No value restriction is needed for algebraic effects and handlers *

Ohad Kammar[1], Sean K. Moss[2], and Matija Pretnar[3]

[1] University of Cambridge Computer Laboratory
ohad.kammar@cl.cam.ac.uk
[2] University of Cambridge Department of Pure Mathematics and Mathematical Statistics
s.k.moss@dpmms.cam.ac.uk
[3] University of Ljubljana, Faculty of Mathematics and Physics
matija.pretnar@fmf.uni-lj.si

The proposed talk describes submitted [7] and ongoing [6] work about the interaction of computational effects and predicative polymorphism. ML-style reference cells are known to be hard to combine with polymorphism [11, 4, 14], where a naïve type system is unsound (cf. Figure 1). The working solution, the value restriction [18] and its relaxation [3], are ad-hoc and restrict the programmer unnecessarily. We reexamine this problem in the context of algebraic effects [12] which extend the monadic account of computational effects [10] (e.g., the state monad) with the syntactic operations involving them (e.g., memory look-up and update). Bauer and Pretnar [2] use effect handlers, a generalisation of exception handlers that allows to handle arbitrary user-defined algebraic effects, to structure impure functional code, in analogy with monads [17]. The smooth integration of algebraic effects with polymorphism is surprising as effect handlers can implement local-state-like programming examples by manipulating continuations (k below) [13]:

$$\begin{array}{l}\textbf{let } r = \textbf{ ref } [\,] \textbf{ in}\\ r := [()];\\ \textbf{true} :: !r\end{array}$$

Figure 1: unsound polymorphism and references [3]

$$\begin{array}{lll}(\textbf{with } H_{ST} \textbf{ handle } \mathsf{set}\ \textbf{true}; & \text{where: } H_{ST} := & \textbf{handler } \{\\ \quad \textbf{let } y = \mathsf{get}\ () \textbf{ in} & & \textbf{return } x \mapsto \textbf{fun } \_ \mapsto \textbf{return } x\\ \quad \textbf{return } y)\ \textbf{false} & & \mathsf{get}(\_; k) \mapsto \textbf{fun } s \mapsto k\ s\ s\\ \leadsto^*\ \textbf{return true} & & \mathsf{set}(s'; k) \mapsto \textbf{fun } \_ \mapsto k\ ()\ s'\}\end{array}$$

In this work, we extend Bauer and Pretnar's [2] calculus for algebraic effects and handlers with Hindley-Milner polymorphism, in a standard way, without any value restriction:

- We add local effect signatures [8] Σ as finite mappings from operations op to pairs of value types A, B, which we denote by $(\mathsf{op} : A \to B) \in \Sigma$.
- We extend types with *type variables* α.
- We introduce *schemes* $\forall\vec{\alpha}.A$, where $\vec{\alpha}$ denotes a finite set of $|\vec{\alpha}|$-many type variables ranged over by α_i.

Our main result concerns the soundness of the type system w.r.t. the reduction relation $\leadsto$:

Theorem (Safety)**.** *If* $\vdash c : A\,!\,\Sigma$ *holds, then either: (i)* $c \leadsto c'$ *for some* $\vdash c' : A\,!\,\Sigma$*; (ii)* $c = \textbf{return } v$ *for some* $\vdash v : A$*; or (iii)* $c = \mathsf{op}(v; y.\, c')$ *for some* $(\mathsf{op} : A_{\mathsf{op}} \to B_{\mathsf{op}}) \in \Sigma$*,* $\vdash v : A_{\mathsf{op}}$*, and* $y : B_{\mathsf{op}} \vdash c' : A\,!\,\Sigma$*. In particular, when* $\Sigma = \emptyset$*, evaluation will not get stuck before returning a value.*

*Supported by the European Research Council grant 'events causality and symmetry — the next-generation semantics', and the Air Force Office of Scientific Research, Air Force Materiel Command, USAF under Award No. FA9550-14-1-0096.

We use Leroy's [9] benchmarks for evaluating the interaction of effects and polymorphism. For example, if we extend the language with lists and bounded iteration, we can integrate effects in polymorphic functions, as for any Σ:

$$\begin{array}{l}\textbf{let}\ \mathrm{imp\_map} = \textbf{fun}\ f\ xs \mapsto \\ \quad \textbf{with}\ H_{ST}\ \textbf{handle}\ \left(\left(\textbf{foldl}\ \ \left(\textbf{fun}\ x \mapsto \mathsf{set}(f\ x :: \mathsf{get}\ ())\right)\ ()\ xs\right);\ \mathrm{reverse}(\mathsf{get}\ ())\right. \\ \quad [\,]\ (*\ \text{initial state}\ *)\ \textbf{in}\ \ldots \qquad (*\ \mathrm{imp\_map} : \forall\alpha\beta.(\alpha \to \beta\,!\,\Sigma) \to (\alpha\ \mathsf{list} \to \beta\ \mathsf{list}\,!\,\Sigma)\,!\,\emptyset\ *)\end{array}$$

These benchmarks also highlight the limited expressiveness of effect handlers — we do not know how to implement, using effect handlers, even Leroy's basic benchmark, in which we return a newly allocated reference cell. The advantage is that, when trying to express the problematic program in Figure 1, we cannot express the first line, and the type system forbids handling the last two lines using the same state handler, as they refer to memory cells of different types.

A deeper soundness result comes from a denotational model for algebraic effects and polymorphism [6]. We modify Seely's models of impredicative polymorphism [15] by separating the fibred category of types into a fibred embedding of a fibred category of types into a fibred category of schemes. The universal quantifier $\forall$, previously right adjoint to structural weakening, is now replaced by a *relative* right adjoint [16, 1] along the inclusion of types in schemes. Using this relativisation, we can construct a parametric version of Harper and Mitchell's [5] set-theoretic models relative to a universal set $\mathcal{U}$. To add computational effects to this model, we construct a free fibred monad T_Δ and prove the following theorem, which allows us to interpret the calculus of algebraic effects and handlers, establishing soundness via a denotational model.

Theorem. *If $\mathcal{U} \neq \emptyset$, then the canonical morphism $T_{\Delta'}\forall\Delta.\tau \to \forall\Delta.T_{\Delta'\times\Delta}\tau$ is invertible.*

References

[1] Altenkirch, T., Chapman, J., and Uustalu, T. (2014). Monads need not be endofunctors. *CoRR*, **abs/1412.7148**.

[2] Bauer, A. and Pretnar, M. (2015). Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, **84**(1), 108–123.

[3] Garrigue, J. (2004). Relaxing the value restriction. In Y. Kameyama and P. J. Stuckey, editors, *Functional and Logic Programming*, volume 2998 of *Lecture Notes in Computer Science*, pages 196–213. Springer Berlin Heidelberg.

[4] Harper, R. and Lillibridge, M. (1993). Polymorphic type assignment and CPS conversion. *Lisp and Symbolic Computation*, **6**(3-4), 361–380.

[5] Harper, R. and Mitchell, J. C. (1993). On the type structure of standard ml. *ACM Trans. Program. Lang. Syst.*, **15**(2), 211–252.

[6] Kammar, O. and Moss, S. K. (2015). A denotational semantics for Hindley-Milner polymorphism. talk given at the 4th ACM SIGPLAN Workshop on Higher-Order Programming with Effects.

[7] Kammar, O. and Pretnar, M. (2015). No value restriction is needed for algebraic effects and handlers. *Journal of Functional Programming*. Submitted.

[8] Kammar, O., Lindley, S., and Oury, N. (2013). Handlers in action. *SIGPLAN Not.*, **48**(9), 145–158.

[9] Leroy, X. (1992). *Typage polymorphe d'un langage algorithmique*. Phd thesis (in french), Université Paris 7.

[10] Moggi, E. (1991). Notions of computation and monads. *Information and Computation*, **93**(1), 55–92.

[11] Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press, Cambridge, MA, USA.

[12] Plotkin, G. D. and Power, J. (2002). Notions of computation determine monads. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer Berlin Heidelberg.

[13] Plotkin, G. D. and Pretnar, M. (2013). Handling algebraic effects. *Logical Methods in Computer Science*, **9**(4).

[14] Rémy, D. (2015). Type systems. Lecture notes, Parisian Master of Research in Computer Science.

[15] Seely, R. A. G. (1987). Categorical semantics for higher order polymorphic lambda calculus. *J. Symb. Log.*, **52**(4), 969–989.

[16] Ulmer, F. (1968). Properties of dense and relative adjoint functors. *Journal of Algebra*, **8**(1), 77 – 95.

[17] Wadler, P. (1990). Comprehending monads. In *Proceedings of the 1990 ACM conference on LISP and functional programming*, LFP '90, pages 61–78, New York, NY, USA. ACM.

[18] Wright, A. K. (1995). Simple imperative polymorphism. *LISP and Symbolic Computation*, **8**(4), 343–355.