightarrow \varphi$ which is not valid in general. This axiom is incoherent if ψ is a theorem, but it is derivable if $\neg\psi$ is derivable.

It seems that there is no obvious conclusion to draw from those remarks. For instance, it is reasonable to assume that the type of the simulated global state from the state monad is a theorem, since we usually need to initialize the state before running the computation. On the other hand, we would clearly like to allow for inhabited exception types. It is also worth noticing that the incoherent axioms mentioned above are (of course) translated into valid axioms (and a proof of $\vdash \bot$ is in fact translated only into a proof $\vdash \Diamond\bot$ which is a theorem for these cases).

One way to understand these apparent issues is to try to extend the representation theorem for **M2L** [8] (called *programming theorem* in $\mathbf{AF}_2$ [7]) to the resulting logic. For that purpose, one need to be able to derive a proof of $\vdash \sigma$ from a proof of $\vdash \Diamond\sigma$ (where σ is atomic). One can rely on Friedman's top level trick [6, 10], for the continuation monad. One can provide an initial state (if its type is a theorem) for the state monad. But, for the exception monad, in order to extract a proof of $\vdash \sigma$ from the proof of $\vdash \sigma \vee \psi$, one need to be able to prove that ψ is not a theorem (and then apply the disjunction property).

References

[1] P. N. Benton, G. M. Bierman, and V. de Paiva. Computational types from a logical perspective. *J. Funct. Program*, 8(2):177–193, 1998.

[2] H.B. Curry. The elimination theorem when modality is present. *The Journal of Symbolic Logic*, 17(4):249–265, 1952.

[3] O. Danvy and A. Filinski. A functional abstraction of typed contexts. Technical report, Copenhagen University, 1989.

[4] M. Fairtlough and M. Mendler. Propositional lax logic. *Information and Computation*, 137(1):1–33, 1997.

[5] A. Filinski. Representing monads. In *Conference Record of the Twenty-First Annual Symposium on Principles of Programming Languages*, pages 446–457, Portland, Oregon, January 1994.

[6] H. Friedman. Classically and intuitionistically provably recursive functions. *Higher Set Theory*, pages 21–27, 1978.

[7] J.-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood, 1993.

[8] D. Leivant. Contracting proofs to programs. In Odifreddi, editor, *Logic and Computer Science*, pages 279–327. Academic Press, 1990.

[9] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in computer science*, pages 14–23, Piscataway, NJ, USA, 1989. IEEE Press.

[10] C. R. Murthy. *Extracting Constructive Content from Classical proofs*. PhD thesis, Cornell University, Department of Computer Science, 1990.