ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220608644

Algebraic Operations and Generic Effects

Article *in* Applied Categorical Structures · February 2003

DOI: 10.1023/A:1023064908962 · Source: DBLP

CITATIONS
131

READS
168

2 authors, including:

John Power
Macquarie University
168 PUBLICATIONS **3,476** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Category theoretic semantics for Logic Programming View project

Project The conceptual space of Lawvere theories View project

All content following this page was uploaded by John Power on 27 January 2014.

The user has requested enhancement of the downloaded file.

Algebraic Operations and Generic Effects

Gordon Plotkin and John Power [⋆]

Division of Informatics, University of Edinburgh, King's Buildings,
Edinburgh EH9 3JZ, Scotland

Abstract. Given a complete and cocomplete symmetric monoidal closed category V and a symmetric monoidal V-category C with cotensors and a strong V-monad T on C, we investigate axioms under which an ObC-indexed family of operations of the form $\alpha_x : (Tx)^v \longrightarrow (Tx)^w$ provides semantics for algebraic operations on the computational λ-calculus. We recall a definition for which we have elsewhere given adequacy results, and we show that an enrichment of it is equivalent to a range of other possible natural definitions of algebraic operation. In particular, we define the notion of generic effect and show that to give a generic effect is equivalent to giving an algebraic operation. We further show how the usual monadic semantics of the computational λ-calculus extends uniformly to incorporate generic effects. We outline examples and non-examples and we show that our definition also enriches one for call-by-name languages with effects.

1 Introduction

Eugenio Moggi, in [13, 15], introduced the idea of giving a unified category theoretic semantics for computational effects, modelling each of them in the Kleisli category of an appropriate strong monad T on a base category C with finite products. Examples of such effects are: nondeterminism, probabilistic nondeterminism, exceptions, interactive input/output, side-effects, and continuations with, in the case of Set, the corresponding monads being given by finite non-empty subsets, finite distributions, and the monads $(- + E)$ (for a set E of exceptions), $Tx = \mu y.((O \times y) + y^I + x)$ (for sets O of outputs and I of inputs), $(S \times -)^S$ (for a set S of states), and A^{A^-} (for a set A of answers), respectively. Moggi supported his use of the Kleisli construction by developing the computational λ-calculus or λ_c-calculus, for which such categories provide a sound and complete class of models. The computational λ-calculus is essentially the same as the simply typed λ-calculus except for making a careful systematic distinction between computations and values. It represents a fragment of a call-by-value programming language; in particular, it was designed to model fragments of Milner et al's language ML [17]. We describe it in Appendix A.

However, the calculus does not contain *operations*, the constructs that actually create the effects. For example, for binary nondeterminism, one has a binary

[⋆] This work has been done with the support of EPSRC grant GR/M56333.

nondeterministic choice operation symbol **or**. Operation symbols are polymorphic. For instance, in modelling nondeterminism, one has the rule (suppressing contexts):

$$\frac{\mathbf{M}, \mathbf{N} : \sigma}{\mathbf{M} \text{ or } \mathbf{N} : \sigma}$$

for all types σ. So, in order to give a semantics for **or**, a minimal demand is to model it by a natural transformation:

$$\vee_x : (Tx)^2 \longrightarrow Tx$$

Again, for exceptions, for each exception e, one has a nullary operation symbol $\mathbf{raise_e}$ for raising the exception e and a binary one $\mathbf{handle_e}$ for handling e. Similar computationally natural operations exist for all the other examples except, it seems, for continuations, which are accordingly beyond the scope of this paper. One should note that in such cases as interactive input/output and state, these operations may be infinitary (see below).

Moggi's computational metalanguage does contain operations, and his paper [14] includes semantics for them, but he only demanded naturality of the operations in C, and he did not develop a body of theory in support of that semantics. Here, by demanding the stronger coherence condition of parametrised naturality in C_T, we provide a notion of *algebraic* operations, which we support by equivalence theorems to indicate definitiveness of the axioms, and which are further supported by our development of a unified operational semantics in [20]. In all cases we can go further, taking the monad T to be generated by the operations subject to accompanying equations; this idea is explored in [22, 7].

Of the various operations, **handle** is of a different computational character and, although natural, it is not algebraic. Andrzej Filinski (personal communication) describes **handle** as a *deconstructor*, whereas the other operations are *constructors* (of effects). In this paper, we make the notion of constructor precise by identifying it with the notion of *algebraic* operation.

Algebraic operations are, in the sense we shall make precise, a natural generalisation, from *Set* to an arbitrary symmetric monoidal *V*-category C with cotensors, of the usual operations of universal algebra, taking T to be a strong *V*-monad on C. The key point is that the operations:

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

(where $(-)^v$ denotes cotensor with an object v of V) are parametrically natural in the Kleisli *V*-category C_T. Enrichment allows us to employ complex arities, i.e., objects of V, as in the case of local state—see below. (Enrichment by, e.g., $V = \omega Cpo$ allows us to handle recursion, cf [1], but that is a rather different matter, not involving complex arities; here ωCpo is the category of small ω-cpos, i.e, the category of posets with sups of ω-chains.) Parametrisation allows us to model open terms. And naturality in the Kleisli category means that the operations commute with evaluation contexts. In this paper, we do not make use of the possibility of V, C, and *Set* all being different, but it does seem to us to be the mathematically natural general level at which to formulate our results.

We could equally formulate the notion of algebraic operation in terms of an enriched version of closed *Freyd*-categories in the spirit of [1]. A preliminary version of our definition and results appeared in [21]. In [20], we gave an unenriched version of our definition together with a syntactic counterpart in terms of the computational λ-calculus, and we proved adequacy results: these required naturality in C_T.

Our most interesting result is essentially about the relationship between V-monads and Lawvere V-theories for suitable V [23, 7]: the result, in a more general setting than usual, characterises algebraic operations via *generic effects*. The general idea of generic effect seems to be new, although particular examples are known and their mathematical identification in the unenriched context is well-known. In the case of nondeterministic choice **or**, the corresponding generic effect is given by the term **arb** : **Bool**, which is, nondeterministically, either true or false, so that the equivalence:

$$\mathbf{arb} \;\equiv\; \mathbf{true\ or\ false}$$

holds; for the converse, we have:

$$\mathbf{M\ or\ N} \;\equiv\; \mathbf{if\ arb\ then\ M\ else\ N}$$

In programming languages, sometimes one sees syntax for operations and sometimes one sees it for generic effects; in cases where the operations are infinitary, one generally finds the generic effects used. We note that in [2], Benton, Hughes and Moggi essentially remarked on the construction that, to a generic effect, yields an operation, but they did not identify the notions of algebraic operation or generic effect or develop a theory.

The paper is organised as follows. In Section 2, we provide background on enriched monoidal categories, strong monads and cotensors, and we set notation for the paper. In Section 3, we recall and enrich the definition of algebraic operation given in [20], discuss corresponding extensions of the λ_c-calculus (including the infinitary case), and exhibit some simple reformulations of the definition. In Section 4, we give direct equivalent versions of these statements under the assumption that C is V-closed. In Section 5, we give a more substantial reformulation of the notion in terms of operations on homs, some when C is closed and some when C is not closed. In Section 6, we characterise algebraic operations as generic effects, and again consider the corresponding extensions of the λ_c-calculus. Finally, in Section 7, we characterise algebraic operations in terms of operations on the V-category T-Alg, as this gives an indication of how to incorporate call-by-name languages with computational effects into the picture. We give conclusions and an outline of possible future directions in Section 8.

2 Enriched monoidal categories, strong monads, and cotensors

We assume that V is a complete and cocomplete symmetric monoidal closed category: those are the conditions on V required for the preponderance of results of Kelly's definitive book [10] on enriched category theory. Implicitly using a larger universe, the category V-CAT of locally small V-categories has a symmetric monoidal structure, with $A * B$ having object set $ObA \times ObB$, with:

$$(A * B)((a,b),(a',b')) = A(a,a') \circ B(b,b')$$

where $\circ$ is the monoidal structure of V, with the evident composition.

If $V = Set$, an object of V-CAT is an ordinary locally small category, and the tensor product is just the ordinary product of categories. If $V = Poset$, an object of V-CAT is a locally small locally ordered category, and again the tensor product is just the ordinary product of locally ordered categories. A similar statement holds for $V = \omega Cpo$, the category of ω-cpos. More generally, for any cartesian closed V, the tensor product is simply given by product. But if, for example, V was the category $\omega Cpo_\perp$ of ω-cpos with least element, with maps preserving least upper bounds of ω-chains and the least element, the tensor product of V-categories is a little more subtle: the objects are still pairs (a,b), but the hom-object $(A * B)((a,b),(a',b'))$ is given by the tensor product $A(a,a') \circ B(b,b')$ in $\omega Cpo_\perp$ rather than by a product.

Using the tensor product on V-CAT, one can routinely define the notion of a *symmetric monoidal* V-category. Given V-functors $H, K : A \longrightarrow B$, a V-natural transformation $\alpha : H \Rightarrow K$ is just an ordinary natural transformation between the ordinary underlying functors $H_0, K_0 : A_0 \longrightarrow B_0$ subject to the evident enrichment of the naturality condition, i.e., such that for every pair (a,a') of objects of A, the diagram:

$$\begin{tikzcd}
A(a,a') \arrow[r, "H"] \arrow[d, "K"'] & B(Ha,Ha') \arrow[d, "{B(Ha,\alpha_{a'})}"] \\
B(Ka,Ka') \arrow[r, "{B(\alpha_a,Ka')}"'] & B(Ha,Ka')
\end{tikzcd}$$

commutes. A symmetric monoidal V-category may then be defined to be a V-category C together with an object I of C, a V-functor:

$$\otimes : C * C \longrightarrow C$$

and invertible V-natural transformations with components $a_{x,y,z} : (x \otimes y) \otimes z \longrightarrow x \otimes (y \otimes z)$, $l_x : I \otimes x \longrightarrow x$, $r_x : x \otimes I \longrightarrow x$, and $c_{x,y} : x \otimes y \longrightarrow y \otimes x$, subject to the usual axioms for a symmetric monoidal category [10, 12].

A monoidal V-category C is V-*closed* if for every object x of C, the V-functor $- \otimes x : C \longrightarrow C$ has a right V-adjoint. Note that if C is a monoidal V-category, the underlying ordinary category C_0 of C is a monoidal category, similarly for symmetry and closedness. This is ultimately because the functor $(-)_o : V\text{-}CAT \longrightarrow CAT$ is symmetric monoidal, i.e., it comes equipped with coherent maps

$$A_0 \times B_0 \longrightarrow (A * B)_0$$

and

$$1 \longrightarrow I_0$$

where I is the unit V-category, so respects the symmetric monoidal structure of V-CAT, which we used to define the notion of symmetric monoidal V-category.

A *strength* for a monad T on a monoidal category C consists of a natural transformation with components:

$$t_{x,y} : x \otimes Ty \longrightarrow T(x \otimes y)$$

subject to coherence conditions with respect to the monoidal structure of C and the monad structure of T [13–15]. The notion of strong monad, i.e, a monad together with a strength, is fundamental to Moggi's analysis. The notion of strength generalises readily from ordinary monads on monoidal categories to V-monads on monoidal V-categories by asking for the strength to be V-natural.

A strong monad T on a monoidal category C is said to have *Kleisli exponentials* if, for every object x of C, the functor $J(- \otimes x) : C \longrightarrow C_T$ has a right adjoint, where C_T is the Kleisli category for T and $J : C \longrightarrow C_T$ is the canonical functor. Moggi used strong monads with Kleisli exponentials in order to model the λ_c-calculus. If C is cartesian closed, it follows that every strong monad T on C has Kleisli exponentials. For any V-monad T on a V-category C, there is a Kleisli V-category C_T, and its underlying ordinary category is $(C_0)_{T_0}$. The notion of Kleisli exponential enriches to the assertion that for each object x of C, the V-functor $J(- \otimes x) : C \longrightarrow C_T$ has a right V-adjoint.

The notion of *cotensor* is fundamental for us here. Given an object v of V and an object x of a V-category C, a cotensor x^v is a representing object:

$$C(y, x^v) \cong [v, C(y, x)]$$

V-natural in y, where $[-, -]$ denotes the closed structure of V. In the case that $V = Set$, a cotensor is a power-object, i.e., x^v is just the product of v-many copies of x. In the case that $C = V$, cotensors are exactly exponentials in V. More generally, e.g., when $V = Poset$ and C is an arbitrary locally ordered category, cotensors include not only products of copies of an object, but also further constructions. For instance, if v is Sierpinski space, one cannot in general represent x^v as a product of copies of x: to give an arrow from 1 to x^v is eqivalent to giving a pair of arrows from 1 to x, with the first less than or equal to the second. The mathematical setting of our work therefore provides the opportunity for considerable generality, although our leading examples in this paper all have $V = Set$ or $V = C$, and so we need only power-objects and exponentials here. By a *finite cotensor*, we mean a cotensor with a finitely presentable object v. A *tensor* in a V-category may be defined to be a cotensor in the V-category C^{op}. If $C = Set$, the tensor is given by the coproduct of v-many copies of x, and if $C = V$, tensors are given by the monoidal structure of V.

We henceforth assume C is a symmetric monoidal V-category with cotensors, and $< T, \eta, \mu, st >$ is a strong V-monad on C with Kleisli V-exponentials. It is only for simplicity of exposition that we assume that C has all cotensors:

typically, we only need finite cotensors, but occasionally, for instance in modelling state, we need more (see Section 6). To make such a size condition precise requires a corresponding size condition on V, the simplest being that V be locally finitely presentable: we do not want to clutter the paper with details, which may be found, for example, in [23]. We do not take C to be V-closed in general: we shall need to assume it for some later results, but not for all of them.

Given V and C as we have assumed them, we define parametrised lifting $(-)^\dagger$ by:

$$C(y \otimes x, Tz) \xrightarrow{T} C(T(y \otimes x), T^2 z) \xrightarrow{C(st, \mu_z)} C(y \otimes Tx, Tz)$$

The operation $(-)^\dagger$ can be axiomatised by asserting coherence with respect to the monad structure of T: coherence with respect to the unit of T asserts that the construction yields an extension, and coherence with respect to the multiplication means that that extension is a lifting to a parametrised map of algebras from Tx to Tz.

The operation can be further extended by parametrisation in V. We use the same notation $(-)^\dagger$ for the composite, for any v in V:

$$C(y \otimes x, Tz) \xrightarrow{(-)^\dagger} C(y \otimes Tx, Tz) \xrightarrow{(-)^v} C((y \otimes Tx)^v, (Tz)^v) \longrightarrow C(y \otimes (Tx)^v, (Tz)^v)$$

where the unlabelled map is given by composition with the comparison map $y \otimes (Tx)^v \longrightarrow (y \otimes Tx)^v$ in C corresponding, via the definition of cotensor, to the map $v \longrightarrow C(y \otimes (Tx)^v, y \otimes Tx)$ in V given by the unit for the cotensor $v \longrightarrow C((Tx)^v, Tx)$ composed with $y \otimes - : C \longrightarrow C$. There is no danger of confusion between our two definitions of $(-)^\dagger$ because, putting $v = I$, we recover our former definition of $(-)^\dagger$ from the latter.

3 Algebraic operations and simple equivalents

In this section, we give the central definition of the paper. Its central feature is the precise choice of coherence condition. The technical content of the paper shows it to be the definitive choice and gives an equivalent formulation that also appears in programming practice. In programming language terms, the condition of the definition is that evaluation contexts $\mathbf{E}[-]$ commute with the computational effects induced by the operations. For example, in a nondeterministic programming context, e.g. [20], one is asserting the equivalence:

$$\mathbf{E}[\mathbf{M} \textbf{ or } \mathbf{N}] \;\equiv\; \mathbf{E}[\mathbf{M}] \textbf{ or } \mathbf{E}[\mathbf{N}]$$

for all evaluation contexts $\mathbf{E}[-]$.

Definition 1. *Given a strong V-monad T, an* algebraic operation *on T is an ObC-indexed family of maps:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

such that the diagram:

$$\begin{tikzcd}
C(y \otimes x, Tz) \arrow[r, "(-)^\dagger"] \arrow[d, "(-)^\dagger"'] & C(y \otimes (Tx)^v, (Tz)^v) \arrow[d, "{C(y \otimes (Tx)^v, \alpha_z)}"] \\
C(y \otimes (Tx)^w, (Tz)^w) \arrow[r, "{C(y \otimes \alpha_x, (Tz)^w)}"'] & C(y \otimes (Tx)^v, (Tz)^w)
\end{tikzcd}$$

commutes.

If $V = Set$, the definition of algebraic operation requires v and w to be sets, typically finite ones n and m. To give the data for an algebraic operation is equivalent to giving m ObC-indexed families of maps:

$$\alpha_x : (Tx)^n \longrightarrow Tx$$

and the condition is the assertion that for each of these ObC-indexed families, for every map $f : y \otimes x \longrightarrow Tz$ in C, the diagram:

$$\begin{tikzcd}
y \otimes (Tx)^n \arrow[r, "\langle f^\dagger \cdot (y \otimes \pi_i) \rangle_{i=1}^n"] \arrow[d, "y \otimes \alpha_x"'] & (Tz)^n \arrow[d, "\alpha_z"] \\
y \otimes Tx \arrow[r, "f^\dagger"'] & Tz
\end{tikzcd}$$

commutes.

In the following examples we put $V = C = Set$, except for the case of local state; we also briefly consider partiality, when we take instead $V = Set$ and $C = \omega Cpo$ or $Dcpo$ (*dcpos* are partial orders with lubs of all directed sets).

Example 1. Nondeterminism The nonempty finite power-set monad T supports a binary choice operation [18, 1]:

$$\vee_x : (Tx)^2 \longrightarrow Tx$$

where $\vee_x(u, v) = u \cup v$. It is routine to verify the coherence condition. This generalises from Set to ωCpo, where the power-domain can be characterised as giving the free semilattice together with a least element on an ω-cpo [19], and it is the usual semantics for **or**. It follows from its freeness that the operation of the semilattice is an algebraic operation (see Section 7), but one can also verify the coherence condition directly.

Example 2. Probabilistic Nondeterminism Similarly to the situation for nondeterminism, the monad of finite distributions for probabilistic nondeterminism [20] on *Set* supports a binary probabilistic choice operation $+_{\mathrm{r}}$ for every real number r in the interval $[0, 1]$. And just as for nondeterminism, the semantic operation $+_{\mathrm{r}}$ models a corresponding probabilistic choice operation symbol $+_{\mathrm{r}}$. This also generalises from *Set*, here to the probabilistic power-domain on *Dcpo* [8, 9, 6].

Example 3. Exceptions The monad $Tx = x + E$ for exceptions supports E evident nullary operations, one for each e in E; these model the nullary operation symbols $\mathbf{raise_e}$ for raising exceptions. For each e in E, the natural transformation:

$$handle_e : (Tx)^2 \longrightarrow Tx$$

defined by $handle_e(e, b) = e$ and $handle_e(a, b) = b$ for all $a \neq e$, is used to model the exception-handling binary operation $\mathbf{handle_e}$. As we have already remarked, it does not satisfy the coherence condition required of an algebraic operation; to see this consider the simpler unparametrised case, and take $x = Bool\,(= 2)$. Then, for any $f : Bool \to T(Bool)$ such that $f(true) = e$, we have $f^{\dagger}(handle_e(true, false)) = f^{\dagger}(true) = e$ and $handle_e(f^{\dagger}(true), f^{\dagger}(false))$ $= handle_e(e, f^{\dagger}(false)) = f^{\dagger}(false)$ (putting $true$, $false$ for the two elements of *Bool*) and the two need not be equal. For the combination with partiality, one uses the monad $(- + E)_{\perp}$, where $(-)_{\perp}$ is the lifting monad that adds a new least element; the corresponding versions of the operations are immediate.

Example 4. Interactive Input/Output The monad $\Sigma^* \times -$ for printing supports printing operations [19]. Both the monad for printing and the printing operations extend to the monad $Tx = \mu y.((O \times y) + y^I + x)$ for interactive input/output, where one has *read* and *write* operations with semantics the operations:

$$read_x : (Tx)^I \longrightarrow Tx$$

and

$$write_x : Tx \longrightarrow (Tx)^O$$

(the alphabet Σ has here been renamed to the output set O). The operation $read_x$ is obtained using the isomorphism between Tx and $(O \times Tx) + (Tx)^I + x$ and $write_x$ is is the transpose of the evident map $O \times Tx \to Tx$ obtained using the isomorphism. Observe the non-commutativity of the monad in this example and the fact that the operation $read_x$ is infinitary (assuming that I is infinite). For the combination with partiality, one uses the monad $Tx = \mu y.((O \times y) + y^I + x)_{\perp}$.

Example 5. Global State Let L be a set of locations and let Val be a set of values. We denote the exponential $[L, Val]$ by S, representing a set of states. Let T be the monad $(S \times -)^S$. There are natural infinitary algebraic operations:

$$lookup_x : (Tx)^{Val} \longrightarrow (Tx)^L$$

and

$$update_x : Tx \longrightarrow (Tx)^{L \times Val}$$

given by:

$$lookup(f)(l)(s) = f(s(l))$$

and

$$update(a)(l,b) = a(s[b/l])$$

where $s[b/l]$ is the same as s everywhere, except at l where $s[b/l](l) = b$. Note that we have another infinitary operation, namely *lookup* (assuming that Val is infinite). For the combination with partiality, one uses the monad $(S \times -)^S_\perp$.

For an idea of the setting of the corresponding algebraic operations in one of the more complex examples, consider the following.

Example 6. Local State In order to model local state, one might take C to be a category of the form $[W, Set]$, where W is a category of worlds. Many different categories W of worlds have appeared in the literature, depending upon the specific programming languages or their properties being considered [16]. Putting $W = I$, the category of finite sets and injections, a monad for local state is given by:

$$(Tx)n = (\int^{m\epsilon(n/I)} (Sm \times x(m)))^{Sn}$$

where $\int$ denotes a coend [10, 12], and where $Sn = Val^n$ for a set Val of values. A detailed exploration of this monad appears in [22]; it is closely related to but simpler than a monad in Paul Levy's thesis [11]. As before, one may consider operations $lookup_x : (Tx)^{Val} \longrightarrow (Tx)^L$ and $update_x : Tx \longrightarrow (Tx)^{L \times Val}$ but now Val is treated as a constant functor and L becomes the inclusion of I in Set. So here, we are taking $V = C$ and are using the cartesian structure for enrichment. However there is another natural operation:

$$block_x : [L, Tx] \to Tx$$

for declaring, initialising and then using a new location; here we are using a second closed structure on C, namely Day's convolution closed monoidal structure [3]. Rather than give explicit definitions of these operations here, we take them as defined in terms of the corresponding generic effects, considered below. For partiality, we would instead use a category of the form $[I, \omega Cpo]$ and the monad $(Tp)n = (\int^{m\epsilon(n/I)} (Sm \times p(m)))^{Sn}_\perp$.

One can also consider combinations of these examples, for instance to model internal and external choice operations or to model state and nondeterminism. The first three of the above examples are treated in detail in [20], where a unified operational semantics is given for them.

Having seen several examples of operations, we now consider how they might appear as semantics for corresponding syntactic term-forming constructs in various possible extensions of the λ_c-calculus. We assume the enriched form of the semantics of the λ_c-calculus mentioned in the Appendix, since it is as easy to handle here as the unenriched case. So we assume The V-enriched structure of C is actually a V-product structure and that C has V-enriched Kleisli exponentials.

For finitary operations (say just V-natural transformations):

$$op_x : (Tx)^n \to Tx$$

one considers an n-ary operation symbol **op** and adds terms of the form:

$$\mathbf{op}(\mathbf{M_1}, \ldots, \mathbf{M_n})$$

to the λ_c-calculus with the type inference rule:

$$\frac{\mathbf{M_1}, \ldots, \mathbf{M_n} : \sigma}{\mathbf{op}(\mathbf{M_1}, \ldots, \mathbf{M_n}) : \sigma}$$

and then:

$$[\![\mathbf{op}(\mathbf{M_1}, \ldots, \mathbf{M_n})]\!] = op_{[\![\sigma]\!]^\circ}([\![\mathbf{M_1}]\!], \ldots, [\![\mathbf{M_n}]\!])$$

Again, suppose one has an operation:

$$op_x : (Tx)^n \to (Tx)^P$$

where P is thought of as a parameter space. For a category such as $V = Set$, one possibility is to add an n-ary operation symbol $\mathbf{op_e}$ for each e in P (as, essentially, was done in the case of exceptions). Another possibility, in the general enriched case is to add a new base type symbol **Par** with a suitable collection of constants and then add terms of the form:

$$\mathbf{op}(\mathbf{N}, \mathbf{M_1}, \ldots, \mathbf{M_n})$$

with the type inference rule:

$$\frac{\mathbf{N} : \mathbf{Par} \quad \mathbf{M_1}, \ldots, \mathbf{M_n} : \sigma}{\mathbf{op}(\mathbf{N}, \mathbf{M_1}, \ldots, \mathbf{M_n}) : \sigma}$$

For the semantics, we interpret **Par** as **P**, the tensor of P with 1 (which we assume exists!) and then, noting that the general V-natural isomorphism:

$$C(x \times \mathbf{u}, Ty) \cong C(x, (Ty)^u)$$

for the tensor $\mathbf{u}$ of a V-object u with 1 allows us to transpose, the semantics of $\mathbf{op}(\mathbf{N}, \mathbf{M_1}, \ldots, \mathbf{M_n})$ is given using the transpose of $op_{[\![\sigma]\!]}$.

The case of an operation of the form:

$$op_x : (Tx)^A \to (Tx)^P$$

is trickier. If A is an infinite set, that would suggest the use of infinite terms, which is not compatible with having a finitary programming language, and so in the examples of which we are aware, one rather employs syntax for the corresponding generic effects. However there is something one can do, which is even available in the general enriched case. One adds base type symbols **Arg** and **Par** denoting the relevant tensors, and suitable additional base constants and then adds terms of the form:

$$\mathbf{op}(\mathbf{N}, (\mathbf{x} : \mathbf{Arg})\mathbf{M})$$

with the type inference rule:

$$\frac{\mathbf{\Gamma \vdash N : Par} \quad \mathbf{\Gamma, x : Arg \vdash M} : \sigma}{\mathbf{\Gamma \vdash op(N, (x : Arg)M)} : \sigma}$$

For the semantics, one composes the transpose of $op_{[\![\sigma]\!]}$ with the semantics of $\mathbf{N}$ and the transpose of the semantics of $\mathbf{M}$. Using this form of operation construct, we can give an equivalence expressing the algebraicity of operations at the level of the λ_c-calculus, of the kind we have already seen in the case of nondeterminism:

$$\mathbf{E[op(N, (x : Arg)M)]} \equiv \mathbf{op(N, (x : Arg)E[M])}$$

for suitably typed $\mathbf{E}, \mathbf{M}, \mathbf{N}$. The proof of this equivalence follow from algebraicity and remarks in the appendix on the semantics of evaluation contexts. There are evident simpler equivalences for the other forms of operation construct we have just given.

There are several equivalent formulations of the coherence condition of the definition of algebraic operation. Decomposing it in a maximal way, we have

Proposition 1. *An ObC-indexed family of maps:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

is an algebraic operation if and only if

1. *α is natural in C*
2. *α respects st in the sense that:*

$$\begin{tikzcd}
y \otimes (Tx)^v \arrow[r] \arrow[d, "y \otimes \alpha_x"'] & (y \otimes Tx)^v \arrow[r, "st^v"] & (T(y \otimes x))^v \arrow[d, "\alpha_{y \otimes x}"] \\
y \otimes (Tx)^w \arrow[r] & (y \otimes Tx)^w \arrow[r, "st^w"'] & (T(y \otimes x))^w
\end{tikzcd}$$

   commutes, where the unlabelled maps are comparison maps determined by the universal property of cotensors
3. *α respects μ in the sense that:*

$$\begin{tikzcd}
(T^2x)^v \arrow[r, "\mu_x^v"] \arrow[d, "\alpha_{Tx}"'] & (Tx)^v \arrow[d, "\alpha_x"] \\
(T^2x)^w \arrow[r, "\mu_x^w"'] & (Tx)^w
\end{tikzcd}$$

   commutes.

Proof. It is immediately clear from our formulation of the definition and the proposition that the conditions of the proposition imply the coherence requirement of the definition. For the converse, to prove V-naturality in C, put $y = I$, the unit of the monoidal structure of C, use composition with η_z applied to $C(x, z)$, and apply the coherence condition of the definition. For coherence with respect to st, take $f : y \otimes x \longrightarrow Tz$ to be $\eta_{y \otimes x}$. And for coherence with respect to μ, put $y = I$ and take f to be id_{Tx}.

There are other interesting decompositions of the coherence condition of the definition too. In the above, we have taken T to be an endo-V-functor on C. But one often also writes T for the right V-adjoint to the canonical V-functor $J : C \longrightarrow C_T$ as the behaviour of the right adjoint on objects is given precisely by the behaviour of T on objects. So with this overloading of notation, we have V-functors $(T-)^v : C_T \longrightarrow C$ and $(T-)^w : C_T \longrightarrow C$, we can speak of V-natural transformations between them, and we have the following proposition, for which a proof is routine.

Proposition 2. *An ObC-indexed family of maps:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

is an algebraic operation if and only if α is V-natural in C_T and α respects st.

In another direction, as we shall investigate further below, it is sometimes convenient to separate the μ part of the coherence condition from the rest of it. We can do that with the following somewhat technical result, again a with routine proof.

Proposition 3. *An ObC-indexed family:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

forms an algebraic operation if and only if α respects μ and:

$$\begin{tikzcd}
C(y \otimes x, z) \arrow[r, "(-)^*"] \arrow[d, "(-)^*"'] & C(y \otimes (Tx)^v, (Tz)^v) \arrow[d, "{C(y \otimes (Tx)^v, \alpha_z)}"] \\
C(y \otimes (Tx)^w, (Tz)^w) \arrow[r, "{C(y \otimes \alpha_x, (Tz)^w)}"'] & C(y \otimes (Tx)^v, (Tz)^w)
\end{tikzcd}$$

commutes, where $(-)^*$ is defined, parametrically in V, by the composition of $T : C(y \otimes x, z) \longrightarrow C(T(y \otimes x), Tz)$ with the composite:

$$C(T(y \otimes x), Tz) \xrightarrow{C(st, Tz)} C(y \otimes Tx, Tz) \xrightarrow{(-)^v} C((y \otimes Tx)^v, (Tz)^v) \longrightarrow C(y \otimes (Tx)^v, (Tz)^v)$$

4 Equivalent formulations if C is V-closed

For our more interesting results, we first assume C is V-closed, explain the results in those terms, and later drop the closedness condition and explain corresponding variants of the results. We give the results using closedness first because they are more elegant, so perhaps easier to understand. For the results in this section, we shall assume C is V-closed.

Let the V-closed structure of C be denoted by $[-,-]$. Given a V-functor $H : C \longrightarrow C$, an *enrichment* of H is a C-functor $K : C \longrightarrow C$ such that H is the underlying V-functor of K, i.e., H and K agree on objects and the monoidal V-functor $C(I,-) : C \longrightarrow V$ sends $[Kx, Ky]$ to $C(Hx, Hy)$, respecting composition. Enrichment of a V-natural transformation does not alter the data but requires the stronger property of a commutativity in C rather than one in V. With these definitions, one can speak of the enrichment of a V-monad T to a C-monad.

Given a V-monad $< T, \eta, \mu >$ on C, to give a V-strength for T is equivalent to giving an enrichment of T in C: given a strength, one has an enrichment:

$$T_{x,y} : [x,y] \longrightarrow [Tx, Ty]$$

given by the transpose of:

$$[x,y] \otimes Tx \xrightarrow{st} T([x,y] \otimes x) \xrightarrow{Tev} Ty$$

and given an enrichment of T, one has a V-strength given by the transpose of:

$$x \longrightarrow [y, x \otimes y] \xrightarrow{T_{y,x\otimes y}} [Ty, T(x \otimes y)]$$

It is routine to verify that the axioms for a V-strength for a V-functor are equivalent to the axioms for an enrichment of the V-functor, and that the coherence axioms of a V-strength with respect to η and μ are equivalent to their enrichment; and so the axioms for a V-strength for a V-monad are equivalent to the axioms for an enrichment of the V-monad. So, given a V-strong V-monad $< T, \eta, \mu, st >$ on C, the monad T is enriched in C, and so is the V-functor $(-)^v : C \longrightarrow C$.

The V-category C_T also canonically acquires an enrichment in C, i.e, the homobject $C_T(x,y)$ of C_T in V lifts to a homobject in C: the object $[x, Ty]$ of C acts as a homobject, applying the V-functor $C(I,-) : C \longrightarrow V$ to it giving the V-homobject $C_T(x,y)$; composition:

$$C_T(y,z) \circ C_T(x,y) \longrightarrow C_T(x,z)$$

in V lifts to a map in C:

$$[y, Tz] \otimes [x, Ty] \longrightarrow [x, Tz]$$

determined by taking a transpose and applying evaluation maps twice and each of the V-strength and the multiplication once; and identities and the axioms for a V-category lift too.

The canonical V-functor $J : C \longrightarrow C_T$ becomes a C-enriched functor with a C-enriched right adjoint. The main advantage of the closedness condition for us is that it allows us to dispense with the parametrisation of the V-naturality, or equivalently with the coherence with respect to the V-strength, as follows.

Proposition 4. *If C is V-closed, an ObC-indexed family:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

forms an algebraic operation if and only if:

$$\begin{tikzcd}
{[x, Tz]} \arrow[r, "(-)^v \cdot [Tx, \mu_z] \cdot T_{x,Tz}"] \arrow[d, "(-)^w \cdot [Tx, \mu_z] \cdot T_{x,Tz}"'] & {[(Tx)^v, (Tz)^v]} \arrow[d, "{[(Tx)^v, \alpha_z]}"] \\
{[(Tx)^w, (Tz)^w]} \arrow[r, "{[\alpha_x, (Tz)^w]}"'] & {[(Tx)^v, (Tz)^w]}
\end{tikzcd}$$

commutes.

The left-hand vertical map in the diagram here is exactly the behaviour of the C-enriched functor
$(T-)^w : C_T \longrightarrow C$ on homs, and, correspondingly, the top horizontal map is exactly the behaviour of the C-enriched functor $(T-)^v : C_T \longrightarrow C$ on homs. As explained in Section 3, the data for an enriched natural transformation between enriched functors is identical to the data for an ordinary natural transformation between the underlying ordinary functors. But the enriched naturality condition is stronger: rather than saying that, for every map, a square commutes, it internalises the idea of naturality in terms of the homobject. So, a priori, it is a stronger condition than that of ordinary naturality. The coherence condition in the proposition is precisely the statement that α forms a C-enriched natural transformation from the C-enriched functor $(T-)^v : C_T \longrightarrow C$ to the C-enriched functor $(T-)^w : C_T \longrightarrow C$.

Proof. Given an object y of C, applying the V-functor $C(y, -) : C \longrightarrow V$ to the coherence condition here yields the coherence condition of the definition. The converse holds by the (ordinary) Yoneda lemma.

The same argument can be used to give a further characterisation of the notion of algebraic operation if C is V-closed by modifying Proposition 3. This yields

Proposition 5. *If C is V-closed, an ObC-indexed family:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

forms an algebraic operation if and only if α respects μ and:

$$\begin{tikzcd}
{[x,z]} \arrow[r, "(-)^v \cdot T_{x,z}"] \arrow[d, "(-)^w \cdot T_{x,z}"'] & {[(Tx)^v,(Tz)^v]} \arrow[d, "{[(Tx)^v,\alpha_z]}"] \\
{[(Tx)^w,(Tz)^w]} \arrow[r, "{[\alpha_x,(Tz)^w]}"'] & {[(Tx)^v,(Tz)^w]}
\end{tikzcd}$$

commutes.

This proposition says that if C is V-closed, an algebraic operation is exactly a C-enriched natural transformation from the C-enriched functor $(T-)^v : C \longrightarrow C$ to the C-enriched functor $(T-)^w : C \longrightarrow C$ that is coherent with respect to μ.

5 Algebraic operations as operations on homs

In our various formulations of the notion of algebraic operation so far, we have always had an ObC-indexed family:

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

and considered equivalent conditions on it under which it might be called an algebraic operation. In computing, this amounts to considering an operator on terms. But there is another approach in which homs of the V-category C_T may be seen as primitive, regarding them as sets or ω-cpos or the like of programs. This was the underlying idea of the reformulation [1] of the semantics for finite nondeterminism of [18]. So we should like to reformulate the notion of algebraic operation in these terms. Proposition 4 allows us to do that. In order to explain the reason for the coherence conditions, we shall start by expressing the result assuming C is V-closed; after which we shall drop the closedness assumption and see how the result can be re-expressed using parametrised naturality.

We first need to explain an enriched version of the Yoneda lemma as in [10]. If D is a C-enriched category, then D^{op} may also be seen as a C-enriched category: one requires symmetry of C, which we have consistently assumed, to make D^{op} a C-enriched category, but one can adapt the following line of argument if ever non-symmetric examples of C arise. In general, we do not want to restrict ourselves to the assumption that C is complete. But if C was complete and D was small, we would have a C-enriched functor category $[D^{op}, C]$ and a C-enriched Yoneda embedding:

$$Y_D : D \longrightarrow [D^{op}, C]$$

The C-enriched Yoneda embedding Y_D would be a C-enriched functor and it would be fully faithful in the strong sense that the map:

$$D(x, y) \longrightarrow [D^{op}, C](D(-, x), D(-, y))$$

would be an isomorphism in the category C: see [10] for details. By applying the V-functor $C(I,-) : C \longrightarrow V$, this would induce an isomorphism from the homobject of V underlying $D(x,y)$ to the object of V underlying the homobject from the C-enriched functor $D(-,x) : D^{op} \longrightarrow C$ to the C-enriched functor $D(-,y) : D^{op} \longrightarrow C$: if $V = Set$, the former object is the set of maps from x to y, and the latter is the set of C-enriched natural transformations from $D(-,x)$ to $D(-,y)$.

This is the result we need, except that, as we wrote, we do not want to assume that C is complete, and the C-enriched categories of interest to us are of the form C_T, so in general are not small. So we need to generalise the above argument by dropping its assumptions that C is complete and D is small. This is not difficult, but it goes a little beyond the standard formulation of enriched category theory in [10]. Both assumptions can be avoided by embedding C into a larger universe C' just as one can embed Set into a larger universe Set' when necessary: the required mathematics for the enriched analysis appears in [10]. We still have what can reasonably be called a Yoneda embedding of D into $[D^{op}, C]$, with both categories regarded as C'-enriched rather than C-enriched, and it is still fully faithful as a C'-enriched functor.

In fact, we can formulate our result even without reference to C' by stating a restricted form of the enriched Yoneda lemma: letting $Fun_C(D^{op}, C)$ denote the V'-category, for a suitable extension V' of V, of C-enriched functors from D^{op} to C, the underlying V-functor:

$$D \longrightarrow Fun_C(D^{op}, C)$$

of the Yoneda embedding is fully faithful.

We use this latter statement both here and in the following section. Now for our main result of this section under the assumption that C is V-closed.

Theorem 1. *If C is V-closed, to give an algebraic operation is equivalent to giving an $ObC^{op} \times ObC$ family of maps:*

$$a_{y,x} : [y, Tx]^v \longrightarrow [y, Tx]^w$$

that is C-natural in y as an object of C^{op} and C-natural in x as an object of C_T, i.e., such that:

$$\begin{tikzcd}
{[y,Tx]^v \otimes [y',y]} \arrow[r, "\cong"] \arrow[d, "a_{y,x} \otimes {[y',y]}"'] & {[y,(Tx)^v] \otimes [y',y]} \arrow[r, "comp"] & {[y',Tx]^v} \arrow[d, "a_{y',x}"] \\
{[y,Tx]^w \otimes [y',y]} \arrow[r, "\cong"'] & {[y,(Tx)^w] \otimes [y',y]} \arrow[r, "comp"'] & {[y',Tx]^w}
\end{tikzcd}$$

and

$$\begin{tikzcd}
{[x,Tz] \otimes [y,Tx]^v} \arrow[r] \arrow[d, "{[x,Tz] \otimes a_{y,x}}"'] & {([x,Tz] \otimes [y,Tx])^v} \arrow[r, "comp_K^v"] & {[y,Tz]^v} \arrow[d, "a_{y,z}"] \\
{[x,Tz] \otimes [y,Tx]^w} \arrow[r] & {([x,Tz] \otimes [y,Tx])^w} \arrow[r, "comp_K^w"'] & {[y,Tz]^w}
\end{tikzcd}$$

commute, where comp is the C-enriched composition of C, the unlabelled isomorphisms of the first diagram are determined by the fact that $[y,-] : C \longrightarrow C$ is a right adjoint, so preserves cotensors, $comp_K$ is C-enriched Kleisli composition, and the unlabelled maps of the second diagram are determined by the universal property of cotensors.

Proof. It follows from our C-enriched version of the Yoneda lemma that to give the data together with the first axiom of the theorem is equivalent to giving an ObC-indexed family:

$$\alpha : (Tx)^v \longrightarrow (Tx)^w$$

By a further application of our C-enriched version of the Yoneda lemma, it follows that the second condition of the theorem is equivalent to the coherence condition of Proposition 4.

As mentioned earlier, we can still state essentially this result even without the condition that C be closed. There are two reasons for this. First, for the paper, we have assumed the existence of Kleisli exponentials, as are essential in order to model λ-terms. But most of the examples of the closed structure of C we have used above are of the form $[y, Tx]$, which can equally be expressed as the Kleisli exponential $y \Rightarrow x$. The Kleisli exponential routinely extends to a V-functor:

$$- \Rightarrow - : C_T^{op} \times C_T \longrightarrow C$$

Second, in the above, we made one use of a construct of the form $[y', y]$ with no T protecting the second object. But we can replace that by employing the V'-enriched Yoneda lemma to express the first condition of the theorem in terms of homobjects of V of the form $C(w \otimes y', y)$.

Summarising, we have

Corollary 1. *To give an algebraic operation is equivalent to giving an $ObC^{op} \times ObC$ family of maps:*

$$a_{y,x} : (y \Rightarrow x)^v \longrightarrow (y \Rightarrow x)^w$$

in C, such that for all objects z' and y' of C, the diagram:

$$\begin{tikzcd}
C(z' \otimes y', y) \arrow[r, "(- \Rightarrow x)^v"] \arrow[d, "(- \Rightarrow x)^w"'] & C((y \Rightarrow x)^v, ((z' \otimes y') \Rightarrow x)^v) \arrow[d, "C((y \Rightarrow x)^v, a_{z' \otimes y', x})"] \\
C((y \Rightarrow x)^w, ((z' \otimes y') \Rightarrow x)^w) \arrow[r, "C(a_{y,x}, ((z' \otimes y') \Rightarrow x)^w)"'] & C((y \Rightarrow x)^v, ((z' \otimes y') \Rightarrow x)^w)
\end{tikzcd}$$

commutes, and for every object z of C, the diagram:

$$\begin{tikzcd}
(x \Rightarrow z) \otimes (y \Rightarrow x)^v \arrow[r] \arrow[d, "(x \Rightarrow z) \otimes a_{y,x}"'] & ((x \Rightarrow z) \otimes (y \Rightarrow x))^v \arrow[r, "comp_K^v"] & (y \Rightarrow z)^v \arrow[d, "a_{y,z}"] \\
(x \Rightarrow z) \times (y \Rightarrow x)^w \arrow[r] & ((x \Rightarrow z) \otimes (y \Rightarrow x))^w \arrow[r, "comp_K^w"'] & (y \Rightarrow z)^w
\end{tikzcd}$$

commutes, where $comp_K$ is the canonical internalisation of Kleisli composition.

6 Algebraic operations as generic effects

In this section, we apply our formulation of the C-enriched Yoneda lemma to characterise algebraic operations in entirely different terms again as maps in C_T, i.e., in terms of generic effects. Observe that if C has a tensor $\mathbf{v}$ of v with I, the V-functor $(T-)^v : C_T \longrightarrow C$ is isomorphic to the V-functor $\mathbf{v} \Rightarrow - : C_T \longrightarrow C$. If C is V-closed, the V-functor $\mathbf{v} \Rightarrow -$ enriches canonically to a C-enriched functor, namely the representable C-functor $C_T(\mathbf{v}, -) : C_T \longrightarrow C$, where C_T is regarded as a C-enriched category. So by Proposition 4 together with our C-enriched version of the Yoneda lemma, we have

Theorem 2. *If C is V-closed, the C-enriched Yoneda embedding induces a bijection between maps $\mathbf{w} \longrightarrow \mathbf{v}$ in C_T and algebraic operations:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

Spelling out the bijection, given a map $f : \mathbf{w} \longrightarrow \mathbf{v}$ in C_T, the corresponding algebraic operation is:

$$(Tx)^v \xrightarrow{\cong} [\mathbf{v}, Tx] \xrightarrow{[f, Tx]} [\mathbf{w}, Tx] \xrightarrow{\cong} (Tx)^w$$

and given an algebraic operation α_x, the corresponding map in C_T is:

$$I \longrightarrow [\mathbf{v}, T\mathbf{v}] \xrightarrow{\cong} (T\mathbf{v})^v \xrightarrow{\alpha_{\mathbf{v}}} (T\mathbf{v})^w \xrightarrow{\cong} [\mathbf{w}, T\mathbf{v}]$$

This result is essentially an enriched version of the identification of maps in a Lawvere theory with operations of the Lawvere theory [23]. It generalises that identification in three ways: it uses enrichment, it does not use finitariness, and it allows V and C to differ. In studying Lawvere theories, where $V = Set$, one typically restricts arities to be natural numbers, but we sometimes have infinitary operations, for instance in modelling global state and interactive input/output, hence our need for an infinitary statement. Moreover, to model local state, we need arities that are not sets at all, but are presheaves (see Example 11), hence our generalisation from Set. For an enriched version of Lawvere's idea without the finitariness but with the restriction to $C = V$, see [4].

The result as we have formulated it above is an elegant instance of the enriched Yoneda lemma [10]. But once again, using parametrisation but losing a little elegance, we can generalise it to avoid the closedness assumption on C.

Theorem 3. *Functoriality of* $- \Rightarrow - : C_T^{op} \times C_T \longrightarrow C$ *in its first variable induces a bijection from the small set of maps* $\mathbf{w} \longrightarrow \mathbf{v}$ *in* C_T *to the set of algebraic operations:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

The constructions are a rewriting of those described under the supposition of closedness of C. The bijection asserted in the statement of the theorem can be extended to giving an isomorphism between the homobject $C_T(\mathbf{w}, \mathbf{v})$ of V and an object of V given by lifting the set of algebraic operations of corresponding arity on T to V. The correspondence is still essentially an instance of the Yoneda embedding.

We regard this theorem as the most interesting result of the paper. Given an algebraic operation $\alpha_x : (Tx)^v \longrightarrow (Tx)^w$, we define the *generic effect* corresponding to α to be the map $\alpha_g : \mathbf{w} \longrightarrow \mathbf{v}$ in C_T determined by the theorem. So α_g is a constant of type the arity of α. This correspondence has programming implications, suggesting natural extensions to the λ_c-calculus, which we now study (we omit consideration of the straightforward addition of partiality).

Example 7. Nondeterminism To give a binary algebraic operation on a strong monad T is equivalent to giving a constant of type 2, i.e, a map from 1 to 2 in C_T, or equivalently, a map from 1 to $T2$ in C. For instance, let T be the non-empty finite powerset monad. Given the union operation for nondeterminism $\vee_x : (Tx)^2 \longrightarrow Tx$, the corresponding constant arb is given by $true \vee_2 false$. So, in order to model binary nondeterminism, one could extend the λ_c-calculus either by adding a binary nondeterministic choice operation symbol **or** or equivalently by adding a constant **arb** of type **Bool**. Similar remarks hold for probabilistic nondeterminism where the constant corresponding to $+_r$ is $rand_r$, the distribution on *Bool* giving *true* with probability r and $false$ with probability $1 - r$.

Example 8. Exceptions The generic effect corresponding to a nullary operation $raise_e$ has type T(0). In programming language terms that means using a zero type, and that is not normally done, although there is, in principle, no reason why not. Presumably it would seem rather odd as there would be no possibility of any value being returned of that type, so the *only* point of having the type would be for the occurrence of effects.

Example 9. Interactive Input/Output As we have said above, the operations for interactive input/output for the monad $Tx = \mu y.((O \times y) + y^I + x)$ on Set are $read_x : (Tx)^I \longrightarrow Tx$ and $write_x : Tx \longrightarrow (Tx)^O$, and so the corresponding generic effects are:

$$read : 1 \longrightarrow TI$$

and

$$write : O \longrightarrow T1$$

In programming terms, it would be natural to extend the λ_c-calculus to include input and output basic types **In** and **Out** denoting I and O (together with appropriate basic constants) and to add term constructs:

$$\mathbf{read} : \mathbf{In} \qquad \frac{\mathbf{M} : \mathbf{Out}}{\mathbf{write}\ \mathbf{M} : \mathbf{1}}$$

modelled by *read* and *write* respectively. Note how the infinitary operation has been replaced by a finitary generic effect, albeit involving an infinite type.

Example 10. Global State The operations here are $lookup_x : (Tx)^{Val} \longrightarrow (Tx)^L$ and $update_x : Tx \longrightarrow (Tx)^{L \times Val}$, of which the first is infinitary. The corresponding generic effects are:

$$deref : L \longrightarrow T(Val) \qquad assign : L \times Val \longrightarrow T1$$

defined by:

$$deref(l)(s) = (s, s(l)) \qquad assign(l, v)(s) = (s[v/l], *)$$

The corresponding extensions of the λ_c-calculus has basic types **Val** and **Loc** modelled by Val and L, and term constructs for dereferencing and assignment:

$$\frac{\mathbf{M}:\mathbf{Loc}}{!\mathbf{M}:\mathbf{Val}} \qquad \frac{\mathbf{M}:\mathbf{Loc}\ \ \mathbf{N}:\mathbf{Val}}{(\mathbf{M} := \mathbf{N}):\mathbf{1}}$$

modelled by the generic effects. Happily this is exactly how side-effects are expressed in ML (modulo the fact that there one can have references to values of any type).

Example 11. Local State Although two separate enrichments are used, in each case the relation between the operations and the generic effects is as usual. The generic effects $deref : L \longrightarrow T(Val)$ and $assign : L \times Val \longrightarrow T1$ routinely extend those for global state pointwise, and the generic effect corresponding to the operator *block* is:

$$ref : Val \longrightarrow TL$$

where:

$$(ref)_n(v, \sigma) = ((\sigma, v), 1) \epsilon S(n + 1) \times (n + 1)$$

The computational idea is that ref creates a new location initialised to its argument value; this generic effect is also how ML treats the creation of new references.

Our examples suggest a general way to extend the λ_c-calculus. Given a generic effect $gen : P \longrightarrow TA$, one adds new basic type symbols **Par** and **Arg** and a new term-forming construct:

$$\frac{\mathbf{M} : \mathbf{Par}}{\mathbf{gen}\,\mathbf{M} : \mathbf{Arg}}$$

For the semantics one models the new basic type symbols by the corresponding tensors and the new terms by the evident composition in the Kleisli category with the generic effect. Pleasingly, one can then give the general equivalence between generic effects and operations at the level of the λ_c-calculus by the following equivalences:

$$\mathbf{gen}\,\mathbf{M} \ \equiv\ \mathbf{op}(\mathbf{M}, (\mathbf{x}\!:\!\mathbf{Arg})\,\mathbf{x})$$

$$\mathbf{op}(\mathbf{M}, (\mathbf{x}\!:\!\mathbf{Arg})\,\mathbf{M}) \ \equiv\ \mathbf{let}\ \mathbf{x} = (\mathbf{gen}\,\mathbf{M})\ \mathbf{in}\ \mathbf{N}$$

7 Algebraic operations and the category of algebras

Finally, in this section, we characterise the notion of algebraic operation in terms of the V-category of algebras T-Alg. The co-Kleisli category of the comonad on T-Alg induced by the monad T is used to model call-by-name languages with effects [11], so this formulation gives us an indication of how to generalise our analysis to call-by-name computation or perhaps to some combination of call-by-value and call-by-name, cf [11].

If C is V-closed and has equalisers, generalising Lawvere, the results of the previous section can equally be formulated as equivalences between algebraic operations and operations:

$$\alpha_{(A,a)} : U(A,a)^v \longrightarrow U(A,a)^w$$

natural in (A,a), where $U : T\text{-}Alg \longrightarrow C$ is the C-enriched forgetful functor: equalisers are needed in C in order to give an enrichment of T-Alg in C. We prove the result by use of our C-enriched version of the Yoneda lemma again, together with the observation that the canonical C-enriched functor $I : C_T \longrightarrow T\text{-}Alg$ is fully faithful. Formally, the result is

Theorem 4. *If C is V-closed and has equalisers, the C-enriched Yoneda embedding induces a bijection between maps $\mathbf{w} \longrightarrow \mathbf{v}$ in C_T and C-enriched natural transformations:*

$$\alpha : (U-)^v \longrightarrow (U-)^w.$$

Combining this with Theorem 2, we have

Corollary 2. *If C is closed and has equalisers, to give an algebraic operation:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

is equivalent to giving a C-enriched natural transformation:

$$\alpha : (U-)^v \longrightarrow (U-)^w.$$

One can also give a parametrised version of this result if C is neither closed nor complete along the lines for C_T as in the previous section. It yields

Theorem 5. *To give an algebraic operation:*

$$\alpha_x : (Tx)^v \longrightarrow (Tx)^w$$

is equivalent to giving an $Ob(T\text{-}Alg)$-indexed family of maps:

$$\alpha_{(A,a)} : U(A,a)^v \longrightarrow U(A,a)^w$$

such that commutativity of:

$$\begin{tikzcd}
C(x \otimes A, B) \arrow[r, "C(x \otimes a, B)"] \arrow[d, "T"'] & C(x \otimes TA, B) \\
C(T(x \otimes A), TB) \arrow[r, "C(st, B)"'] & C(x \otimes TA, TB) \arrow[u, "C(x \otimes TA, b)"']
\end{tikzcd}$$

implies commutativity of:

$$\begin{tikzcd}
C(x \otimes A, B) \arrow[r, "(-)^w"] \arrow[d, "(-)^v"'] & C((x \otimes A)^w, B^w) \arrow[r] & C(x \otimes A^w, B^w) \arrow[d, "C(x \otimes \alpha_{(A,a)}, B^w)"] \\
C((x \otimes A)^v, B^v) \arrow[r] & C(x \otimes A^v, B^v) \arrow[r, "C(x \otimes A^v, \alpha_{(B,b)})"'] & C(x \otimes A^v, B^w)
\end{tikzcd}$$

8 Conclusions and Further Work

For some final comments, we note that attention has not been paid in the literature to the parametrised naturality condition on the notion of algebraic operation that we have used heavily here. And none of the main results of [20] used it, although they did require naturality in C_T. So it is natural to ask why that is the case.

For the latter point, in [20], we addressed ourselves almost exclusively to closed terms, and that meant that parametrised naturality of algebraic operations was not emphasised as we did not need a parameter for our main results. Regarding why parametrised naturality does not seem to have been addressed in the past, observe that for $C = Set$, every monad has a unique strength, so parametrised naturality of α is equivalent to ordinary naturality of α. More generally, if the functor $C(1, -) : C \longrightarrow Set$ is faithful, i.e., if 1 is a generator in C, then parametrised naturality is again equivalent to ordinary naturality of α. That is true for categories such as *Poset* and that of ω-cpos, which have been the leading examples of categories studied in this regard. The reason we have a distinction is because we have not assumed that 1 is a generator, allowing us to include examples such as the effective topos.

The goal of this paper was to model *constructive* operations that arise in describing computational effects. In future, we hope to model relevant *deconstructive* operations such as one for handling exceptions. But the approach of this paper seems unlikely to extend directly. Our impression of continuations is that they are more naturally seen as a *logical* construct, following the Curry-Howard correspondence, than one amenable to our algebraic treatment, cf [5]. They should therefore be treated separately, but the question of their integration with the algebraic case remains.

We should also like to extend and integrate this work with work addressing other aspects of giving a unified account of computational effects. We note here Paul Levy's work [11], which can be used to give accounts of both call-by-value and call-by-name in the same setting, and ongoing work on modularity [7].

References

1. S. O. Anderson and A. J. Power, A Representable Approach to Finite Nondeterminism, *Theoret. Comput. Sci.*, Vol. 177, No. 1, pp. 3–25, 1997.
2. N. Benton, J. Hughes, and E. Moggi, *Monads and Effects*, APPSEM '00 Summer School, 2000.
3. B. Day, On Closed Categories of Functors, in *Reports of the Midwest Category Seminar IV*, LNM, Vol. 137, pp. 1–38, Berlin: Springer-Verlag, 1970.
4. E. Dubuc, *Kan Extensions in Enriched Category Theory*, LNM, Vol. 145, Berlin: Springer-Verlag, 1970.
5. C. Fuhrmann and H. Thielecke, On the Call-by-Value CPS Transform and its Semantics, (submitted).
6. R. Heckmann, Probabilistic Domains, in *Proc. CAAP '94*, LNCS, Vol. 136, pp. 21-56, Berlin: Springer-Verlag, 1994.
7. M. Hyland, G. D. Plotkin, and A. J. Power, Combining Computational Effects: Commutativity and Sum, in *Foundations of Information Technology in the Era of Network and Mobile Computing* (eds. Richardo Baeza-Yates, Ugo Montanari, and Nicola Santoro), IFIP, Vol. 223, Kluwer, 2002.
8. C. Jones, *Probabilistic Non-Determinism*, Ph.D. Thesis, University of Edinburgh, Report ECS-LFCS-90-105, 1990.
9. C. Jones and G. D. Plotkin, A Probabilistic Powerdomain of Evaluations, in *Proc. 4th LICS*, Asilomar, pp. 186–195, Washington: IEEE Press, 1989.
10. G. M. Kelly, *Basic Concepts of Enriched Category Theory*, Cambridge: Cambridge University Press, 1982.
11. P. B. Levy, Call-by-Push-Value: A Subsuming Paradigm, in *Proc. TLCA '99* (ed. J.-Y. Girard), LNCS, Vol. 1581, pp. 228-242, Berlin: Springer-Verlag, 1999.
12. S. Mac Lane, *Categories for the Working Mathematician*, Berlin: Springer-Verlag, 1971.
13. E. Moggi, Computational Lambda-Calculus and Monads, in *Proc. LICS '89*, pp. 14–23, Washington: IEEE Press, 1989.
14. E. Moggi, *An Abstract View of Programming Languages*, University of Edinburgh, Report ECS-LFCS-90-113, 1989.

15. E. Moggi, Notions of computation and monads, in *Inf. and Comp.* Vol. 93, No. 1, pp. 55–92, 1991.
16. P. W. O'Hearn and R. D. Tennent, *Algol-like Languages*, Progress in Theoretical Computer Science, Boston: Birkhauser, 1997.
17. R. Milner, M. Tofte, R. Harper and D. MacQueen *The Definition of Standard ML—Revised*, Cambridge: MIT Press, 1997.
18. G. D. Plotkin, A Powerdomain Construction, in *SIAM J. Comput.* Vol. 5, No. 3, pp. 452–487, 1976.
19. G. D. Plotkin, *Domains*, http://www.dcs.ed.ac.uk/home/gdp/, 1983.
20. G. D. Plotkin and A. J. Power, Adequacy for Algebraic Effects, in *Proc. FOSSACS '01* (eds. F. Honsell and M. Miculan), LNCS, Vol. 2030, pp. 1–24, Berlin: Springer-Verlag, 2001.
21. G. D. Plotkin and A. J. Power, Semantics for Algebraic Operations (extended abstract), in *Proc. MFPS XVII* (eds. S. Brookes and M. Mislove), ENTCS, Vol. 45, Amsterdam: Elsevier, 2001.
22. G. D. Plotkin and A. J. Power, Notions of Computation Determine Monads, in *Proc. FOSSACS '02* (eds. M. Nielsen and U. Engberg), LNCS, Vol. 2303, pp. 342–356, Berlin: Springer-Verlag, 2002.
23. A. J. Power, Enriched Lawvere Theories, in *Theory and Applications of Categories*, pp. 83–93, 2000.
24. A. J. Power, Models for the Computational Lambda Calculus, in *Proc. MFCSIT 2000* (eds. T. Hurley, M. Mac an Airchinnigh, M. Schellekens, and A. K. Seda), ENTCS, Vol. 40, Amsterdam: Elsevier, 2001.

A The computational $\boldsymbol{\lambda}$-calculus

In this appendix, we describe the computational λ-calculus, or λ_c-calculus and recall Moggi's notion of λ_c-model [13, 15]. There are several equivalent formulations of the λ_c-calculus. We shall not use the original formulation but a version that is equivalent except for the treatment of basic types and constants. This version of the λ_c-calculus has types given by:

$$\sigma ::= \mathbf{B} \mid \sigma \times \sigma \mid \mathbf{1} \mid \sigma \Rightarrow \sigma$$

where $\mathbf{B}$ ranges over a given set of base types such as **int**, and $\sigma \Rightarrow \tau$, rather than $\sigma \rightarrow \tau$, denotes the exponential. We do not assert the existence of a type construction $\mathbf{T}\sigma$: this formulation is equivalent to the original one because $\mathbf{T}\sigma$ may be defined to be $\mathbf{1} \Rightarrow \sigma$.

The terms of the λ_c-calculus are given by:

$$\mathbf{M} ::= \mathbf{x} \mid \mathbf{b} \mid \mathbf{MM} \mid \lambda\mathbf{x} : \sigma.\mathbf{M} \mid * \mid (\mathbf{M}, \mathbf{M}) \mid \pi_{\mathrm{i}}(\mathbf{M})$$

where $\mathbf{x}$ is a variable; $\mathbf{b}$ ranges over base terms of arbitrary given types, such as $\mathbf{0}$ and **succ**, of respective types **int** and $\mathbf{int} \Rightarrow \mathbf{int}$; and with π_{i} existing for $\mathrm{i} = 1$ or 2. There are evident typing rules for judgements $\Gamma \vdash M : \sigma$, that the term M has type σ in the context Γ (and contexts have the form $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$); in particular $*$ is of type $\mathbf{1}$. This differs from the original formulation of the calculus in that we do not explicitly have a **let** constructor or constructions $[\mathbf{M}]$

or $\mu(\mathbf{M})$. The two formulations are equivalent as we may consider $\mathbf{let\ x} = \mathbf{M} : \sigma$ $\mathbf{in\ N}$ as syntactic sugar for $(\lambda\mathbf{x} : \sigma.\mathbf{N})\mathbf{M}$ (and may then elide the σ), and $[\mathbf{M}]$ as syntactic sugar for $\lambda\mathbf{x} : \mathbf{1}.\mathbf{M}$ where $\mathbf{x}$ is of type $\mathbf{1}$ and does not occur freely in $\mathbf{M}$, and $\mu(\mathbf{M})$ as syntactic sugar for $\mathbf{M}*$.

We use this formulation as it has less data, allowing for easier proofs. Moreover, it is more directly a fragment of a typical call-by-value language: the above type constructors and, with the possible exception of the π_i, term constructors often appear explicitly in call-by-value languages, whereas $\mathbf{T}$-types, μ, and $[-]$ typically do not.

The λ_c-calculus has two predicates, existence, denoted by $\downarrow$, and equivalence, denoted by $\equiv$. The $\downarrow$ rules may be expressed as saying $* \downarrow$, $\mathbf{x} \downarrow$, $\lambda\mathbf{x} : \sigma.\mathbf{M} \downarrow$ for all $\mathbf{M}$, if $\mathbf{M} \downarrow$ then $\pi_\mathbf{i}(\mathbf{M}) \downarrow$, and similarly for $(\mathbf{M}, \mathbf{N})$, typically accompanied by rules for some of the base terms, e.g., $\mathbf{0} \downarrow$, $\mathbf{succ} \downarrow$ and $\mathbf{succ}\,\mathbf{M} \downarrow$ when $\mathbf{M} \downarrow$. A *value* is a term $\mathbf{M}$ such that $\mathbf{M} \downarrow$. There are two classes of rules for $\equiv$. The first class say that $\equiv$ is a congruence, with variables allowed to range over values. And the second class are rules for the basic types and for unit, product and functional types. It follows from the rules for both predicates that types together with equivalence classes of terms form a category, with a subcategory determined by values.

It is straightforward, using the original formulation of the λ_c-calculus in [13], to spell out the inference rules required to make this formulation agree with the original one: one just bears in mind that the models are the same, and uses syntactic sugar as detailed above.

Evaluation contexts for the λ_c-calculus are defined by the following inductive clauses: $[-]_\sigma$ is an evaluation context; $\mathbf{EM}$, $\mathbf{VE}$, $(\mathbf{E}, \mathbf{M})$ and $\pi_i(\mathbf{E})$ are evaluation contexts for any evaluation context $\mathbf{E}$, term $\mathbf{M}$ and value $\mathbf{V}$; and there are also suitable clauses for the base terms, such as that $\mathbf{succ}\,\mathbf{E}$ is an evaluation context if $\mathbf{E}$ is. One can type evaluation contexts by adding the rule that $[-]_\sigma : \sigma$. The computational thought behind evaluation contexts is that in a program of the form $\mathbf{E}[\mathbf{M}]$ the first computational step arises within $\mathbf{M}$.

For category theoretic models, the key feature is that there are two entities, terms and values. So the most direct way to model the language as we have formulated it is in terms of a pair of categories V and T, together with an identity-on-objects inclusion functor $J : V \longrightarrow T$. This train of thought leads directly to the notion of closed *Freyd*-category [24]. But the first sound and complete class of models for the λ_c-calculus was given by Moggi in [13, 15].

For Moggi, a λ_c-model consists of a category C with finite products, together with a strong monad T on C, such that T has Kleisli exponentials, i.e., for each object x of C, the functor $J(- \times x) : C \longrightarrow C_T$ has a right adjoint, where C_T is the Kleisli category for C and $J : C \longrightarrow C_T$ is the canonical functor. A term of type σ in context $\mathbf{\Gamma}$ is modelled by a map in the Kleisli category for T, i.e., by a map in C from $[\![\mathbf{\Gamma}]\!]$ to $T[\![\sigma]\!]$, where $[\![-]\!]$ denotes the semantic construct (and for $\mathbf{\Gamma} = \mathbf{x_1} : \sigma_\mathbf{1}, \ldots, \mathbf{x_n} : \sigma_\mathbf{n}$, $[\![\mathbf{\Gamma}]\!] = [\![\sigma_\mathbf{1}]\!], \times \ldots \times [\![\sigma_\mathbf{n}]\!]$). In terms of Freyd categories, C and C_T evidently correspond to V and T. The extension of these ideas to the more general situation where C is V-enriched is straightforward. Returning to evaluation contexts, to each such context $\mathbf{\Gamma} \vdash \mathbf{E} : \tau$ where the "hole" in $\mathbf{E}$ is $[-]_\sigma$ one can assign a morphism $[\![\mathbf{E}]\!] : [\![\mathbf{\Gamma}]\!] \times [\![\sigma]\!] \to [\![\tau]\!]$. One then has that $[\![\mathbf{E}[\mathbf{M}]]\!] = \mathbf{E} \circ (\mathrm{id}_{[\![\mathbf{\Gamma}]\!]}, [\![\mathbf{M}]\!])$.

View publication stats