Call-by-Push-Value and the Enriched Effect Calculus*

Robert Harper

Spring 2024

1 Introduction

Consideration of effects motivated Paul Levy's formulation (Levy, 2003) of his *call-by-push-value (cbpv)* formalism, which distinguishes *values* from *computations*. According to his memorable formulation, "values *are*, computations *do*." Cbpv can be understood in terms of the *polarities* of type constructors, with the *negative* being characterized by their eliminatory forms, and the *positive* being characterized by their introductory forms. In cbpv the two forms of type are linked by operations that include values among the expressions, and that include suspended computations as values. The lax formulation of effects used in PFPL may be interpreted into the more refined cbpv formalism by regarding the lax modality as a composite of these operations. The cbpv formalism may be enriched by generalizing values to *valuables*, expressions that are tantamount to values in that their effect-free evaluation is assured. This permits adding Cartesian product and total function value types in a natural way.

A more far-reaching generalization, called the *enriched effect calculus (eec)*, introduces a restricted linear substructural context within computations. The one variable in the linear context, if present, stands for a computation, not a value, which is useful when extending the type structure of computations to permit sums and copowers of computation types. Linearity ensures that such computations are never "dropped on the floor" because to do so would be to disregard their effects. It is natural to consider that a computation with a single free computation variable corresponds to an initial segment of a control stack that is waiting for the named computation to finish before it itself can be activated.

2 Call-by-Push-Value Language

Types are classified into two categories, values and computations, defined by the following grammar:[1]

$$\begin{array}{llll} \text{Positive} & A & ::= & \top \mid A_1 \otimes A_2 \mid A_1 + A_2 \mid \mathsf{U}(X) \\ \text{Negative} & X & ::= & X_1 \times X_2 \mid A_1 \multimap X_2 \mid \mathsf{F}(A) \end{array}$$

The value type $\mathsf{U}(X)$ classifies *suspended computations*, or *suspensions*, of computation type X, and the computation type $\mathsf{F}(A)$ classifies *free computations* of value type A. The *(Cartesian) product* of computation types is again a computation type; it is distinct from the *(tensor) product* of value types, itself a value type. The *function* (or *power*) type maps arguments of a value type to results of a computation type, reflecting the "by-value" aspect of cbpv.

[1] Levy writes A for value types, and $\underline{A}$ for computation types.

What is similar to the lax framework is that *variables range over values*, rather than computations. Indeed Levy emphasizes that this requirement is the key to the aforementioned reconciliation of by-name and by-value calculi, which differ in this regard. In the presence of effects (even something as simple as an undefined expression such as division by zero) it is essential that variables range only over well-defined values, and not over ill-defined, or effectful, computations for the simple reason that *variables are given meaning by substitution*.

The syntax of values and computations is given by the following grammar:

$$\begin{array}{llll} \text{Values} & V & ::= & x \mid \star \mid V_1 \otimes V_2 \mid 1 \cdot V_1 \mid 2 \cdot V_2 \mid \mathsf{susp}(C) \\ \text{Computations} & C & ::= & \mathsf{ret}(V) \mid \mathsf{bnd}(C_1 \,;\, x.C_2) \mid \langle C_1, C_2 \rangle \mid C \cdot 1 \mid C \cdot 2 \mid \\ & & & \lambda(x.C) \mid \mathsf{ap}(C;V) \mid \mathsf{force}(V) \mid \mathsf{check}\ V\,\{\,C\,\} \mid \\ & & & \mathsf{split}\, V\,\{\,x_1, x_2.C\,\} \mid \mathsf{case}\ V\,\{\,x.C_1 \mid x.C_2\,\} \end{array}$$

In this setup values are exactly the introductory forms for value types, extended with variables that range over these. Computations are, on other hand, the introductory forms and eliminatory forms for computation types, and the eliminatory forms for value types. Levy includes a special form of binding for values in a computation, which may be defined by

$$\mathsf{letv}(V; x.C) \stackrel{\text{def}}{=} \mathsf{bnd}(\mathsf{ret}(V)\,;\, x.C).$$

Compared to the lax formulation a surprising feature of the cbpv setup is that pairs and λ's are computations, as are their projections and application (to a value). In particular, neither $\langle C_1, C_2 \rangle$ nor $\lambda(x.C)$ are values, though they may be turned into values by "thunking." Thus, active computations of these types are only ever projected or applied, obtaining further computations, but if pair or function is to be used passively as an argument or component of a value, then it must be explicitly turned into a value first. In this regard cbpv is more refined than the lax type system, all of whose types are characterized by the values that inhabit them.

The statics of cbpv is specified by the following two forms of judgment whose definitions are given in Figures 1 and 2:

- Value typing: $\Gamma \vdash V : A$
- Computation typing: $\Gamma \vdash C : X$

These two judgments are distinguished by the classifier being a value or computation type, in contrast to the lax setup in which both values and computations are classified by the same types.

These constructs may be equipped with a dynamics, on closed values and closed computations, that by-and-large mimics the dyamics given to the lax formalism in Harper (2016). Unlike the lax formalism the (closed) values are given syntactically, and transition is defined for computations in much the usual way by matching eliminatory to introductory forms. Equational laws may also be formulated that express β- and, perhaps, η laws for each of the constructs. There are, however, some additional laws that arise in this setting; a selection of the more unusual srules is given in Figure 3. Bear in mind that functions and pairs are computations that can only be applied and projected, respectively, so that it makes no difference whether the letf occurs outside or inside the abstraction and pairing.

Exercise 1. *Extend the cbpv language with nullary sum and nullary product types, that is the empty and unit types, including typing and equational laws.*

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash \star : \top} \qquad \frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash V_1 \otimes V_2 : A_1 \otimes A_2}$$

$$\frac{\Gamma \vdash V_1 : A_1}{\Gamma \vdash 1 \cdot V_1 : A_1 + A_2} \qquad \frac{\Gamma \vdash V_2 : A_2}{\Gamma \vdash 2 \cdot V_1 : A_1 + A_2} \qquad \frac{\Gamma \vdash C : X}{\Gamma \vdash \mathsf{susp}(C) : \mathsf{U}(X)}$$

Figure 1: Statics of CBPV (Values)

$$\frac{\Gamma \vdash V : \mathsf{U}(X)}{\Gamma \vdash \mathsf{force}(V) : X}$$

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{ret}(V) : \mathsf{F}(A)} \qquad \frac{\Gamma \vdash C_1 : \mathsf{F}(A_1) \qquad \Gamma, x : A_1 \vdash C_2 : X_2}{\Gamma \vdash \mathsf{bnd}(C_1 \,;\, x.C_2) : X_2}$$

$$\frac{\Gamma \vdash C_1 : X_1 \qquad \Gamma \vdash C_2 : X_2}{\Gamma \vdash \langle C_1, C_2 \rangle : X_1 \times X_2} \qquad \frac{\Gamma \vdash C : X_1 \times X_2}{\Gamma \vdash C \cdot 1 : X_1} \qquad \frac{\Gamma \vdash C : X_1 \times X_2}{\Gamma \vdash C \cdot 2 : X_2}$$

$$\frac{\Gamma, x : A_1 \vdash C_2 : X_2}{\Gamma \vdash \lambda(x.C_2) : A_1 \rightharpoonup X_2} \qquad \frac{\Gamma \vdash C_1 : A_2 \rightharpoonup X \qquad \Gamma \vdash V_2 : A_2}{\Gamma \vdash \mathsf{ap}(C_1; V_2) : X}$$

$$\frac{\Gamma \vdash V : \top \qquad \Gamma \vdash C : X}{\Gamma \vdash \mathsf{check}\ V\,\{\,C\,\} : X} \qquad \frac{\Gamma \vdash V : A_1 \otimes A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash C : X}{\Gamma \vdash \mathsf{split}\, V\,\{\,x_1, x_2.C\,\} : X}$$

$$\frac{\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x : A_1 \vdash C_1 : X \qquad \Gamma, x : A_2 \vdash C_2 : X}{\Gamma \vdash \mathsf{case}\ V\,\{\,x.C_2 \mid x.C_2\,\} : X}$$

Figure 2: Statics of CBPV (Computations)

Exercise 2. *Extend the cbpv language with a total function value type, including typing and equational laws.*

Exercise 3. *Complete the equational laws given in Figure 3 by filling in missing β- and η principles where appropriate. In particular give the appropriate equations for the function computation types and the sum value type.*

Exercise 4. *Extend the cbpv language with the value type* nat *of natural numbers, and the computation type* conat *of co-natural numbers. The numerals will be values of type* nat*, and the recursor will be of computation type. Dually, the state type of the generator will be a value type, but the generator itself will be a computation, as will the predecessor operation on it. Generalize to lists and streams, and then to inductive and coinductive types given by a suitable class of (monotone) type operators.*

Exercise 5. *Extend the cbpv language with a "print" command that, given a string, forms a command to emit that string to the "standard output". Similarly, extend it with a "read" command that, when executed, yields a string obtained from the "standard input." What equations govern these primitives?*

The cbpv formalism separates values and computations by their types, so that certain types are those of values, and certain other types are those of computations. It is possible, and sometimes useful, to break this strict association by allowing limited forms of computations as tantamount to values, called *generalized values*, or *valuable* expressions (of value type). For example, it is sensible to permit $2 + 2$ to be valuable, if not a value, but $2 \div x$ cannot be so considered, because of undefinedness. This, then, suggests formulation of *total* function types, $A_1 \to A_2$, as positive types, allowing their applications to values as valuable forms of expression. Similarly, one could permit a projective form of product of value types being again a value type, with projection as a valueable expression.

Exercise 6. *Give a precise formulation of these ideas by extending the language of values to permit valuable expressions, those that are tantamount to values using an evaluation relation, and, in conjunction with this, formulate the total function and (projective) product of value types as value types.*

3 Interpreting Lax into CBPV

The cbpv framework is more refined than the lax framework in that the latter is interpretable within the former according to the following general plan. First, as with the lax formulation, variables range only over values, but unlike the lax formulation, their types must be of a restricted class of value types, which does not include products or functions. But products and functions are permitted as arguments to other functions in the lax language. This discrepancy is reconciled using explicit "thunks" in a way that is unfamiliar in the lax setting. In particular, λ's must be suspended before they can be used as values, and similarly for tuples, including the null tuple! Second, it is necessary to account for the lax modality in the cbpv setting as a composite of the F and U modalities, the first classifying "free" computations, the second turning them into suspensions.

Figure 4 defines for each type A of the lax language two type translations into the cbpv language:

1. The *value* interpretation, written $||A||$, which is a value type;

2. The *computation* interpretation, written $|A|$, which is defined to be $\mathsf{F}(||A||)$, a computation type.

THUNK-β
$$\frac{\Gamma \vdash C : X}{\Gamma \vdash \mathsf{force}(\mathsf{susp}(C)) \equiv C : X}$$

THUNK-η
$$\frac{\Gamma \vdash V : \mathsf{U}(X)}{\Gamma \vdash \mathsf{susp}(\mathsf{force}(V)) \equiv V : \mathsf{U}(X)}$$

FREE-β
$$\frac{\Gamma \vdash V : A \qquad \Gamma, x : A \vdash C : Y}{\Gamma \vdash \mathsf{bnd}(\mathsf{ret}(V)\,;\, x.C) \equiv [V/x]C : Y}$$

FREE-η
$$\frac{\Gamma \vdash C : \mathsf{F}(A)}{\Gamma \vdash C \equiv \mathsf{bnd}(C\,;\, x.\,\mathsf{ret}(x)) : \mathsf{F}(A)}$$

CHECK-β
$$\frac{\Gamma \vdash C : X}{\Gamma \vdash \mathsf{check}\ \star\{C\} \equiv C : X}$$

CHECK-η
$$\frac{\Gamma \vdash V : \top \qquad \Gamma \vdash C : X}{\Gamma \vdash C \equiv \mathsf{check}\ V\{C\} : X}$$

SPLIT-β
$$\frac{\Gamma \vdash V_1 : A_1 \qquad \Gamma \vdash V_2 : A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash C : X}{\Gamma \vdash \mathsf{split}\, V_1 \otimes V_2 \{x_1, x_2.C\} \equiv [V_1, V_2/x_1, x_2]C : X}$$

SPLIT-η
$$\frac{\Gamma \vdash V : A_1 \otimes A_2 \qquad \Gamma, x : A_1 \otimes A_2 \vdash C : X}{\Gamma \vdash [V/x]C \equiv \mathsf{split}\, V \{x_1, x_2.[x_1 \otimes x_2/x]C\} : X}$$

LETF-LETF
$$\frac{\Gamma \vdash C_1 : \mathsf{F}(A_1) \qquad \Gamma, x : X_1 \vdash C_2 : \mathsf{F}(A_2) \qquad \Gamma, y : X_2 \vdash C_3 : X_3}{\Gamma \vdash \mathsf{bnd}(C_1\,;\, x.\,\mathsf{bnd}(C_2\,;\, y.C_3)) \equiv \mathsf{bnd}(\mathsf{bnd}(C_1\,;\, x.C_2)\,;\, y.C_3) : X_3}$$

LETF-FUN
$$\frac{\Gamma \vdash C : \mathsf{F}(A) \qquad \Gamma, x : A, y : B_1 \vdash C_2 : Y_2}{\Gamma \vdash \mathsf{bnd}(C\,;\, x.\,\lambda(y.C_2)) \equiv \lambda(y.\,\mathsf{bnd}(C\,;\, x.C_2)) : B_1 \rightharpoonup Y_2}$$

LETF-PAIR
$$\frac{\Gamma \vdash C : \mathsf{F}(A) \qquad \Gamma, x : A \vdash C_1 : Y_1 \qquad \Gamma, x : A \vdash C_2 : Y_2}{\Gamma \vdash \mathsf{bnd}(C\,;\, x.\langle C_1, C_2\rangle) \equiv \langle \mathsf{bnd}(C\,;\, x.C_1), \mathsf{bnd}(C\,;\, x.C_2)\rangle : Y_1 \times Y_2}$$

Figure 3: Equational Laws (Selected)

$$
\begin{aligned}
||\,\mathsf{unit}\,|| &\stackrel{\text{def}}{=} \top \\
||A_1 \times A_2|| &\stackrel{\text{def}}{=} ||A_1|| \otimes ||A_2|| \\
||\,\mathsf{void}\,|| &\stackrel{\text{def}}{=} 0 \\
||A_1 + A_2|| &\stackrel{\text{def}}{=} ||A_1|| + ||A_2|| \\
||A_1 \to A_2|| &\stackrel{\text{def}}{=} ||A_1|| \to ||A_2|| \\
||\,\mathsf{comp}(A)|| &\stackrel{\text{def}}{=} \mathsf{U}(|A|)
\end{aligned}
$$

$$
|A| \stackrel{\text{def}}{=} \mathsf{F}(||A||)
$$

Figure 4: Interpretation of Lax Types

The value interpretation is extended to contexts variable-by-variable, writing $||\Gamma||$ for the context in $x : A$ is translated to $x : ||A||$. This formulation expresses that variables range only over elements of value types.

These translations of types are used to define corresponding translations of terms and expressions from the lax language into the cbpv language:

1. If $\Gamma \vdash^{\text{lax}} M : A$, then $||\Gamma|| \vdash^{\text{cbpv}} ||M|| : ||A||$.
2. If $\Gamma \vdash^{\text{lax}} E \div A$, then $||\Gamma|| \vdash^{\text{cbpv}} |E| : |A|$.

Exercise 7. *Define the translation from lax logic into call-by-push-value using the type translation given in Figure 4 as a guide. Your translation should define the following judgments:*

1. $\Gamma \vdash M : A \rightsquigarrow ||M||$ *such that* $||\Gamma|| \vdash^{cbpv} ||M|| : ||A||$.
2. $\Gamma \vdash E \div A \rightsquigarrow |E|$ *such that* $||\Gamma|| \vdash^{cbpv} |E| : |A|$.

These are examples of type-directed translations *of the kind that are used in type-based compilers.*

Exercise 8. *Extend Exercise 7 to account for the unit and empty types, the types of natural and conatural numbers. More generally, consider a lax account of a class of inductive and coinductive types, and how to render it within a cbpv setting.*

Exercise 9. *Is there a more refined interpretation of lax into cbpv that avoids unnecessary coercion of computations into values? How might the translation be refined to permit this? Alternatively, what laws would be required to "optimize" the given interpretation to avoid unnecessary computations?*

The correctness of the translation suggested by Exercise 7 can be proved using *correspondences* indexed by types of the lax language relating (closed) lax expressions of the given lax type to their translations into cbpv expressions of the translated type as given in Figure 4.

1. For each type A of the lax formalism, a relation $M \approx V \in A$ between closed terms $M : A$ of the lax language and closed valuable terms $V : ||A||$ of the cbpv language.
2. For each type A of the lax formalism, a relation $E \sim C \in A$ between closed expressions $E \div A$ of the lax formalism and closed computations $C : |A|$ of the cbpv language.

$$\text{VAR}\ \frac{}{\Gamma, x : A \vdash x : A \rightsquigarrow x} \qquad \text{LAM}\ \frac{\Gamma, x : A_1 \vdash M_2 : A_2 \rightsquigarrow ||M_2||}{\Gamma \vdash \lambda(x.M_2) : A_1 \to A_2 \rightsquigarrow \lambda(x.||M_2||) : ||A_1|| \to ||A_2||}$$

$$\text{APP}\ \frac{\Gamma \vdash M : A_1 \to A_2 \rightsquigarrow ||M|| \qquad \Gamma \vdash M_1 : A_1 \rightsquigarrow ||M_1||}{\Gamma \vdash \mathsf{ap}(M;M_1) : A_2 \rightsquigarrow \mathsf{ap}(||M||;||M_1||)} \qquad \text{COMP}\ \frac{\Gamma \vdash E \div A \rightsquigarrow |E|}{\Gamma \vdash \mathsf{comp}(E) : \mathsf{comp}(A) \rightsquigarrow \mathsf{susp}(|E|)}$$

$$\text{RET}\ \frac{\Gamma \vdash M : A \rightsquigarrow ||M||}{\Gamma \vdash \mathsf{ret}(M) \div A \rightsquigarrow \mathsf{ret}(||M||)} \qquad \text{BND}\ \frac{\Gamma \vdash M : \mathsf{comp}(A) \rightsquigarrow ||M|| \qquad \Gamma, x : A \vdash E \div B \rightsquigarrow |E|}{\Gamma \vdash \mathsf{bnd}(M;x.E) \div B \rightsquigarrow \mathsf{bnd}(\mathsf{force}(||M||);x.|E|)}$$

Figure 5: Translation of Lax into CBPV (Selected Rules)

These relations are defined such that, at answer type, the correspondence between computations is exact—both yield yes or no when executed.

The closed correspondences are extended to open expressions as usual:

1. $\Gamma \gg M \approx V \in A$ iff for all $\gamma \approx \delta \in \Gamma$, $\hat{\gamma}(M) \approx \hat{\delta}(V) \in A$.
2. $\Gamma \gg E \sim C \in A$ iff for all $\gamma \approx \delta \in \Gamma$, $\hat{\gamma}(E) \sim \hat{\delta}(C) \in A$.

The correspondence between substitutions on each side, $\gamma \approx \delta \in \Gamma$, is defined variable-wise by requiring that $\gamma(x) \approx \delta(x) \in A$ for each $\Gamma \vdash x : A$.

Illustrative cases of the definition of the correspondences are as follows:

$$M \approx V \in \mathsf{comp}(A) \text{ iff } M \Downarrow \mathsf{comp}(E),\ V \Downarrow \mathsf{susp}(C), \text{ and } E \sim C \in A$$

$$E \sim C \in A \text{ iff } E \longmapsto^* \mathsf{ret}(V),\ C \longmapsto^* \mathsf{ret}(W),\ V \approx W \in A$$

Exercise 10 (Correspondence)**.** *Complete the definition of $M \approx V \in A$ by induction on the structure of A as illustrated in one case above.*

Theorem 1 (Translation Correctness)**.** *1. If $\Gamma \vdash M : A \rightsquigarrow ||M||$, then $\Gamma \gg M \approx ||M|| \in A$.*

2. If $\Gamma \vdash E \div A \rightsquigarrow |E|$, then $\Gamma \gg E \sim |E| \in A$.

Exercise 11 (Translation Correctness)**.** *Prove Theorem 1 using the relations defined in Exercise 10. Conclude that a complete computation of answer type in the lax formalism translates to a complete computation of answer type in the cbpv formalism that yields the same answer.*

4 Enriched Effect Calculus

It is possible to take these ideas a significant step further by permitting constructs with more subtle patterns of dependency using a restricted form of *linearity* in typing. The main idea is to add a linear context to typing that is either empty, or which declares the type of a single *computation variable*, u,

representing the result type of any computation to which it is bound. Intuitively, when the linear context is not empty, the declaration of the computation variable u represents a computation that will already have been completed before the computation under consideration is executed.

As an example consider the computational sum type, $X_1 + X_2$, of two computation types, X_1 and X_2, as a computation type. The introductory form is as expected, with the linear context governing the labelled computation. The elimination form acts on a computation as principle argument, as usual, thereby using up the ambient linear context, freeing it up for use within the type branches. And, indeed, the branches introduce a computation variable, u, standing for the labelled computation in each case; linearity ensures that this computation must be executed within each branch via reference to u. The computation passed to each branch is activated on the (mandatory) use of the variable in the linear context corresponding to that branch. Similarly, the computational *co-power* type, $A_1 \ltimes X_2$ of a value type and a computation type, classifies pairs of a value (or, more generally, valuable) expression of type A_1, and a pending computation of type X_2. The elimination form passes the ambient linear context to the principal argument, and uses a hybrid form of pattern matching with one value variable and one computation variable for use within a computation. Finally, the linear entailment may be internalized as a linear function type, itself a value type, between two computation types.

The other computation types of the cbpv formalism are axiomatized as in linear logic. Note, however, that the suspension type is limited to "closed" computations, those without a free computation variable, in keeping with the unrestricted nature of values. The free computation type behaves much as in cbpv, with an introductory form given in the empty linear context, and the binding form propagating it appropriately.

The typing rules for the eec computations are given in Figure 6. These rules excerpt those defining the following two judgments:

1. Valuable expressions: $\Gamma \vdash V : A$.
2. Computations: $\Gamma; \Lambda \vdash C : Y$.

In the computation judgment Λ is either empty, or declares a single computation variable, u, of some computation type X.

Exercise 12. *Formulate the statics, dynamics, and equational theory of the linear arrow value type between two computation types.*

Exercise 13. *Formulate the statics for the unit and empty computation types, unit and void, and the partial computation type, $A \rightharpoonup X$, in the eec setting.*

Exercise 14. *Define an appropriate dynamics for eec (including the extensions in Exercise 13) that respects the treatment of ordinary variables as values and linear variables as computations, and is consistent with the informal descriptions given above. State and verify (for illustrative cases) appropriate safety properties for this formulation.*

Exercise 15. *Define an equational theory for eec along the lines given for cbpv, leaving specific effects for separate consideration. The equations should be consistent with the dynamics given in Exercise 14. How might one formulate a precise statement of soundness for such equations?*

Exercise 16 (For Thought)**.** *Consider the implications of permitting a general linear context with any number of variable declarations. What constructs must restrict this context in some manner? What new constructs might be appropriate in such a setting?*

$$\textsc{comp-var}\ \frac{}{\Gamma; u : X \vdash u : X} \qquad \textsc{thunk-I}\ \frac{\Gamma; \varepsilon \vdash C : X}{\Gamma \vdash \mathsf{susp}(C) : \mathsf{U}(X)} \qquad \textsc{thunk-E}\ \frac{\Gamma \vdash V : \mathsf{U}(X)}{\Gamma; \varepsilon \vdash \mathsf{force}(V) : X}$$

$$\textsc{free-I}\ \frac{\Gamma \vdash V : A}{\Gamma; \varepsilon \vdash \mathsf{ret}(V) : \mathsf{F}(A)} \qquad \textsc{free-E}\ \frac{\Gamma; \Lambda \vdash C : \mathsf{F}(A) \qquad \Gamma, x : A; \varepsilon \vdash D : Y}{\Gamma; \Lambda \vdash \mathsf{bnd}(C\,; x.D) : Y}$$

$$\textsc{prod-I}\ \frac{\Gamma; \Lambda \vdash C_1 : X_1 \qquad \Gamma; \Lambda \vdash C_2 : X_2}{\Gamma; \Lambda \vdash \langle C_1, C_2 \rangle : X_1 \times X_2} \qquad \textsc{prod-E-}i\ \frac{\Gamma; \Lambda \vdash C : X_1 \times X_2}{\Gamma; \Lambda \vdash C \cdot i : X_i}$$

$$\textsc{copow-I}\ \frac{\Gamma \vdash V : A \qquad \Gamma; \Lambda \vdash C : X}{\Gamma; \Lambda \vdash V \ltimes C : A \ltimes X} \qquad \textsc{copow-E}\ \frac{\Gamma; \Lambda \vdash C : A \ltimes X \qquad \Gamma, x : A; u : X \vdash D : Y}{\Gamma; \Lambda \vdash \mathsf{cosplit}\, C\, \{\, x, u.D \,\} : Y}$$

$$\textsc{sum-I-}i\ \frac{\Gamma; \Lambda \vdash C : X_i}{\Gamma; \Lambda \vdash i \cdot C : X_1 + X_2} \qquad \textsc{sum-E}\ \frac{\Gamma; \Lambda \vdash C : X_1 + X_2 \qquad \Gamma; u_1 : X_1 \vdash C_1 : Y \qquad \Gamma; u_2 : X_2 \vdash C_2 : Y}{\Gamma; \Lambda \vdash \mathsf{case}\, C\, \{\, u_1.C_1 \mid u_2.C_2 \,\} : Y}$$

Figure 6: Enriched Effect Calculus: Statics

References

P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models: Extended abstract. In *Computer Science Logic*, volume 933, pages 121–135. Springer Berlin Heidelberg. ISBN 978-3-540-60017-6 978-3-540-49404-1. doi: 10.1007/BFb0022251. URL http://link.springer.com/10.1007/BFb0022251. Series Title: Lecture Notes in Computer Science.

Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. Linear-use CPS translations in the enriched effect calculus. Volume 8, Issue 4:923. ISSN 1860-5974. doi: 10.2168/LMCS-8(4:2)2012. URL https://lmcs.episciences.org/923.

Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, Cambridge, England, Second edition, 2016.

Paul Blain Levy. *Call-By-Push-Value*. Springer Netherlands, Dordrecht, 2003. ISBN 978-94-010-3752-5 978-94-007-0954-6. doi: 10.1007/978-94-007-0954-6. URL http://link.springer.com/10.1007/978-94-007-0954-6.