Varieties of Effects

Carsten Führmann

School of Computer Science, University of Birmingham
cxf@cs.bham.ac.uk

Abstract. We introduce the notion of *effectoid* as a way of axiomatising the notion of "computational effect". Guided by classical algebra, we define several effectoids equationally and explore their relationship with each other. We demonstrate their computational relevance by applying them to global exceptions, partiality, continuations, and global state.

1 Introduction

In this article, we shall introduce *effectoids* as sublanguages that stand for limitations of computational effects. The focus will be on call-by-value programming languages, and we shall use the computational lambda-calculus (λ_C-calculus) as the theoretical backbone.

Because the values of the λ_C-calculus are effect-free in any reasonable sense, one might think they should form the smallest effectoid. But this would be unsatisfactory, because values are not closed under equality. (For example, $(\lambda x.x)y$ is not a value, whereas its normal form y is.) In particular, the notion of value cannot be defined semantically. By contrast, effectoids will be closed under the equality in every denotational model, so we could treat them as sets of morphisms when required. In particular, we shall replace the notion of value by a notion of *algebraic value* which yields an effectoid.

Several sets of expressions (or morphisms) that received considerable attention in the recent literature turn out to be effectoids: Thielecke defined the sets of *central*, *copyable*, and *discardable* morphisms in models of continuations, capturing fundamental classes of program behaviour that correspond to different uses of control [12]. Selinger used these notions in his analysis of the duality between call-by-value and call-by-name in the presence of continuations [11]. Dealing with the same duality, Hasegawa and Kakutani pointed out a fundamental relationship between central expressions and rigid functionals [5], which play a key rôle in Filinski's recursion-from-iteration construction [2]. It was soon pointed out that those sets of expressions are interesting for arbitrary computational effects, not only continuations [3,4]. Recently, it was discovered that typical models of partiality (i.e. models where divergence is the only "effect") can be characterised by requiring every morphism to be central and "strongly copyable"[1], and also

[1] The notion of "strong copyability" was defined in the CTCS'99 conference version of [1], and we shall replace that terminology by "repeatable" in this article.

M. Nielsen and U. Engberg (Eds.): Fossacs 2002, LNCS 2303, pp. 144–159, 2002.

that discardable morphisms and algebraic values coincide for such models and yield the right notion of totality [1].

All sets of expressions mentioned above, except for the set of copyable expressions, will turn out to be effectoids. Also, all those sets are given as the solutions of equations. In algebraic geometric, sets given as solutions of (polynomial) equations are called *algebraic sets*. In analogy, we shall call effectoids that are given as solutions of equations *algebraic effectoids*.

The consideration of algebraic effectoids will lead us to notions that do not occur in the literature mentioned above: centralisers, stabilisers, and distributive expressions. Figure 1 presents an overview of most of the varieties that we shall discuss. (The numbers in the diagram are for later reference.)

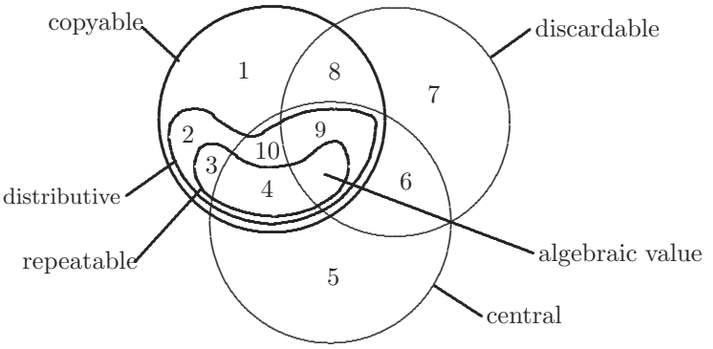

Fig. 1. Varieties introduced in this article

Remark 1. All effectoids presented in this article have interpretations in *premonoidal categories* with extra structure [10,4,3]. (In fact, several effectoids where introduced categorically.) However, the λ_C-calculus, which is the internal language of those premonoidal categories with extra structure, turned out to be the most efficient way of conveying these concepts in a computational context.

2 Preliminaries

The λ_C-calculus. The λ_C-calculus [7] has proved itself useful for reasoning about call-by-value programs. Its syntax, typing, and axioms on the well-typed terms are summarised in Figure 2, where b ranges over base types, and c^A ranges over constants of type A.

Following common practice, we write $\mathtt{let}\, x^A\, \mathtt{be}\, M\, \mathtt{in}\, N$ for $(\lambda x^A.N)M$. We shall often omit type annotations of variables when the type is evident or does not matter.

A *λ_C-theory* over given base types and constants is a set $\mathcal{T}$ of equations $\Gamma \vdash M \equiv N : A$, where $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$ are well-formed according to

$$\begin{array}{rl} \text{Types} & A, B ::= b \mid A \to B \mid A \times B \mid 1 \\ \text{Expressions} & M, N ::= x \mid c^A \mid \lambda x^A.M \mid M\,N \mid (M, N) \mid \pi_i M \mid () \\ \text{Values} & V, U ::= x \mid c^A \mid \lambda x^A.M \mid (V, U) \mid \pi_i V \mid () \end{array}$$

$$\frac{}{\Gamma \vdash x : A}\, x^A \in \Gamma \qquad \Gamma \vdash c^A : A \qquad \frac{\Gamma, x^A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \qquad \frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \pi_i M : A_i} \qquad \Gamma \vdash () : 1$$

$$\begin{array}{rll} \mathtt{let}\, x\, \mathtt{be}\, V\, \mathtt{in}\, M \equiv M[V/x] & & \\ \lambda x.Vx \equiv V & & (x \notin \mathrm{FV}(V)) \\ \pi_i(V_1, V_2) \equiv V_i & & \\ (\pi_1 V, \pi_2 V) \equiv V & & \\ V \equiv () & & \\ \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, x \equiv M & & \\ \mathtt{let}\, y\, \mathtt{be}\, (\mathtt{let}\, x\, \mathtt{be}\, L\, \mathtt{in}\, M)\, \mathtt{in}\, N \equiv \mathtt{let}\, x\, \mathtt{be}\, L\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, M\, \mathtt{in}\, N & & (x \notin \mathrm{FV}(N)) \\ MN \equiv \mathtt{let}\, f\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, N\, \mathtt{in}\, fx & & \\ (M, N) \equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x, y) & & \\ \pi_1 M \equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \pi_i x & & \end{array}$$

Fig. 2. The λ_C-calculus

the rules in Figure 2, that contains all equations presented in Figure 2 and is a congruence stable under weakening and permutation.

We write $N[M_1/x_1, \ldots, M_n/x_n]$ for the expression that results from substituting M_i for all free occurrences of x_i in N simultaneously for $i \in \{1, \ldots, n\}$ (avoiding the capture of free variables). We say that $y_1^{A_1}, \ldots, y_n^{A_n} \vdash N : B$ results from $x_1^{A_1}, \ldots, x_n^{A_n} \vdash M : B$ by *environment renaming* if $N = M[y_1/x_1, \ldots, y_n/x_n]$. We shall write $\lambda().M$ instead of $\lambda x^1.M$. If X is a set of occurrences of a variable x in M, we write $M[N/X]$ for the expression that results from replacing all those occurrences with N.

λ_C-models and their internal language. The semantics of the λ_C-calculus can be provided by *λ_C-models*. A λ_C-model is given by a strong monad T on a category C with finite products and T-exponentials—that is, exponentials of the form $(TA)^B$. (A strong monad is a monad T together with a *strength*, which is a natural transformation $t_{A,B} : A \times TB \longrightarrow T(A \times B)$ satisfying some equations due to Kock [6]. For a discussion of strong monads and a summary of those equations, see [8].)

It helps our calculations to work with the internal language of λ_C-models instead of their categorical presentation. That internal language is given by the simply-typed λ-calculus (with product types) together with a unary type constructor T, function types of the restricted form $A \to TB$ (instead of arbitrary function types), and typed-indexed families of constants $\eta_A : A \to TA$ and $*_{A,B} : TA \times (A \to TB) \to TB$ (written infix and without type annotations) satisfying the equations below (where L, M, and N range over open terms):

$$(L * M) * N \equiv L * \lambda l.((Ml) * N) \qquad \lambda x.((\eta x) * M) \equiv M \qquad M * \eta \equiv M$$

This internal language is essentially the "metalanguage" presented in [8].

Semantics of the λ_C-calculus. The categorical semantics of the λ_C-calculus can be presented by a transform $\overline{(-)}$ into the metalanguage. For every base type b, an *interpretation* must provide a type $\overline{b}$, and obey the laws

$$\overline{A \to B} = \overline{A} \to T\overline{B} \qquad \overline{A \times B} = \overline{A} \times \overline{B} \qquad \overline{1} = 1$$

For each constant c^A, an interpretation must provide a closed expression M_c of type $\overline{A}$, and transform every expression $x_1^{A_1}, \ldots, x_n^{A_n} \vdash M : B$ into an expression $x_1^{\overline{A_1}}, \ldots, x_n^{\overline{A_n}} \vdash \overline{M} : T\overline{B}$ as follows:

$$\begin{array}{lll} \overline{x} = \eta x & \overline{\lambda x.M} = \eta(\lambda x.\overline{M}) & \overline{MN} = \overline{M} * \lambda m.(\overline{N} * \lambda n.mn) \\ \overline{c^A} = \eta(M_c) & \overline{\pi_i M} = \overline{M} * \lambda m.\eta(\pi_i m) & \overline{(M,N)} = \overline{M} * \lambda m.(\overline{N} * \lambda n.\eta(m,n)) \\ \overline{()} = \eta() & & \end{array}$$

We call such a transform $\overline{(-)}$ a *monadic-style transform*. An interpretation of a λ_C-theory is called a *model* if it validates the theory's equations. This semantics is known to be sound and complete [7]. That is, the equations induced by an interpretation form a λ_C-theory (soundness), and when an equation holds in all models of a λ_C-theory, then it is a theorem (completeness).

3 Algebraic Effectoids

In this section, we shall define the notion of effectoid, introduce the varieties presented in Figure 1, show how they are related with each other, and apply them to some models (accumulation, powerset, global exceptions, partiality).

3.1 Algebraic Values

An *algebraic value* M is an expression that can be substituted for the occurrences of a formal parameter x in any procedure body N whenever M is passed as the actual parameter. Formally, an expression $\Gamma \vdash M : A$ is defined to be an algebraic value of a λ_C-theory $\mathcal{T}$ if every well-formed equation

$$\Gamma' \vdash \texttt{let}\, x\, \texttt{be}\, M\, \texttt{in}\, N \equiv N[M/x] : B \tag{1}$$

is a theorem of $\mathcal{T}$ whenever Γ' contains Γ.

Every algebraic value M of function type is equivalent to a value, because $M \equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \lambda y.xy \equiv \lambda y.My$. At other types this can be false. For example, in λ_C-theories induced by computationally realistic interpretations, the algebraic value $x^{\mathtt{int}} \vdash -x : \mathtt{int}$ is not equivalent to a value. (This follows from a simple inductive argument, using only that the expression is not equivalent to a constant or a variable, and that its type is a base type different from 1.)

Importantly, if $\Gamma \vdash M : A$ and $\Gamma, x^A \vdash L : B$ are algebraic values, then so is $\Gamma \vdash \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, L : B$, because

$$\begin{aligned}&\mathtt{let}\, x\, \mathtt{be}\, (\mathtt{let}\, y\, \mathtt{be}\, M\, \mathtt{in}\, L)\, \mathtt{in}\, N \equiv \mathtt{let}\, y\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, L\, \mathtt{in}\, N\\ &\equiv \mathtt{let}\, y\, \mathtt{be}\, M\, \mathtt{in}\, N[L/x] \equiv N[L[M/y]/x] \equiv N[\mathtt{let}\, y\, \mathtt{be}\, M\, \mathtt{in}\, L/x]\end{aligned}$$

Definition 1. *An* effectoid *in a λ_C-theory is a set of expressions $\Gamma \vdash M : A$ which contains all algebraic values, and is closed under weakening, permutation, equality, and the let-construct.*

Evidently, effectoids are closed under arbitrary intersection, so they form a complete lattice, with the algebraic values as the smallest element.

Lemma 1. *An expression $\Gamma \vdash M : A$ is an algebraic value if and only if*

$$\Gamma \vdash \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \lambda().x \equiv \lambda().M : 1 \to A \tag{2}$$

To see the right-to-left implication, assume Equation 2, and consider

$$\begin{aligned}\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N &\equiv \mathtt{let}\, y\, \mathtt{be}\, (\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \lambda().x)\, \mathtt{in}\, N[y()/x]\\ &\equiv \mathtt{let}\, y\, \mathtt{be}\, \lambda().M\, \mathtt{in}\, N[y()/x] \equiv N[M/x]\end{aligned}$$

Remark 2. Being an algebraic value is not well defined for expressions without environment. For example, for every $\Gamma \vdash M : A$, the weakened version $\Gamma, x^0 \vdash M : A$ is an algebraic value if $\overline{0}$ denotes the initial object. The same applies to all algebraic effectoids in this article.

3.2 Centralisers and Central Expressions

Just as we can state what it means for two group elements to commute, we can define a notion of commuting expressions for any λ_C-theory: expressions $\Gamma \vdash M : A$ and $\Delta \vdash N : B$ where Γ and Δ are disjoint are said to commute if the equation below is a theorem

$$\begin{aligned}\Gamma, \Delta \vdash &\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x, y)\\ \equiv &\mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, y) : A \times B\end{aligned} \tag{3}$$

If Γ and Δ have variables in common, then $\Gamma \vdash M : A$ and $\Delta \vdash N : B$ are said to commute if $\Gamma' \vdash M' : A$ and $\Delta' \vdash N' : B$ commute, where the latter

two expressions result from environment-renaming the former such that Γ' and Δ' are disjoint. (Obviously, this definition does not depend on the choice of the renaming.)

Equation 3 holds if and only if for all expressions environments Γ' containing Γ and Δ, every well-formed equation of the form

$$\Gamma' \vdash \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, L \equiv \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, L : C \quad (4)$$

is a theorem. The right-to-left implication is trivial, and the left-to-right implication follows immediately from applying $\lambda(x, y).L$ to both sides of Equation 3.

In group theory, the *centraliser* $\mathrm{Cen}(S)$ of a set S of group elements is defined to be the set of group elements that commute with every element of S. We define the centraliser of a set S of λ_C-expressions in exactly the same way. Just as group-theoretic centralisers form a subgroup, centralisers in λ_C-theories form an effectoid. (The closure under `let` follows from the associativity of `let` and Equation 4.)

In analogy to group theory, we define the *centre* of a λ_C-theory to be the centraliser of the set of all expressions. (This notion of centre for computational models was first introduced, categorically and without defining centralisers, in [10], and used in [12,3,11,4,5].)

Example 1. The following model links centralisers in λ_C-theories with centralisers from classical algebra: for a monoid $(\mathcal{M}, \cdot, 1)$, consider the *accumulation monad* on **Set**:

$$TA = A \times \mathcal{M} \qquad \eta = \lambda x.(x, 1) \qquad (x, m) * f^{A \to B \times M} = (f_1 x, m \cdot f_2 x)$$

where $f_1 : A \to B$ and $f_2 : A \to \mathcal{M}$ are the evident components of f. Let $\mathcal{T}_\mathcal{M}$ be the λ_C-theory whose base types are sets, whose constants are of the form $f : A \to B$ where f is a function from A to TB, and whose equations are induced by the interpretation given by $\overline{A} = A$ and $\overline{f} = f$. Then two expressions $x^{A'} \vdash M : A$ and $y^{B'} \vdash N : B$ of $\mathcal{T}_\mathcal{M}$ commute if and only if $(\overline{M})_2 x$ and $(\overline{N})_2 y$ commute in $\mathcal{M}$ for all $x \in \overline{A'}$ and $y \in B'$.

3.3 Stabilisers and Discardable Expressions

For a group G acting on a set X, the stabiliser (a.k.a. isotropic group) $\mathrm{Stab}(x)$ of an element $x \in X$ is the set of group elements g that stabilise x—that is, for which $g(x) = x$. In particular, for the operation $g(h) = g \circ h$ of G on itself, the stabiliser of g is the set of all h such that $g \circ h = h$. In a λ_C-theory, an expression $\Gamma \vdash M : A$ is defined to stabilise an expression $\Delta \vdash N : B$ (with Δ disjoint from Γ) if the equation

$$\Gamma, \Delta \vdash M; N \equiv N : B \quad (5)$$

is a theorem where $M; N$ stands for $\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N$ where x is fresh. As in the previous section, we use environment-renaming to extend the definition to the

case where Γ and Δ overlap. Just as group-theoretic stabilisers are subgroups, stabilisers in λ_C-theories are effectoids.

Of particular interest are the expressions that stabilise the empty tuple $\vdash () : 1$. Such expressions are called *discardable*. (The notion of discardability was introduced categorically in [12] and consequently used in [3,11,4,1]. The notion of stabiliser is a contribution of this article.)

Lemma 2. *If an expression stabilises a value V, then it stabilises every expression N. (In particular, the discardable expressions are the smallest stabiliser.)*

This follows immediately from applying $\lambda x.N$ to both sides of the equation $M; V \equiv V$.

In our accumulation-monad example $\mathcal{T}_{\mathcal{M}}$, $x^{A'} \vdash M : A$ stabilises $y^{B'} \vdash N : B$ if and only if $(\overline{M})_2 x$ stabilises $(\overline{N})_2 y$ for all $x \in \overline{A'}$ and $y \in \overline{B'}$. Therefore, $x^{A'} \vdash M : A$ is discardable in $\mathcal{T}_{\mathcal{M}}$ if and only if $(\overline{M})_2 x \equiv 1$ for every $x \in \overline{A'}$. This already implies that $x^{A'} \vdash M : A$ is an algebraic value.

Example 2. Writing PA for the powerset of A, the *powerset monad* is given by

$$TA = PA \qquad \eta = \lambda x.\{x\} \qquad X * f^{A \to PB} = \bigcup_{x \in X} fx$$

Define $\mathcal{T}_{Pow}$ to be the evident λ_C-theory induced by the powerset monad. Essentially, expressions of $\mathcal{T}_{Pow}$ denote relations. While the algebraic values of $\mathcal{T}_{Pow}$ are the expressions that denote total functions, the discardable expressions are those that denote total relations—that is, relations R such that $\forall x \exists y : xRy$. All expressions are central.

3.4 Copyable Expressions

One notion that has found some attention in the literature is that of *copyable* expressions [12,11,3,4]. An expression $\Gamma \vdash M : A$ of a λ_C-theory $\mathcal{T}$ is called copyable if

$$\Gamma \vdash \texttt{let}\, x\, \texttt{be}\, M\, \texttt{in}\, (x, x) \equiv (M, M) : A \times A \tag{6}$$

In our accumulation-monad example, an expression $x^{A'} \vdash M : A$ is copyable if and only if $(\overline{M})_2 x$ is an idempotent in $\mathcal{M}$ for every $x \in \overline{A'}$. In our powerset example, an expression is copyable if and only if the denoted relation is a partial function.

Among all varieties in this article, the copyable expressions form the only one which is not an effectoid. For the copyable expressions of $\mathcal{T}_{\mathcal{M}}$ where closed under `let`, then the idempotents of $\mathcal{M}$ would be closed under composition. But there are obviously counterexamples—for example, let $\mathcal{M}$ be the monoid of endofunctions on a set with at least three elements.

3.5 Distributive Expressions

Next we introduce a notion which has not been studied in the literature so far. An expression $\Gamma \vdash M : A$ is said to *distribute* over an expression $\Delta, x^A \vdash N : B$ with Δ disjoint from Γ if

$$\begin{aligned}\Gamma, \Delta \vdash & \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x, y) \\ \equiv & \mathtt{let}\, y\, \mathtt{be}\, (\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N)\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, y) : A \times B \end{aligned} \tag{7}$$

Using environment-renaming, we extend this definition to the case where Γ and Δ overlap. An expression $\Gamma \vdash M : A$ is called *distributive* if it distributes over all $\Delta, x^A \vdash N : B$.

In our accumulation-monad example, an expression $x^{A'} \vdash M : A$ distributes over $y^{B'}, x^{A'} \vdash N : B$ if and only if for all $x \in \overline{A'}$ and $y \in \overline{B'}$

$$(\overline{M})_2 x \cdot (\overline{N})_2 (y, x) \equiv (\overline{M})_2 x \cdot (\overline{N})_2 (y, x) \cdot (\overline{M})_2 x$$

Lemma 3. *An expression $\Gamma \vdash M : A$ is distributive if and only if every well-formed equation*

$$\Gamma' \vdash \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N \equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N[M/X] : B$$

holds whenever Γ' contains Γ and X is any set of free occurrences of x in N which are outside the scope of any λ-binder.

The distributive expressions form an effectoid in every λ_C-theory. To see the closure under the let-construct, let $\Gamma \vdash M_1 : A$ and $\Gamma, y^A \vdash M_2 : B$ be distributive, let X be a set of free occurrences of x in N, and consider

$$\begin{aligned} & \mathtt{let}\, x\, \mathtt{be}\, (\mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, M_2)\, \mathtt{in}\, N \\ & \equiv \mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M_2\, \mathtt{in}\, N \\ & \equiv \mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M_2\, \mathtt{in}\, N[M_2/X] && \text{(Lemma 3)} \\ & \equiv \mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M_2\, \mathtt{in}\, N[\mathtt{let}\, y'\, \mathtt{be}\, y\, \mathtt{in}\, M_2[y'/y]/X] \\ & \equiv \mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M_2\, \mathtt{in}\, N[\mathtt{let}\, y'\, \mathtt{be}\, M_1\, \mathtt{in}\, M_2[y'/y]/X] && \text{(Lemma 3)} \\ & \equiv \mathtt{let}\, x\, \mathtt{be}\, (\mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, M_2)\, \mathtt{in}\, N[\mathtt{let}\, y\, \mathtt{be}\, M_1\, \mathtt{in}\, M_2/X] \end{aligned}$$

Proposition 1. *An expression $\Gamma \vdash M : A$ is copyable if and only if it distributes over $x^A \vdash x : A$.*

Proposition 2. *Expressions that are central and copyable are distributive.*

To see this, suppose that M is central and copyable, and consider

$$\begin{aligned}
&\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x, y)\\
&\equiv \mathtt{let}\, (x, x')\, \mathtt{be}\, (\mathtt{let}\, z\, \mathtt{be}\, M\, \mathtt{in}\, (z, z))\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x', y)\\
&\equiv \mathtt{let}\, (x, x')\, \mathtt{be}\, (M, M)\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x', y) && (M \text{ copyable})\\
&\equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, x'\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x', y)\\
&\equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, \mathtt{let}\, x'\, \mathtt{be}\, M\, \mathtt{in}\, (x', y) && (M \text{ central})\\
&\equiv \mathtt{let}\, y\, \mathtt{be}\, (\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N)\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, y)
\end{aligned}$$

Proposition 3. *Expressions that are distributive and discardable are central.*

To see this, suppose that M is distributive and discardable and $x \notin \mathrm{FV}(N)$, and consider

$$\begin{aligned}
&\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, (x, y)\\
&\equiv \mathtt{let}\, y\, \mathtt{be}\, (M; N)\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, y) && (M \text{ distributive})\\
&\equiv \mathtt{let}\, y\, \mathtt{be}\, N\, \mathtt{in}\, \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, y) && (\text{Lemma 2})
\end{aligned}$$

3.6 Repeatable Expressions

In this section, we introduce a very useful effectoid which has no evident counterparts in classical algebra. An expression $\Gamma \vdash M : A$ of a λ_C-theory is called *repeatable* if every well-formed equation

$$\Gamma' \vdash \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N \equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N[M/X] : B \tag{8}$$

is a theorem whenever Γ' contains Γ and X is any set of free occurrences of x in N. Lemma 3 immediately implies the following result:

Proposition 4. *Every repeatable expression is distributive.*

However, the converse is far from true: In our accumulation-monad example $\mathcal{T}_{\mathcal{M}}$, the repeatable expressions turn out to coincide with the algebraic values—that is, expressions that produce the unit of the monoid $\mathcal{M}$.

The repeatable expressions form an effectoid in every λ_C-theory. (Proving the closure under the let-construct works like for distributivity.)

Proposition 5. *An expression $\Gamma \vdash M : A$ is repeatable if and only if*

$$\Gamma \vdash \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, \lambda().x) \equiv (M, \lambda().M) : A \times (1 \to A) \tag{9}$$

To see the right-to-left implication, let S be the set of free occurrences of x in N, let $X \subseteq S$, and consider

$$\begin{aligned}
&\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N\\
&\equiv \mathtt{let}\, (y, f)\, \mathtt{be}\, (\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, \lambda().x))\, \mathtt{in}\, N[f()/X, y/S - X]\\
&\equiv \mathtt{let}\, (y, f)\, \mathtt{be}\, (M, \lambda().M)\, \mathtt{in}\, N[f()/X, y/S - X] && (\text{Equation 9})\\
&\equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N[M/X]
\end{aligned}$$

Proposition 6. *An expression is an algebraic value if and only if it is discardable and repeatable.*

The left-to-right implication is trivial. For the other implication, suppose that M is discardable and repeatable, and consider

$$\begin{aligned} \texttt{let}\, x\, \texttt{be}\, M\, \texttt{in}\, N &\equiv M; (N[M/x]) && (M \text{ repeatable}) \\ &\equiv N[M/x] && (\text{Lemma 2}) \end{aligned}$$

Propositions 4 and 6 finish our validation of Figure 1. In our relations example, the repeatable expressions turn out to be the (expressions denoting) partial functions—that is, they coincide with the copyable expressions. The only non-empty areas in Figure 1 are area 4 (partial functions that are not total), area 6 (total relations that are not functions), algebraic values (total functions), and area 5 (relations that fit into none of the other categories). By contrast, in our accumulation-monad example $\mathcal{T}_{\mathcal{M}}$, we only have areas 1, 2, 5, 10, and the algebraic values. In fact, for each of those areas, there is a monoid $\mathcal{M}$ such for which $\mathcal{T}_{\mathcal{M}}$ has an inhabitant of the area. (The proof is left as a challenge to the reader.)

Repeatability and the lifting equation. In [1], *equational lifting monads* are defined as commutative monads that satisfy the *lifting equation*

$$T\langle \eta_A,\, id_A\rangle = t_{TA,A} \circ \langle id_{TA},\, id_{TA}\rangle \tag{10}$$

It is proved that a large class of models of partial computation, *dominical lifting monads*, are equational lifting monads[2]. Also, it is easy to check that *exceptions monads* (i.e. the well-known monads of the form $TA = A + E$) satisfy the lifting equation. (But in contrast to equational lifting monads, they are not generally commutative.) Due to this scope of the lifting equation, the following proposition is most useful:

Proposition 7. *Let $\mathcal{T}$ be a λ_C-theory induced by an interpretation $\overline{(-)}$ in a λ_C-model with monad T. If the λ_C-model satisfies the lifting equation, then every expression of $\mathcal{T}$ is repeatable. If $\overline{(-)}$ is full on types, the converse holds.*

Proof. In the metalanguage, the lifting equation is represented by

$$y^{TA} \vdash y * \lambda x.\eta(\eta x, x) \equiv y * \lambda x.\eta(y, x) : T(TA \times A) \tag{11}$$

This implies, for all M,

$$\overline{M} * \lambda x.\eta(x, \eta x) \equiv \overline{M} * \lambda x.\eta(x, \overline{M}) \tag{12}$$

which is Equation 9 sent through $\overline{(-)}$. So the lifting equation implies that every expression of $\mathcal{T}$ is repeatable. For the converse, suppose that $\overline{(-)}$ is full on objects, and that every expression of $\mathcal{T}$ is repeatable. Letting $M = f()$ for some variable $f : 1 \to A'$ with $\overline{A'} = A$, we have $\overline{M} \equiv f()$. Using this $\overline{M}$ in Equation 12 and substituting $\lambda().x$ for f implies Equation 11. □

[2] In fact, the point of [1] is to prove a more interesting statement in the opposite direction, which states that every equational lifting monad can in a certain sense be fully embedded into a dominical one.

4 Effectoids for Continuations

An analysis of algebraic values[3], centrality, copyability, and discardability for continuations was undertaken by Hayo Thielecke [12]. The novelty in this section is the analysis of distributivity and repeatability, as well as the consideration of global state.

4.1 Continuations per Se

A *continuations* monad for response type R (for which exponentials of the form $A \to R$ must exist) is given by the data below:

$$TA = (A \to R) \to R \qquad \eta = \lambda x.\lambda k.kx \qquad M * N = \lambda k.M(\lambda m.Nmk)$$

Instantiating the monadic-style transform with these data yields the "Plotkin CPS-transform". This situation allows the definition of a unary type constructor and operators for control flow manipulation:

$$\begin{array}{lll} A\,\mathtt{cont} & \mathtt{callcc} : (A\,\mathtt{cont} \to A) \to A & \mathtt{throw} : A\,\mathtt{cont} \to A \to B \\ \overline{A\,\mathtt{cont}} = \overline{A} \to R & \overline{\mathtt{callcc}} = \eta(\lambda f.\lambda k.f\,k\,k) & \overline{\mathtt{throw}} = \eta(\lambda l.\eta(\lambda x.\lambda k.l\,x)) \end{array}$$

(These operators are available in SML of New Jersey via the "SMLofNj.Cont" module[4]). Before we prove the main results of this section (Propositions 8 and 9), we gather some facts. We shall use the equation

$$\mathtt{callcc}(\lambda k.\mathtt{throw}\,k\,M) \equiv M : A \tag{13}$$

which can easily be checked with the CPS transform. Also, we define

$$\begin{aligned} \mathtt{force} &= \lambda x.\mathtt{callcc}(\mathtt{throw}\,x) : A\,\mathtt{cont}\,\mathtt{cont} \to A \\ [M] &= \mathtt{callcc}(\lambda k.\mathtt{throw}(\mathtt{force}\,k)M) : A\,\mathtt{cont}\,\mathtt{cont} \end{aligned}$$

(where M is any expression of type A). We have $\overline{\mathtt{force}\,h} \equiv \lambda k.hk$ in the case where h is a variable, and $\overline{[M]} \equiv \lambda k.k\,\overline{M}$. With these CPS-transforms we can easily check that $\mathtt{force}[M] \equiv M$ and $[\mathtt{force}\,h] \equiv h$. These two equations together and the fact that $[M]$ is an algebraic value imply immediately that the following two maps are mutually inverse:

$$\begin{aligned} \varphi &= \lambda h.\lambda().\mathtt{force}\,h : A\,\mathtt{cont}\,\mathtt{cont} \to (1 \to A) \\ \psi &= \lambda f.[f()] : (1 \to A) \to A\,\mathtt{cont}\,\mathtt{cont} \end{aligned}$$

The following result is due to Hayo Thielecke (Remark 4.4.2 in [12]) and has recently been used and restated several times (e.g. in [11] and [5]).

[3] under the name of "thunkable morphisms"

[4] Of course, the real implementation is not a simple CPS transform.

Proposition 8. *In the λ_C-theory of a continuations monad, central expressions are algebraic values.*

Proof. This is true because an expression that commutes with jumps must be effect free. Formally, if M is a central expression of type A, then it commutes with expressions of the form $\zeta\, h$, where h is a variable, and ζ is defined as $\lambda h.\mathtt{throw}(\mathtt{force}\, h)$. We have

$$\zeta\, hN \equiv \mathtt{throw}\, h[N] \tag{14}$$

because the CPS transforms of both sides are equal to $\lambda k.h\overline{N}$. Now consider

$$\begin{aligned}
&\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, [x] \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{throw}\, h(\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, [x])) && \text{(Equation 13)} \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{throw}\, h[x]) && (\mathtt{throw}\, h \text{ is an alg. value}) \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \zeta\, h\, x) && \text{(Equation 14)} \\
&\equiv \mathtt{callcc}(\lambda h.\zeta\, hM) && (M \text{ is central}) \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{throw}\, h[M]) && \text{(Equation 14)} \\
&\equiv [M] && \text{(Equation 13)}
\end{aligned}$$

Applying φ to both sides yields Equation 2. □

Proposition 9. *In the λ_C-theory of a continuations monad, distributive expressions are repeatable.*

Proof. Let $\zeta' = \lambda x.\lambda h.\mathtt{throw}(\mathtt{callcc}(\lambda l.\mathtt{throw}\, h(x, l)))$. For every distributive expression M, we have

$$\zeta'\, h(x, N) \equiv \mathtt{throw}\, h(x, [N]) \tag{15}$$

because the CPS transforms of both sides are $\lambda k.h(x, \overline{N})$. Now consider

$$\begin{aligned}
&(M, [M]) \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{throw}\, h(\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, [M]))) && \text{(Equation 13)} \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{throw}\, h(x, [M])) && (\mathtt{throw}\, h \text{ is an alg. value}) \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \zeta'\, x\, h(x, M)) && \text{(Equation 15)} \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \zeta'\, x\, h(x, x)) && \text{(Lemma 3)} \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{throw}\, h(x, [x])) && \text{(Equation 15)} \\
&\equiv \mathtt{callcc}(\lambda h.\mathtt{throw}\, h(\mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, [x]))) && (\mathtt{throw}\, h \text{ is an alg. value}) \\
&\equiv \mathtt{let}\, x\, \mathtt{be}\, M\, \mathtt{in}\, (x, [x]) && \text{(Equation 13)}
\end{aligned}$$

Applying $\lambda(x, k).(x, \varphi k)$ to both sides yields Equation 9. □

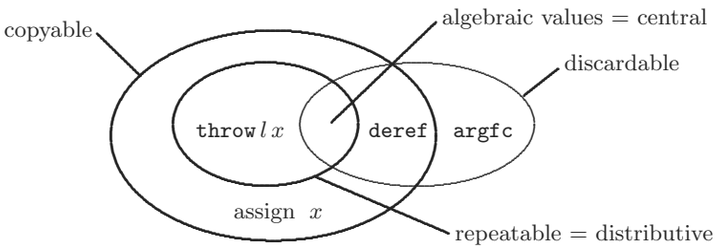

Fig. 3. Effectoids for continuations

Propositions 8 and 9 reduce Figure 1 to Figure 3. (The four expressions inhabiting the areas of Figure 3 will be discussed soon.)

To prove that an expression is inside an algebraic effectoid, we plug it into the effectoid's equations and check them with the CPS-transform. To prove that an expression is outside an algebraic effectoid, we plug it into one of the effectoid's equations and refute the equation using a *separator*. That is, letting M and N be the two sides of the equation, we find a context $C[-]$ such that $C[M]$ and $C[N]$ are programs of ground type that are reduced to different values[5]. While separators are computationally most convincing, they require a ground type with at least two elements (which we shall call `int`). So Figure 3 is to be interpreted as follows: whenever an expression is inside an area, it is in the corresponding effectoid for every continuations monad (or in the case of `assign` and `deref`, for every continuations monad with the required extra structure, which is introduced in the next section). Whenever an expression is outside an area, then it is outside the corresponding effectoid, provided that the model allows a realistic range of observations. (This includes all models for which there are at least two different expressions of type $\overline{\texttt{int}}$ and there exists a mono from $\overline{\texttt{int}}$ into R.)

That $M = \texttt{throw}\, l\, x$ (where l and x are variables) is repeatable follows immediately from plugging $\overline{M} = \lambda k.l\, x$ into Equation 12. To see that $\texttt{throw}\, l\, x$ is not discardable, it suffices to prove that $\texttt{throw}\, l\, 42$ is not discardable. This can be checked by using the separator $C[-] = \texttt{callcc}(\lambda l.\texttt{let}\, x\, \texttt{be}\, -\, \texttt{in}\, 0)$ and observing that $C[\texttt{let}\, y\, \texttt{be}\, \texttt{throw}\, l\, 42\, \texttt{in}\, ()]$ evaluates to 42 whereas $C[()]$ evaluates to 0.

The expression `argfc` ("argument of first call"), and the following explanation why it is discardable but not copyable, are taken from [13]. We define

$$\texttt{argfc} = \texttt{callcc}(\lambda k.\lambda x.\texttt{throw}\, k(\lambda y.x)) : A \to A$$

Roughly speaking, when `argfc` is called with an argument x, it turns into the constant function returning x. That `argfc` is discardable is easily checked by sending the equation $\texttt{let}\, f\, \texttt{be}\, \texttt{argfc}\, \texttt{in}\, () \equiv ()$ through the CPS transform. To see that `argfc` is not copyable, let $M_1 = \texttt{let}\, f\, \texttt{be}\, \texttt{argfc}\, \texttt{in}\, (f, f)$ and

[5] A separator is more than a distinguisher, because a distinguisher does not require both sides to terminate.

$M_2 = (\mathtt{argfc}, \mathtt{argfc})$. Then $C[-] = \mathtt{let}\,(f_1, f_2)\,\mathtt{be} - \mathtt{in}\,(f_1 1; f_2 2)$ is a separator, because $C[M_1]$ evaluates to 1, whereas $C[M_2]$ evaluates to 2. Intuitively, during the evaluation of $C[M_1]$, evaluating $f_1 1$ causes backtracking to the binding of f, rebinding f (and therefore f_1 and f_2) to the constant function that returns 1. By contrast, during the evaluation of $C[M_2]$, evaluating $f_1 1$ causes backtracking to the binding of f_1, rebinding f_1 to the constant function that returns 1, and f_2 to $\mathtt{argfc}$. Next, the evaluation of $f_2 2$ causes backtracking to the binding of f_2, rebinding f_2 to the constant function that returns 2. For a formal, operational discussion of these separators, please see [13].

4.2 Adding Global State

Next, we shall add a global reference of type S to the λ_C-theory of a continuations monad $TA = R^{R^{(-)}}$. Formally, we apply the "side-effect" monad transformer $\langle T \mapsto (T(- \times S))^S \rangle$ to the continuations monad, resulting in a monad $T_S A = (R^{R^{A \times S}})^S$. Because $(R^{R^{A\times S}})^S$ is isomorphic to $(R^S)^{(R^S)^B}$, we still have a continuations monad, with response object R^S rather than R. This means that `callcc` and `throw` can still be implemented by the CPS-transform (with the former response type R replaced by $S \to R$). Moreover, we can define operators $\mathtt{assign} : S \to 1$ and $\mathtt{deref} : S$ as follows:

$$\overline{\mathtt{assign}} = \eta(\lambda x^S.\lambda k^{1\to S\to R}.\lambda s^S.k()x) \qquad \overline{\mathtt{deref}} = \lambda k^{S\to S\to R}.\lambda s^S.kss$$

From this definition, we can see that $\mathtt{assign}\,x$ forgets the state s and replaces it by x, and `deref` returns the state (the first s in kss) and also passes s on unchanged (the second s in kss).

Amazingly, these two operators for state provide exactly the witnesses that we need to complete Figure 3. In particular, they drive a wedge between copyability and distributivity. That $\mathtt{assign}\,x$ is copyable and that `deref` is copyable and discardable can be easily checked with the CPS transform. To see that `deref` is not repeatable, consider let $C[-] = \mathtt{assign}\,0; \mathtt{let}\,f\,\mathtt{be} - \mathtt{in}\,(\mathtt{assign}\,1; f())$ and observe that $C[\lambda().\mathtt{deref}]$ evaluates to 1 whereas $C[\mathtt{let}\,x\,\mathtt{be}\,\mathtt{deref}\,\mathtt{in}\,\lambda().x]$ evaluates to 0. To see that $\mathtt{assign}\,x$ is not repeatable, consider let $C[-] = \mathtt{let}\,f\,\mathtt{be} - \mathtt{in}\,(\mathtt{assign}\,0; f(); \mathtt{deref})$ and observe that $C[\lambda().\mathtt{assign}\,1]$ evaluates to 1 whereas $C[\mathtt{let}\,x\,\mathtt{be}\,\mathtt{assign}\,1\,\mathtt{in}\,\lambda().x]$ evaluates to 0. Showing that $\mathtt{assign}\,x$ is not discardable is trivial. This completes the discussion of Figure 3.

5 Conclusions

Comparison with traditional effects. The notion of effectoid is definable in the λ_C-calculus *without extra structure* (i.e. without operators like `assign`, `raise`, etc.). The same is true for the specific effectoids we introduced in this article. By contrast, traditional effects depend on extra structure. For example, an effect can be a set of exceptions that might be raised, or—in "side-effect analysis"—a set of actions of the form $\mathtt{deref}\,\pi$, $\mathtt{assign}\,\pi$, or $\mathtt{new}\,\pi$, where π is

a "program point" [9]. So our effectoids apply to a wider range of models than traditional effects.

At a first glance, our effectoids seem to be less expressive than some traditional effects (e.g. those that involve program points). However, centralisers and stabilisers may turn out to be very expressive in their own way, because they allow to specify a set of expressions to commute with or stabilise. (E.g. if n is a global variable that cannot be aliased, then Cen(`deref` n) should exclude exactly those expressions that can change n *and return*.)

The *notion* of effectoids (as opposed to the specific effectoids defined in this article) covers *some* traditional effects: for example, the set of expressions that only raise exceptions in a given set is obviously an effectoid.

Algebraic effectoids turn the traditional approach to effects upside down: in the traditional approach, we define concrete effects first, and may study their equational aspects later. For algebraic effectoids, we define equational properties first and may study later how they constrain concrete effects. Crucially, the equational characterisation allows to use an algebraic effectoid without knowing its concrete meaning.

Acknowledgements

Many thanks to Hayo Thielecke, Uday Reddy, and Brian Dunphy for discussions, and to Peter Selinger and Masahito Hasegawa for helpful comments.

References

1. Anna Bucalo, Carsten Führmann, and Alex Simpson. An equational notion of lifting monad. *Theoretical Computer Science*, to appear.
2. Andrzej Filinski. Recursion from iteration. *Lisp and Symbolic Computation,*, 7(1):11–38, 1994.
3. Carsten Führmann. Direct models of the computational lambda-calculus. In *Proceedings MFPS XV*, volume 20 of *Electronic Notes in Theoretical Computer Science*, New Orleans, 1999. Elsevier.
4. Carsten Führmann. *The structure of call-by-value*. PhD thesis, Division of Informatics, University of Edinburgh, 2000.
5. Masahito Hasegawa and Yoshihiko Kakutani. Axioms for recursion in call-by-value. *Higher-Order and Symbolic Computation*, To appear.
6. A. Kock. Strong functors and monoidal monads. *Archive der Mathematik*, 23:113–120, 1972.
7. E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Edinburgh Univ., Dept. of Comp. Sci., 1988.
8. E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.
9. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.
10. John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, October 1997.

11. Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11:207–260, 2001.
12. Hayo Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.
13. Hayo Thielecke. Using a continuation twice and its implications for the expressive power of `call/cc`. *Higher-Order and Symbolic Computation*, 12(1):47–74, 1999.