Lecture Notes on
The Enriched Effect Calculus

15-417/817: HOT Compilation
Frank Pfenning

Lecture 22
April 17, 2025

1 Introduction

Similar to the previous lecture, we do not give a full account of the enriched effect calculus (EEC) which can be found in Egger et al. [2014]. Instead, the purpose of this lecture is to put it into the context of the course and also the preceding lecture on polarizing types and call-by-push-value (CBPV). In both cases the adjoint point of view can add some color and additional opportunities.

The polarized type system of CBPV distinguishes between values and computations, where the latter may have effects. Various restrictions guarantee that there is only one way to compose computations, using the introduction and elimination rule for $\downarrow A$. These are fundamentally the same as the **return** and **bind** constructs for strong monads in Moggi's [1989, 1991] computational λ-calculus.

We would like to implement commonly occurring effects *imperatively*. For example, state update should modify an underlying store, or printing should modify an output stream. We have already seen two examples where in-place update was enabled by linear or affine typing: an optimization of Sax in Lecture 6 and wholesale heap-free functions in LFPL [Hofmann, 2000]. Others include work by Lorenzen et al. [2023]. An underlying theme is substructural typing (linear or affine, in these examples).

One way to achieve this is to make the computational layer of call-by-push-value linear. Forgetting momentarily about the division between values and effects, this looks very much like Benton's [1994] LNL, which is in fact the origin of adjoint type systems. However, there is an issue: LNL allows arbitrary abstractions in both linear and nonlinear layers. This is fine in the setting of pure logic or type theory, but when computations have effects we then have to answer the question of the order in which such computations are executed. There is no clear answer to this question, so the approach only works with effects that are commutative such as independent flips of a coin (see also remarks by Benton and Wadler [1996, Section 8]).

2 From Call-by-Push-Value to the Enriched Effect Calculus

The approach of EEC is to carefully engineer the type system so that there can be at most one variable ranging of computations in the context. This obviates the question on the order of computations and is also familiar: already in the typing of abstract machine states in CBPV $\Gamma \mid \underline{A} \vdash K : \underline{C}$ there is exactly one computation type $\underline{A}$ in the context. Note that intuitively in a state $e \rhd K$, the computation $\cdot \vdash e : \underline{A}$ takes place before passing its terminal computation to the continuation

$\cdot \mid \underline{A} \vdash K : \underline{C}$. As already exploited by Levy [2006] we can internalize the continuation K as a computation, which then yields the following three typing judgments:

$$\begin{array}{l} \Gamma \vdash v : A \\ \Gamma \vdash e : \underline{A} \qquad (\text{written } \Gamma \mid \cdot \vdash e : \underline{A}) \\ \Gamma \mid x : \underline{A} \vdash e : \underline{C} \end{array}$$

Unlike in CBPV, the last judgment is linear in $x : \underline{A}$. On the other hand, values can be freely dropped or duplicated (as we observed also in this course), so they admit weakening and contraction. In other words, the mode v of values is structural, while the mode c of computations is linear.

At this point the language of types would be

$$\begin{array}{llcll} \text{Values} & A, B & ::= & A \times B \mid 1 \mid +\{\ell : A_\ell\}_{\ell \in L} \mid \uparrow \underline{A} & (\text{mode v with } \sigma(\mathsf{v}) = \{\mathsf{W}, \mathsf{C}\}) \\ \text{Computations} & \underline{A}, \underline{B} & ::= & A \multimap \underline{B} \mid \&\{\ell : \underline{A}_\ell\}_{\ell \in L} \mid \downarrow A & (\text{mode c with } \sigma(\mathsf{c}) = \{\,\}) \end{array}$$

This language of types does not tell the full story, because of the restriction to the above three typing judgments. Through a clever use and overloading of syntax, EEC avoids proliferation of typing rules; we focus on $\Gamma \mid \delta \vdash e : \underline{C}$ where δ could be empty $(\cdot)$ or $x : \underline{A}$.

$$\frac{\Gamma \vdash v : A}{\Gamma \mid \cdot \vdash \langle v \rangle : \downarrow A} \downarrow I \qquad \frac{\Gamma \mid \delta \vdash e : \downarrow A \quad \Gamma, x : A \mid \cdot \vdash e' : \underline{C}}{\Gamma \mid \delta \vdash \mathbf{match}\ e\ (\langle x \rangle \Rightarrow e') : \underline{C}} \downarrow E$$

Note that because any effect represented by δ must happen first, before any effect in e or e', we cannot propagate δ to the second premise and omit it from the first. What other computations could we form and adhere to the discipline? We cannot form $\underline{A} \otimes \underline{B}$ because the left rule would require two distinct computations in the premise (while $\underline{A} \,\&\, \underline{B}$ is of course okay). But we can form a skew tensor where the first component is a value and the second component is a computation, (v, e). We write this as $A \otimes \underline{B}$. This is useful to represent single-threaded mutable state of type $\underline{S}$ with the type $\underline{S} \multimap (A \otimes \underline{S})$, where A represents a regular value that might have been read from memory.

$$\frac{\Gamma \vdash v : A \quad \Gamma \mid \delta \vdash e : \underline{B}}{\Gamma \mid \delta \vdash (v, e) : A \otimes \underline{B}} \otimes I \qquad \frac{\Gamma \mid \delta \vdash e : A \otimes \underline{B} \quad \Gamma, x : A \mid y : \underline{B} \vdash e' : \underline{C}}{\Gamma \mid \delta \vdash \mathbf{match}\ e\ ((x, y) \Rightarrow e') : \underline{C}} \otimes E$$

The new skew connective $A \otimes \underline{B}$ represents a computation and is a special case of the general $(A_k \otimes B_m)_m$ for $k \geq m$. One can check the case of cut reduction as we did for $(A_k \to B_m)_m$ in the last lecture and verify that it makes sense as a (skew) connective.

The second part of the mutable state monad, expressed here as $\underline{S} \multimap (A \otimes \underline{S})$, requires a type $\underline{A} \multimap \underline{B}$. Allowing this as a computation, however, leads to similar problems with multiple effects. In particular, we cannot allow something like $\underline{A} \multimap (\underline{B} \multimap \underline{C})$ because it would overload the "stoup" in which just one computation is allowed. We also cannot lift it to a value with $\uparrow(\underline{A} \multimap \underline{B})$ because after forcing a suspension of this type, the resulting effectful expression would recreate the problem.

At this point we recall that we are really interpolating two systems: call-by-push-value (with a mode of values and a mode of computations, where only computations can have effects) and LNL (with an unrestricted mode and a linear mode, where both modes are fully populated with connectives). If we generalize away from the value/computation distinction to pure/effectful (logically: structural and linear), then we can allow function spaces in the structural layer. We

observe that with at most one computation in the context, any function of type $\underline{A} \multimap \underline{B}$ must only depend on structural variables and can therefore itself be dropped or duplicated. In other words, it is sound to apply weakening and contraction to antecedents of this type.

Generalizing to the adjoint case, we have new skew connective

$$(A_m \to B_m)_k \quad \sim \quad \uparrow^k_m (A_m \to B_m) \qquad \text{for } k \geq m$$

Again, we can check the right and left rules of the sequent calculus and establish that this is a proper (skew) connective. It is probably not a coincidence that the two skew connectives, expressed in adjoint terms as $\downarrow^k_m A_k \to B_m$ and $\uparrow^k_m (A_m \to B_m)$, represent the positive and negative translations from intuitionistic to linear logic [Girard, 1987, Section 5.1].

Getting back to the EEC rules, we obtain (keeping in mind that $\underline{A} \multimap \underline{B}$ itself is a "value" type):

$$\frac{\Gamma \mid x : \underline{A} \vdash e : \underline{B}}{\Gamma \vdash \lambda x.\, e : \underline{A} \multimap \underline{B}} \multimap I \qquad \frac{\Gamma \vdash v : \underline{A} \multimap \underline{B} \quad \Gamma \mid \delta \vdash e' : \underline{A}}{\Gamma \mid \delta \vdash v\, e' : \underline{B}} \multimap E$$

It would probably be best to abandon the value (v) and computation (e) notations, and just think of pure, structural and (potentially) effectful, linear expressions. This allows us to fill up the set of connectives to at least match Levy's complex values and more. The main backstop is to make are the restrictions to the three typing judgments laid out at the beginning of this section.

We do not show any particular set of effects that can be modeled in this calculus. Linearly-used state, linearly-used continuations, and nondeterministic choice are mentioned by Egger et al. [2014] and treated in more detail in other papers (for example, Egger et al. [2012] or Møgelberg and Staton [2011]).

3 Order, a Way Forward?

The enriched effects calculus goes through a lot of complications in order to model non-commutative effects. Specifically, it allows only a single variable ranging over computations in its context, a restriction that threads through many of the rules when compared to LNL [Benton, 1994] (and generalized in adjoint logic [Pruiksma et al., 2018]).

We extensively covered ordered types in Lectures 11, 12, 13, and 18. One could suspect that in an ordered type theory we can allow multiple variables ranging over computations in the context. Unfortunately, this is not immediate. Consider the rule of cut for ordered logic.

$$\frac{\Omega \vdash A \quad \Omega_L\, A\, \Omega_R \vdash C}{\Omega_L\, \Omega\, \Omega_R \vdash C}\ \mathsf{cut}$$

For cut elimination it is important that the formula A may occur anywhere in the context, not just at one end or another. For example, if we forced A to be at the right and (that is, $\Omega_R = (\cdot)$), then we would not be able to "push up" the cut past an application of $\twoheadrightarrow R$ in the second premise of the cut.

This behavior is inherited by the positive elimination rules. For example,

$$\frac{\Omega \vdash e : \downarrow A \quad \Omega_L\, (x : A)\, \Omega_R \vdash e' : C}{\Omega_L\, \Omega\, \Omega_R \vdash \mathbf{match}\ e\ (\langle x \rangle \Rightarrow e') : C} \downarrow E$$

However, this **match** expression is our syntax for the monadic **bind** operation that sequences (possibly effectful) computations. We see that the effect of e takes place before the effects of other

expressions that might the types in Ω_L and Ω_R. In other words, the context order does not necessarily match the effect order.

However, this is not necessarily a dead end. Reed's [2009] "Queue Logic" restricts the set of ordered connectives so that we can, in fact, restrict sequent calculus left rules to take place only at the left end of the context. This might bear some further investigation.

References

P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Selected Papers from the 8th International Workshop on Computer Science Logic (CSL'94)*, pages 121–135, Kazimierz, Poland, September 1994. Springer LNCS 933. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

P. N. Benton and Philip Wadler. Linear logic, monads, and the lambda calculus. In E. Clarke, editor, *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 420–431, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. Linear-use CPS translations in the enriched effect calculus. *Logical Methods in Computer Science*, 8(4), 2012.

Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. The enriched effect calculus. *Journal of Logic and Computation*, 24(3):615–654, 2014.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

Martin Hofmann. A type system for bounded space and functional in-place update. In G. Smolka, editor, *Proceedings of the European Symposium on Programming (ESOP 2000)*, pages 165–179, Berlin, Germany, March 2000. Springer LNCS 1782.

Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006.

Anton Lorenzen, Daan Leijen, and Wouter Swierstra. FP^2: Fully in-place functional programming. In *International Conference on Functional Programming (ICFP 2023)*, Proceedings on Programming Languages, pages 275–304. ACM, August 2023.

Rasmus Ejlers Møgelberg and Sam Staton. Linearly-used state in models of call-by-value. In *4th Conference on Algebra and Coalgebra in Computer Science (CALCO 2011)*, pages 293–313. Springer LNCS 6859, 2011.

Eugenio Moggi. Computational lambda calculus and monads. In *Proceedings of the Fourth Symposium on Logic in Computer Science*, pages 14–23, Asilomar, California, June 1989. IEEE Computer Society Press.

Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

Klaas Pruiksma, Willow Chargin, Frank Pfenning, and Jason Reed. Adjoint logic. Unpublished manuscript, April 2018. URL http://www.cs.cmu.edu/~fp/papers/adjoint18b.pdf.

Jason Reed. Queue logic: An undisplayable logic? Unpublished note, April 2009. URL https://www.cs.cmu.edu/~jcreed/papers/queuelogic.pdf.