Handler calculus

Sam Lindley

Abstract

We present handler calculus, a core calculus of effect handlers. Inspired by the Frank programming language, handler calculus does not have primitive functions, just handlers. Functions, products, sums, and inductive types, are all encodable in handler calculus. We extend handler calculus with recursive effects, which we use to encode recursive data types. We extend handler calculus with parametric operations, which we use to encode existential data types. We then briefly outline how one can encode universal data types by composing a CPS translation for parametric handler calculus into System F with Fujita's CPS translation of System F into minimal existential logic.

1 A calculus of effect handlers

Effect handlers [5] offer a modular foundation [4] for higher-order programming with effects. We introduce a core calculus of effect handlers. Our aim is to build most other features on top of effects and handlers, which we take as primitive.

Handler calculus is a fine-grain call-by-value language, which makes an explicit distinction between values and computations at the level of types and terms. The typing rules calculus are in Figure 1, and the small-step operational semantics in Figure 2. Full details are in Appendix A.

Apart from effect types (E) and handler types ($C \Rightarrow D$), the only other type we build into the calculus is the empty type (0) which we use to get off the ground. Operations $\mathsf{op}(\overline{V})$ are n-ary and handlers are first class as in Frank [1]. Unlike in Frank, handlers are unary. A handler consists of a return clause ($\{\mathbf{return}\ x \mapsto N\}$) and a collection of zero or more operation clauses. An operation clause $\{\langle \mathsf{op}(\overline{p}) \to r\rangle \mapsto N\}$ specifies how to interpret op. As well as the parameters $\overline{p}$ it binds the delimited resumption r of the computation being handled. We may omit the return clause of a handler when it is the identity $\{\mathbf{return}\ x \mapsto \mathbf{return}\ x\}$. Similarly, we may omit operation clauses that perform forwarding ($\{\langle \mathsf{op}(\overline{x}) \to r\rangle \mapsto \mathbf{let}\ x = \mathsf{op}(\overline{x})\ \mathbf{in}\ r\ x\}$). There are four reduction rules. The first is the standard let rule. The second interprets a returned value using the return clause. The third interprets an operation using the appropriate operation clause. The fourth lifts reduction through evaluation contexts.

Handler calculus handlers are *deep* [4] in that the resumption reinvokes the original handler. Handler calculus is terminating, as may be readily shown via a CPS translation to System F (Appendix D).

2 Data types

We often omit empty effects (writing A instead of $[]A$). Thus, a pure function has type $A \Rightarrow B$.

$\boxed{\Gamma \vdash V : A}$ $\boxed{\Gamma \vdash M : C}$

$$\textsc{T-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{T-Absurd}\ \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{absurd}\ V : [E]A}$$

$$\textsc{T-Return}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\ V : [E]A} \qquad \textsc{T-Let}\ \frac{\Gamma \vdash M : [E]A \quad \Gamma, x : A \vdash N : [E]B}{\Gamma \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : [E]B}$$

$$\textsc{T-Op}\ \frac{\Gamma \vdash V : A \quad \mathsf{op} : (\overline{A}) \to B \in E}{\Gamma \vdash \mathsf{op}(\overline{V}) : [E]B} \qquad \textsc{T-Handle}\ \frac{\Gamma \vdash M : C \quad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : D}$$

$$\textsc{T-Handler}\ \frac{\begin{array}{c} C = [(\mathsf{op}_i : (\overline{A_i}) \to B_i)_i]A \\ H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\langle \mathsf{op}_i(\overline{p_i}) \to r_i\rangle \mapsto N_i\}_i \\ \Gamma, x : A \vdash M : D \qquad (\Gamma, \overline{p_i} : \overline{A_i}, r_i : B_i \Rightarrow D \vdash N_i : D)_i \end{array}}{\Gamma \vdash H : C \Rightarrow D}$$

Figure 1. Handler calculus typing rules

Evaluation contexts

$$\mathcal{E} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{in}\ H$$

Pure evaluation contexts

$$\mathcal{F} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{F}\ \mathbf{in}\ N$$

Reduction rules

$$\begin{array}{rl} \mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ N \rightsquigarrow N[V/x] & \\ \mathbf{handle}\ V\ \mathbf{with}\ H \rightsquigarrow N[V/x], & \text{if } \{\mathbf{return}\ x \mapsto N\} \in H \\ \mathbf{handle}\ \mathcal{F}[\mathsf{op}(\overline{V})]\ \mathbf{with}\ H \rightsquigarrow & \\ \quad N[\overline{V}/\overline{p}, \{\mathbf{return}\ z \mapsto \mathbf{handle}\ \mathcal{F}[\mathbf{return}\ z]\ \mathbf{with}\ H\}/r], & \text{if } \{\langle \mathsf{op}(\overline{p}) \to r\rangle \mapsto N\} \in H \\ \mathcal{E}[M] \rightsquigarrow \mathcal{E}[N], & \text{if } M \rightsquigarrow N \end{array}$$

Figure 2. Handler calculus small-step operational semantics

Functions.

$$\begin{array}{l} \lambda x.M \equiv \{\mathbf{return}\ x \mapsto M\} \\ V\ W \equiv \mathbf{handle}\ (\mathbf{return}\ W)\ \mathbf{with}\ V \end{array}$$

Unit.

$$\begin{array}{l} 1 \equiv 0 \Rightarrow 0 \\ () \equiv \{\mathbf{return}\ x \mapsto x\} \end{array}$$

We write $\{C\}$ as shorthand for the type of thunks $1 \Rightarrow C$ and $\{M\}$ as shorthand for the thunk $\{\mathbf{return}\ () \mapsto M\}$.

It is often useful to encode data as a thunk with an empty return type, that is a computation that cannot return. This is not so surprising when one considers that an effectful computation of type $[E]A$ denotes a free monad $TA = \mu X.A + FX$ over a functor F determined by E [4]. Setting A to be 0 yields a type isomorphic to $\mu X.FX$, and we can instantiate F in order to obtain a variety of data types. We write $\{[E]\}$ as shorthand for $\{[E]0\}$ and $\mathsf{op} : (\overline{A})$ as shorthand for $\mathsf{op} : (\overline{A}) \to 0$. We write $\bot$ as shorthand for $\{\mathbf{return}\ x \mapsto \mathbf{absurd}\ x\}$ and $\langle \mathsf{op}(\overline{p}) \rangle$ for $\langle \mathsf{op}(\overline{p}) \to r \rangle$ when r is unused (as will typically be the case when the domain of r is 0).

Products.

$$\begin{aligned} A_1 \times A_2 &\equiv \{[\mathsf{pair} : (A_1, A_2)]\} \\ (W_1, W_2) &\equiv \{\mathsf{pair}(W_1, W_2)\} \\ \mathbf{let}\ (x_1, x_2) = V\ \mathbf{in}\ N &\equiv \mathbf{handle}\ V\ ()\ \mathbf{with}\ \bot \uplus \\ &\qquad \{\langle \mathsf{pair}\ (x_1, x_2) \rangle \mapsto N\} \end{aligned}$$

Sums.

$$\begin{aligned} A_1 + A_2 &\equiv \{[\mathsf{in}_1 : (A_1), \mathsf{in}_2 : (A_2)]\} \\ \mathbf{in}_i\ V &\equiv \{\mathsf{in}_i(V)\} \\ \mathbf{case}\ V\ \{\mathbf{in}_i\ x_i \mapsto N_i\}_i &\equiv \mathbf{handle}\ V\ ()\ \mathbf{with}\ \bot \uplus \\ &\qquad \{\langle \mathsf{in}_i\ x_i \rangle \mapsto N_i\}_i \end{aligned}$$

Inductive data types. We can take advantage of the fix point in free monads to encode inductive data types.

Natural numbers.

$$\begin{aligned} \mathsf{Nat} &\equiv \{[\mathsf{zero} : () \to 0, \mathsf{succ} : () \to 1]\} \\ \mathsf{zero} &\equiv \{\mathsf{zero}()\} \\ \mathsf{succ}\ V &\equiv \{\mathsf{succ}(); V\ ()\} \\ \mathsf{natrec}\ V_z\ V_s\ W &= \mathbf{handle}\ W\ ()\ \mathbf{with}\ \bot \uplus \\ &\qquad \{\langle \mathsf{zero}() \rangle \mapsto \mathbf{return}\ V_z\} \uplus \\ &\qquad \{\langle \mathsf{succ}() \to r \rangle \mapsto V_s\ (r\ ())\} \end{aligned}$$

Lists of natural numbers.

$$\begin{aligned} \mathsf{NatList} &\equiv \{[\mathsf{nil} : () \to 0, \mathsf{cons} : (\mathsf{Nat}) \to 1]\} \\ \mathsf{nil} &\equiv \{\mathsf{nil}()\} \\ \mathsf{cons}\ V\ W &\equiv \{\mathsf{cons}(V); W\ ()\} \\ \mathsf{listrec}\ V_n\ V_c\ W &= \mathbf{handle}\ W\ ()\ \mathbf{with}\ \bot \uplus \\ &\qquad \{\langle \mathsf{nil}() \rangle \mapsto \mathbf{return}\ V_n\} \uplus \\ &\qquad \{\langle \mathsf{cons}(n) \to r \rangle \mapsto V_c\ n\ (r\ ())\} \end{aligned}$$

3 Recursion

In practice it can be useful to have recursive effects.

$$E ::= (\mathsf{op}_i : (\overline{A_i}) \to B_i)_i \mid X \mid \mu X.E$$

We can now define a cooperative concurrency effect

$$\mathsf{Coop} = \mu X.(\mathsf{fork} : (\{[X]1\}) \to 1, \mathsf{yield} : () \to 1)$$

that supports forking new threads and yielding control. Crucially, the recursive effect allows forking and yielding to be performed *inside* a new thread.

Recursive data types.

$$\begin{aligned} \mu X.A &\equiv \{[\mu X.\mathsf{roll} : A[\{[X]\}/X]]\} \\ \mathsf{roll}\ V &\equiv \{\mathsf{roll}(V)\} \\ \mathsf{unroll}\ V &\equiv \mathbf{handle}\ V\ ()\ \mathbf{with}\ \bot \uplus \{\langle \mathsf{roll}(x) \rangle \mapsto \mathbf{return}\ x\} \end{aligned}$$

4 Quantification

Parametric operations. It is natural to extend handler calculus to support parametric operations [4]. A parametric operation can be invoked at different types, and a handler clause for a parametric operation must handle the operation uniformly according to the types it is instantiated with. A simple example is an exception effect.

$$\mathsf{Fail} = \mathsf{fail} : (X).() \to X$$

The fail operation can be invoked with X any type. More generally the effect signature of an operation may be parameterised by a collection of type variables $(\mathsf{op} : (\overline{X}).(\overline{A}) \to B)$.

Naively, one may imagine that parametric effects give rise to a direct encoding of universal data types; in fact they give rise to a direct encoding of existential data types. Invoking an operation constructs an existential package (a type/value pair) and handling it unpacks the existential package.

Existential data types.

$$\begin{aligned} \exists X.B &\equiv \{[\mathsf{pack} : (X).(B)]\} \\ (A, V) &\equiv \{\mathsf{pack}(A)(V)\} \\ \mathbf{let}\ (X, x) = V\ \mathbf{in}\ N &\equiv \mathbf{handle}\ V\ ()\ \mathbf{with}\ \bot \uplus \\ &\qquad \{\langle \mathsf{pack}(X).(x) \rangle \mapsto N\} \end{aligned}$$

Universal data types. The encoding of existential data types as universal data types in System F is well-known, arising as a generalisation of the classical de Morgan dual.

$$\exists X.A \equiv \forall Z.((\forall X.(A \to Z)) \to Z)$$

This encoding is local in that it does not change the representation of any other type constructors. It seems that no similarly local encoding of universal data types as existential data types is possible. However, Fujita's CPS translation [2] does provide a global encoding of System F in minimal existential logic. By composing Fujita's CPS translation with a CPS translation from parametric handler calculus into System F [3], we may obtain a (somewhat convoluted) global transformation that yields an encoding of polymorphic data types. It is not so clear that such an encoding is necessarily desirable (why not just add universal data types and be done with it?), but its existence may provide some amusement, and perhaps in certain impoverished settings it could be practically useful (e.g. as a compilation technique).

Effect polymorphism. Effect polymorphism is crucial for making effect handlers scale in practice. However, there are a number of distinct approaches, typically based on row polymorphism. We leave consideration of canonical extensions of handler calculus with effect polymorphism to further work.

References

[1] Lukas Convent, Sam Lindley, Conor McBride, and Craig McLaughlin. 2020. Doo bee doo bee doo. *J. Funct. Program.* 30 (2020), e9.

[2] Ken-etsu Fujita. 2010. CPS-translation as adjoint. *Theor. Comput. Sci.* 411, 2 (2010), 324–340.

[3] Daniel Hillerström, Sam Lindley, Robert Atkey, and KC Sivaramakrishnan. 2017. Continuation Passing Style for Effect Handlers. In *FSCD (LIPIcs, Vol. 84)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 18:1–18:19.

[4] Ohad Kammar, Sam Lindley, and Nicolas Oury. 2013. Handlers in action. In *ICFP*. ACM, 145–158.

[5] Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. *Logical Methods in Computer Science* 9, 4 (2013).

A Handler calculi

Figure 3 gives the full syntax, kinding rules, typing rules, and operational semantics for handler calculus. Figure 4 gives the full syntax, kinding rules, typing rules, and operational semantics for recursive handler calculus. The differences with respect to plain handler calculus are highlighted in grey. Figure 5 gives the full syntax, kinding rules, typing rules, and operational semantics for parametric handler calculus. Again, the differences with respect to plain handler calculus are highlighted in grey.

B Non-compositional minimalism

Technically, in handler calculus **return** is redundant, both in terms and in handlers. We could conflate value and computation terms and simply write V in place of **return** V.

Moreover, we could dispense with return clauses altogether, insisting that handlers be only applied to computations with return type 0, and simulating **return** V, where $V : A$ via an operation $\mathsf{ret} : (A) \to 0$. Concretely each handler application

$$\textbf{handle}\ M\ \textbf{with}\ H$$

could be rewritten as

$$\textbf{handle}\ (\textbf{let}\ x = M\ \textbf{in}\ \mathsf{ret}\ x)\ \textbf{with}\ H$$

and each handler

$$\begin{array}{l}\{\textbf{return}\ x \mapsto N\} \uplus \\ \{\langle \mathsf{op}_i(\overline{p_i}) \to r\rangle \mapsto N_i\}_i\end{array}$$

could be rewritten as:

$$\begin{array}{l}\{\langle \mathsf{ret}(x)\rangle \mapsto N\} \uplus \\ \{\langle \mathsf{op}_i(\overline{p_i}) \to r\rangle \mapsto N_i\}_i\end{array}$$

In a similar vein, let binding is also redundant. It could be simulated by:

$$\textbf{let}\ x = M\ \textbf{in}\ N \equiv \textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N\}$$

I choose to avoid the siren-call of these particular encodings, preferring to operate in a fine-grain call-by-value setting where there is a strict separation between value and computation terms, and control flow is made explicit. Nonetheless, I am happy to present examples in direct-style when understood as an abbreviation for the more explicit core calculus.

C CPS translation of parametric handler calculus into System F

We adapt Hillerström et al's CPS translation for effect handlers into System F [3] in order to account for parametric operations and first-class handlers. The translation is given in Figure 6. It makes use of standard encodings of the empty type, products, labelled sums, and existentials.

D CPS translation from System F into minimal existential logic

The types of minimal existential logic are given by the following grammar.

$$A, B \quad ::= \bot \mid \neg A \mid A \times B \mid \exists X.A \mid X$$

The negated type A is the type of a function that takes an A but does not return a value. The terms of minimal existential logic are given by the following grammar.

$$\begin{array}{rl} M, N ::= & x \\ \mid & \lambda x^A.M \mid M\ N \\ \mid & (M, N) \mid \textbf{let}\ (x, y) = M\ \textbf{in}\ N \\ \mid & (A, M) \mid \textbf{let}\ (X, y) = M\ \textbf{in}\ N \end{array}$$

The CPS translation we give here is a naive variant of Fujita's [2]. He assumes a distinguished continuation variable, whereas we always abstract over the continuation in the object language. Our version is slightly more uniform and avoids substitutions in the translation, but at the expense of introducing administrative redexes. It is perhaps most instructive to begin with the translation on judgements.

$$(\Gamma \vdash M : A)^* = \neg\Gamma^* \vdash M^* : \neg A^*$$

Both the context and the type is negated, but these are morally double-negations as the translation on types already factors in one negation (i.e., it really denotes the type of a *continuation*). The translation on types is as follows.

$$\begin{array}{c} X^* = X \\ (A \to B)^* = \neg A^* \times B^* \\ (\forall X.A)^* = \exists X.A^* \end{array}$$

We can see that the type translation factors in a negation because $(A \to B)^*$ is $\neg A^* \times B^*$ and not $\neg(A^* \times \neg B^*)$ as one might expect. Type variables in the translation denote continuations; hence X^* is X and not $\neg X$. The translation on terms is as follows.

$$\begin{array}{rl} (x : A)^* &= \lambda k^{A^*}.x\ k \\ (\lambda x.M : A)^* &= \lambda k^{A^*}.\textbf{let}\ (x, k) = k\ \textbf{in}\ M^*\ k \\ (M\ N : A)^* &= \lambda k^{A^*}.M^*\ (N^*, k) \\ (\Lambda X.M : A)^* &= \lambda k^{A^*}.\textbf{let}\ (X, k) = k\ \textbf{in}\ M^*\ k \\ (M\ B : A)^* &= \lambda k^{A^*}.M^*\ (B^*, k) \end{array}$$

Each term variable has a doubly-negated type in the translated term; hence continuation application is inverted ($x\ k$).

Kinds

$K ::= \mathsf{Val} \mid \mathsf{Comp} \mid \mathsf{Eff}$

Types

Values	$A, B ::= 0 \mid C \Rightarrow D$
Computations	$C, D ::= [E]A$
Effects	$E ::= (\mathsf{op}_i : (\overline{A_i}) \to B_i)_i$

Typing contexts

$\Gamma ::= \cdot \mid \Gamma, x : A$

Terms

Values	$V, W ::= x \mid H$
Computations	$M, N ::= \mathbf{absurd}\ V$
	$\mid\ \mathbf{return}\ V \mid \mathbf{let}\ x = M\ \mathbf{in}\ N$
	$\mid\ \mathsf{op}(\overline{V}) \mid \mathbf{handle}\ M\ \mathbf{with}\ H$
Handlers	$H ::= \{\mathbf{return}\ x \mapsto N\} \uplus$
	$\{\langle \mathsf{op}_i(\overline{p_i}) \to r \rangle \mapsto N_i\}_i$

(a) Handler calculus syntax

$\boxed{\Gamma \vdash T : K}$ $\boxed{\vdash \Gamma}$

$$\textsc{K-Zero}\ \frac{}{\Gamma \vdash 0 : \mathsf{Val}} \qquad \textsc{K-Handler}\ \frac{\Gamma \vdash C : \mathsf{Comp} \qquad \Gamma \vdash D : \mathsf{Comp}}{\Gamma \vdash C \Rightarrow D : \mathsf{Val}}$$

$$\textsc{K-Effect}\ \frac{(\Gamma \vdash A_i : \mathsf{Val})_i \qquad (\Gamma \vdash B_i : \mathsf{Val})_i}{\Gamma \vdash (\mathsf{op}_i : (\overline{A_i}) \to B_i)_i : \mathsf{Eff}} \qquad \textsc{K-Comp}\ \frac{\Gamma \vdash A : \mathsf{Val} \qquad \Gamma \vdash E : \mathsf{Eff}}{\Gamma \vdash [E]A : \mathsf{Comp}}$$

$$\textsc{K-Nowt}\ \frac{}{\vdash \cdot} \qquad \textsc{K-ExtendType}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : \mathsf{Val}}{\vdash \Gamma, x : A}$$

(b) Handler calculus kinding rules

$\boxed{\Gamma \vdash V : A}$ $\boxed{\Gamma \vdash M : C}$

$$\textsc{T-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{T-Absurd}\ \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{absurd}\ V : [E]A} \qquad \textsc{T-Return}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\ V : [E]A}$$

$$\textsc{T-Let}\ \frac{\Gamma \vdash M : [E]A \qquad \Gamma, x : A \vdash N : [E]B}{\Gamma \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : [E]B} \qquad \textsc{T-Op}\ \frac{\Gamma \vdash V : A \qquad \mathsf{op} : (\overline{A}) \to B \in E}{\Gamma \vdash \mathsf{op}(\overline{V}) : [E]B}$$

$$\textsc{T-Handle}\ \frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : D}$$

$$\textsc{T-Handler}\ \frac{\begin{array}{c} C = [(\mathsf{op}_i : (\overline{A_i}) \to B_i)_i]A \qquad H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\langle \mathsf{op}_i(\overline{p_i}) \to r_i \rangle \mapsto N_i\}_i \\ \Gamma, x : A \vdash M : D \qquad (\Gamma, \overline{p_i} : \overline{A_i}, r_i : B_i \Rightarrow D \vdash N_i : D)_i \end{array}}{\Gamma \vdash H : C \Rightarrow D}$$

(c) Handler calculus typing rules

Evaluation contexts

$\mathcal{E} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{in}\ H$

Pure evaluation contexts

$\mathcal{F} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{F}\ \mathbf{in}\ N$

Reduction rules

$$\begin{array}{rll} \mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ N & \rightsquigarrow N[V/x] & \\ \mathbf{handle}\ V\ \mathbf{with}\ H & \rightsquigarrow N[V/x], & \text{if } \{\mathbf{return}\ x \mapsto N\} \in H \\ \mathbf{handle}\ \mathcal{F}[\mathsf{op}(\overline{V})]\ \mathbf{with}\ H & \rightsquigarrow & \\ N[\overline{V}/\overline{p}, \{\mathbf{return}\ z \mapsto \mathbf{handle}\ \mathcal{F}[\mathbf{return}\ z]\ \mathbf{with}\ H\}/r], & & \text{if } \{\langle \mathsf{op}(\overline{p}) \to r \rangle \mapsto N\} \in H \\ \mathcal{E}[M] & \rightsquigarrow \mathcal{E}[N], & \text{if } M \rightsquigarrow N \end{array}$$

(d) Handler calculus small-step operational semantics

Figure 3. Handler calculus

Kinds

$K ::= \mathsf{Val} \mid \mathsf{Comp} \mid \mathsf{Eff}$

Types

Values $A, B ::= 0 \mid C \Rightarrow D$

Computations $C, D ::= [E]A$

Effects $E ::= (\mathsf{op}_i : (\overline{A_i}) \to B_i)_i \mid \mu X.E \mid X$

Typing contexts

$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, X : \mathsf{Eff}$

Terms

Values $V, W ::= x \mid H$

Computations $M, N ::= \mathbf{absurd}\ V \mid \mathbf{return}\ V \mid \mathbf{let}\ x = M\ \mathbf{in}\ N \mid \mathsf{op}(\overline{V}) \mid \mathbf{handle}\ M\ \mathbf{with}\ H$

Handlers $H ::= \{\mathbf{return}\ x \mapsto N\} \uplus \{\langle \mathsf{op}_i(\overline{p_i}) \to r \rangle \mapsto N_i\}_i$

(a) Recursive handler calculus syntax

$\boxed{\Gamma \vdash T : K}$ $\boxed{\vdash \Gamma}$

$$\textsc{RK-Zero}\ \frac{}{\Gamma \vdash 0 : \mathsf{Val}} \qquad \textsc{RK-Handler}\ \frac{\Gamma \vdash C : \mathsf{Comp} \qquad \Gamma \vdash D : \mathsf{Comp}}{\Gamma \vdash C \Rightarrow D : \mathsf{Val}} \qquad \textsc{RK-Effect}\ \frac{(\Gamma \vdash A_i : \mathsf{Val})_i \qquad (\Gamma \vdash B_i : \mathsf{Val})_i}{\Gamma \vdash (\mathsf{op}_i : (\overline{A_i}) \to B_i)_i : \mathsf{Eff}} \qquad \textsc{RK-Var}\ \frac{X : \mathsf{Eff} \in \Gamma}{\Gamma \vdash X : \mathsf{Eff}}$$

$$\textsc{RK-Rec}\ \frac{\Gamma, X : \mathsf{Eff} \vdash E : \mathsf{Eff}}{\Gamma \vdash \mu X.E : \mathsf{Eff}} \qquad \textsc{RK-Comp}\ \frac{\Gamma \vdash A : \mathsf{Val} \qquad \Gamma \vdash E : \mathsf{Eff}}{\Gamma \vdash [E]A : \mathsf{Comp}} \qquad \textsc{RK-Nowt}\ \frac{}{\vdash \cdot} \qquad \textsc{RK-ExtendType}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : \mathsf{Val}}{\vdash \Gamma, x : A} \qquad \textsc{RK-ExtendKind}\ \frac{\vdash \Gamma}{\vdash \Gamma, X : \mathsf{Eff}}$$

(b) Recursive handler calculus kinding rules

$\boxed{\Gamma \vdash V : A}$ $\boxed{\Gamma \vdash M : C}$

$$\textsc{RT-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{RT-Absurd}\ \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{absurd}\ V : [E]A} \qquad \textsc{RT-Return}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\ V : [E]A} \qquad \textsc{RT-Let}\ \frac{\Gamma \vdash M : [E]A \qquad \Gamma, x : A \vdash N : [E]B}{\Gamma \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : [E]B} \qquad \textsc{RT-Op}\ \frac{\Gamma \vdash V : A \qquad \mathsf{op} : (\overline{A}) \to B \in E}{\Gamma \vdash \mathsf{op}(\overline{V}) : [E]B}$$

$$\textsc{RT-Handle}\ \frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : D} \qquad \textsc{RT-Handler}\ \frac{C = [(\mathsf{op}_i : (\overline{A_i}) \to B_i)_i]A \qquad H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\langle \mathsf{op}_i(\overline{p_i}) \to r_i \rangle \mapsto N_i\}_i \qquad \Gamma, x : A \vdash M : D \qquad (\Gamma, \overline{p_i} : \overline{A_i}, r_i : B_i \Rightarrow D \vdash N_i : D)_i}{\Gamma \vdash H : C \Rightarrow D}$$

(c) Recursive handler calculus typing rules

Evaluation contexts

$\mathcal{E} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{in}\ H$

Pure evaluation contexts

$\mathcal{F} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{F}\ \mathbf{in}\ N$

Reduction rules

$$\begin{array}{lll} \mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ N \rightsquigarrow N[V/x], & & \\ \quad \mathbf{handle}\ V\ \mathbf{with}\ H \rightsquigarrow N[V/x], & & \text{if } \{\mathbf{return}\ x \mapsto N\} \in H \\ \mathbf{handle}\ \mathcal{F}[\mathsf{op}(\overline{V})]\ \mathbf{with}\ H \rightsquigarrow & & \\ \quad N[\overline{V}/\overline{p}, \{\mathbf{return}\ z \mapsto \mathbf{handle}\ \mathcal{F}[\mathbf{return}\ z]\ \mathbf{with}\ H\}/r], & & \text{if } \{\langle \mathsf{op}(\overline{p}) \to r \rangle \mapsto N\} \in H \\ \quad \mathcal{E}[M] \rightsquigarrow \mathcal{E}[N], & & \text{if } M \rightsquigarrow N \end{array}$$

(d) Recursive handler calculus small-step operational semantics

Figure 4. Recursive handler calculus

Kinds

$K ::= \mathsf{Val} \mid \mathsf{Comp} \mid \mathsf{Eff}$

Types

Values	$A, B ::= 0 \mid C \Rightarrow D \mid X$
Computations	$C, D ::= [E]A \mid X$
Effects	$E ::= (\mathsf{op}_i : \overline{(X_i : K_i)}.(\overline{A_i}) \to B_i)_i \mid X$
Types	$T ::= A \mid C \mid E$

Typing contexts

$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, X : K$

Terms

Values	$V, W ::= x \mid H$
Computations	$M, N ::= \mathbf{absurd}\ V$
	$\mid\ \mathbf{return}\ V \mid \mathbf{let}\ x = M\ \mathbf{in}\ N$
	$\mid\ \mathsf{op}(\overline{T})(\overline{V}) \mid \mathbf{handle}\ M\ \mathbf{with}\ H$
Handlers	$H ::= \{\mathbf{return}\ x \mapsto N\} \uplus$
	$\{\langle \mathsf{op}_i(\overline{X_i}).(\overline{p_i}) \to r\rangle \mapsto N_i\}_i$

(a) Parametric handler calculus syntax

$\boxed{\Gamma \vdash T : K}$ $\boxed{\vdash \Gamma}$

$$\text{PK-Zero}\ \frac{}{\Gamma \vdash 0 : \mathsf{Val}} \qquad \text{PK-Handler}\ \frac{\Gamma \vdash C : \mathsf{Comp} \qquad \Gamma \vdash D : \mathsf{Comp}}{\Gamma \vdash C \Rightarrow D : \mathsf{Val}} \qquad \text{PK-Var}\ \frac{X : K \in \Gamma}{\Gamma \vdash X : K}$$

$$\text{PK-Effect}\ \frac{(\Gamma, \overline{X_i : K_i} \vdash A_i : \mathsf{Val})_i \qquad (\Gamma, \overline{X_i : K_i} \vdash B_i : \mathsf{Val})_i}{\Gamma \vdash (\mathsf{op}_i : \overline{(X_i : K_i)}.(\overline{A_i}) \to B_i)_i : \mathsf{Eff}}$$

$$\text{PK-Comp}\ \frac{\Gamma \vdash A : \mathsf{Val} \qquad \Gamma \vdash E : \mathsf{Eff}}{\Gamma \vdash [E]A : \mathsf{Comp}} \qquad \text{PK-Nowt}\ \frac{}{\vdash \cdot} \qquad \text{PK-ExtendType}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : \mathsf{Val}}{\vdash \Gamma, x : A} \qquad \text{PK-ExtendKind}\ \frac{\vdash \Gamma}{\vdash \Gamma, X : K}$$

(b) Parametric handler calculus kinding rules

$\boxed{\Gamma \vdash V : A}$ $\boxed{\Gamma \vdash M : C}$

$$\text{PT-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \text{PT-Absurd}\ \frac{\Gamma \vdash V : 0}{\Gamma \vdash \mathbf{absurd}\ V : [E]A} \qquad \text{PT-Return}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \mathbf{return}\ V : [E]A} \qquad \text{PT-Let}\ \frac{\Gamma \vdash M : [E]A \qquad \Gamma, x : A \vdash N : [E]B}{\Gamma \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : [E]B}$$

$$\text{PT-Op}\ \frac{\Gamma \vdash \overline{T : K} \qquad \Gamma \vdash \overline{V} : \overline{A}[\overline{T}/\overline{X}] \qquad \mathsf{op} : \overline{(X : K)}.(\overline{A}) \to B \in E}{\Gamma \vdash \mathsf{op}(\overline{T})(\overline{V}) : [E](B[\overline{T}/\overline{X}])} \qquad \text{PT-Handle}\ \frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : D}$$

$$\text{PT-Handler}\ \frac{\begin{array}{c} C = [(\mathsf{op}_i : \overline{(X_i : K_i)}.(\overline{A_i}) \to B_i)_i]A \\ H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\langle \mathsf{op}_i(\overline{X_i : K_i}).(\overline{p_i}) \to r_i\rangle \mapsto N_i\}_i \\ \Gamma, x : A \vdash M : D \\ (\Gamma, \overline{X_i : K_i}, \overline{p_i : A_i}, r_i : B_i \Rightarrow D \vdash N_i : D)_i \end{array}}{\Gamma \vdash H : C \Rightarrow D}$$

(c) Parametric handler calculus typing rules

Evaluation contexts

$\mathcal{E} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{in}\ H$

Pure evaluation contexts

$\mathcal{F} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{F}\ \mathbf{in}\ N$

Reduction rules

$$\begin{array}{rll} \mathbf{let}\ x = \mathbf{return}\ V\ \mathbf{in}\ N & \rightsquigarrow N[V/x], & \\ \mathbf{handle}\ V\ \mathbf{with}\ H & \rightsquigarrow N[V/x], & \text{if } \{\mathbf{return}\ x \mapsto N\} \in H \\ \mathbf{handle}\ \mathcal{F}[\mathsf{op}(\overline{T})(\overline{V})]\ \mathbf{with}\ H & \rightsquigarrow & \\ N[\overline{T}/\overline{X}, \overline{V}/\overline{p}, \{\mathbf{return}\ z \mapsto \mathbf{handle}\ \mathcal{F}[\mathbf{return}\ z]\ \mathbf{with}\ H\}/r], & & \text{if } \{\langle \mathsf{op}(\overline{X}).(\overline{p}) \to r\rangle \mapsto N\} \in H \\ \mathcal{E}[M] & \rightsquigarrow \mathcal{E}[N], & \text{if } M \rightsquigarrow N \end{array}$$

(d) Parametric handler calculus small-step operational semantics

Figure 5. Parametric handler calculus

Standard System F encodings

Zero	$0 \equiv \forall Z.Z$
Products	$(A_i)_i \equiv \forall Z.((A_i \to)_i \to Z) \to Z$
Labelled sums	$[\mathsf{l}_i : A_i]_i \equiv \forall Z.((A_i \to Z) \to)_i Z$
Existentials	$\exists \overline{X}.A \equiv \forall Z.(\forall \overline{X}.A \to Z) \to Z$

Auxiliary definitions

Dispatchers	$\mathcal{D}_{E,R} = [\![E]\!]R \to R$
Continuations	$\mathcal{K}_{E,A,R} = ([\![A]\!] \times \mathcal{D}_{E,R}) \to R$
Computations	$\mathcal{T}_{E,A,R} = (\mathcal{K}_{E,A,R} \times \mathcal{D}_{E,R}) \to R$

Value types

$$[\![0]\!] = 0$$
$$[\![[E]A \Rightarrow [E']B]\!] = \forall Z.\mathcal{K}_{E,A,\mathcal{T}_{E',B,Z}} \times \mathcal{D}_{E,\mathcal{T}_{E',B,Z}}$$

Computation types

$$[\![[E]A]\!] = \forall Z.\mathcal{T}_{E,A,Z}$$

Effect types

$$[\![(\mathsf{op}_i : (\overline{X_i : K_i}).(\overline{A_i}) \to B_i)_i]\!]R = [\mathsf{op}_i : \exists \overline{X_i}.[\![\overline{A_i}]\!] \times ([\![B_i]\!] \to R)]_i$$

Value terms

$$[\![x : A]\!] = x$$
$$[\![H : E[A] \Rightarrow [E']B]\!] = \Lambda Z.([\![H : E[A] \Rightarrow [E']B]\!]^{\mathsf{ret}} Z, [\![H : E[A] \Rightarrow [E']B]\!]^{\mathsf{ops}} Z)$$

where

$$H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\langle \mathsf{op}_i(\overline{X_i}).(\overline{p_i}) \to r_i \rangle \mapsto N_i\}_i$$
$$[\![H : E[A] \Rightarrow [E']B]\!]^{\mathsf{ret}} = \Lambda Z.\lambda(x^{[\![A]\!]}, d^{[\![E]\!]\mathcal{T}_{E',B,Z} \to \mathcal{T}_{E',B,Z}}).[\![N]\!]Z$$
$$[\![H : E[A] \Rightarrow [E']B]\!]^{\mathsf{ops}} = \Lambda Z.\lambda y^{[\![E]\!]\mathcal{T}_{E',B,Z}}.\mathbf{case}\ y\ \{\mathsf{op}_i\ (\overline{X_i}, (\overline{p_i}, r_i)) \mapsto [\![N_i]\!]Z\}_i$$

Computation terms

$$[\![\mathbf{absurd}\ V : [E]B]\!] = \Lambda Z.\mathbf{absurd}\ [\![V]\!]$$
$$[\![\mathbf{return}\ V : [E]B]\!] = \Lambda Z.\lambda(k^{\mathcal{K}_{E,B,Z}}, d^{\mathcal{D}_{E,Z}}).k\ ([\![V]\!], d)$$
$$[\![\mathbf{let}\ x = (M : [E]A)\ \mathbf{in}\ N : [E]B]\!] = \Lambda Z.\lambda(k^{\mathcal{K}_{E,B,Z}}, d^{\mathcal{D}_{E,Z}}).[\![M]\!]Z\ (\lambda(x^{[\![A]\!]}, d'^{\mathcal{D}_{E,Z}}).[\![N]\!]Z\ (k, d'), d)$$
$$[\![\mathsf{op}(\overline{T})(\overline{V}) : [E]B]\!] = \Lambda Z.\lambda(k^{\mathcal{K}_{E,B,Z}}, d^{\mathcal{D}_{E,Z}}).d\ (\mathsf{op}\ (\overline{T}, ([\![\overline{V}]\!], \lambda x^{[\![B]\!]}.k\ (x, d))))),$$
$$[\![\mathbf{handle}\ M\ \mathbf{with}\ H : [E]B]\!] = \Lambda Z.[\![M]\!]\mathcal{T}_{E,B,Z}\ [\![H]\!]Z$$

Judgements

$$[\![\Gamma \vdash V : A]\!] = [\![\Gamma]\!] \vdash [\![V]\!] : [\![A]\!]$$
$$[\![\Gamma \vdash M : C]\!] = [\![\Gamma]\!] \vdash [\![M]\!] : [\![C]\!]$$

Figure 6. CPS translation of parametric handler calculus into System F