Notions of Computation Determine Monads

Gordon Plotkin and John Power *

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

Abstract. We model notions of computation using algebraic operations and equations. We show that these generate several of the monads of primary interest that have been used to model computational effects, with the striking omission of the continuations monad. We focus on semantics for global and local state, showing that taking operations and equations as primitive yields a mathematical relationship that reflects their computational relationship.

1 Introduction

Eugenio Moggi, in [14, 16], introduced the idea of giving a unified category theoretic semantics for what he called notions of computation, but which we call computational effects. He modelled each computational effect in the Kleisli category for an appropriate strong monad T on a base category C with finite products. The perspective of this paper is that computational effects *determine* monads but are not identified with monads. We regard a computational effect as being realised by families of operations, with a monad being generated by their equational theory.

Examples of computational effects are: exceptions, interactive input/output, nondeterminism, probabilistic nondeterminism, side-effects and continuations. Moggi's unified approach to modelling them has proved useful, for example in functional programming [2], but there has not been a precise mathematical basis on which to compare and contrast the various effects.

For instance, continuations are computationally of a different character to other computational effects, being an inherently non-local phenomenon. Again, computationally, the introduction of global state is a first step towards the introduction of local state. So we seek a mathematical description of features of the various monads that reflects the comparisons between the corresponding computational phenomena.

An immediate observation is that the monad for continuations $R^{(R^-)}$ does not have a rank (see [9] for a definition), while the monads for all the other above-mentioned computational effects do. There is a theorem to the effect that monads are derivable from algebraic operations and equations if and only if they have bounded rank [10]. So consideration of operations and equations might provide a way to describe features of the computational effects. The equations

* This work is supported by EPSRC grant GR/M56333 and a British Council grant.

may also prove useful when manipulating programs, e.g., for optimisation. We regard it as positive that there are no algebraic operations and equations in the sense of the theorem yielding the continuations monad, taking that as reflecting its differing computational nature.

There are computationally natural families of operations associated with several of the above monads: one has 'raise' operations for exceptions; one has 'read' and 'write' operations associated with interactive input/output; one has a nondeterministic binary 'choice' operation when modelling nondeterminism; one has a 'random choice' operation for probabilistic nondeterminism; and one has 'lookup' and 'update' operations when modelling global state. An analysis of several of these families of operations appears in [20]: they are regarded as *algebraic families of operations* associated with an already given monad, and are characterised in terms of generic effects: e.g., to give a generic effect $e : \underline{n} \longrightarrow T\underline{m}$ is equivalent to giving m n-ary algebraic families of operations, where m and n need not be finite ($\underline{m}$ is the m-fold coproduct of 1 in C). Crucially when a monad is given by algebraic operations and equations in the sense of [10], the algebraic families of operations associated with it are given by the derived operations.

In programming languages, effects are obtained using syntactic constructs corresponding to the operations or the generic effects. Some examples with finitary operations were considered in [19], which considered suitable extensions of call-by-value PCF, itself an extension of Moggi's computational λ-calculus, and gave a unified treatment of operational semantics. Although infinitary operations, as we study here, do not appear directly in programming languages, the corresponding generic effects do, and we sketch appropriate extensions of call-by-value PCF incorporating them.

The present paper investigates whether the monads are given by computationally natural equations for the above naturally occurring families of operations. This is already well known to be so in the case of some of the above examples, notably those associated with nondeterminism; other cases, such as those of exceptions and interactive input/output, are easy; but global and local state required considerable thought. So most of the technical detail is devoted to the situation for state. At this point, we should like to thank Eugenio Moggi for suggesting to us that the monad for global state may be derived from (possibly infinitary) computationally natural operations and equations, and we should like to thank Peter O'Hearn for showing us a monad for local state suited to call-by-value. Two new features emerge in characterising local state. First, the arities must be allowed to be not just sets but presheafs; and second, the block operation, in contrast to lookup and update, is linear, using symmetric monoidal closed structure rather than cartesian closed structure.

The paper is organised as follows. In Section 2, we give a general explanation of the notions of signature, operations, and equations, and we present the straightforward examples discussed above. In Section 3, we give a careful explanation of how the monad $(S \otimes -)^S$ for global state is generated by operations for lookup and update subject to computationally natural equations. And in Section 4, we extend the definitions of Section 3 to see how the addition of block

subject to natural additional equations generates the monad for local state. The central point here is that this gives precise mathematics that reflects the computational relationship between global and local state. For future work, having made some progress in understanding individual computational effects, one can consider their combinations; we address that issue in [6].

2 Operations and Equations

Given a finitary signature Σ in the usual universal algebraic sense, one can speak of a Σ-algebra in any category C with finite products: it consists of an object A of C together with, for each σ in Σ, a map

$$a_\sigma : A^{ar(\sigma)} \longrightarrow A$$

in C, where $ar(\sigma)$ is the arity of σ. One can speak of Σ-equations and their satisfaction in a Σ-algebra, obtaining the notion of a (Σ, E)-algebra in C. This, with the evident definition of homomorphism of algebras, generates a category (Σ, E)-Alg with a forgetful functor

$$U : (\Sigma, E)\text{-}Alg \longrightarrow C$$

which, if C is locally presentable, has a left adjoint F, inducing a monad $T = UF$ on C. The category (Σ, E)-Alg is isomorphic to the category T-Alg of algebras for the monad T.

This is a considerably simplified version of the work in [10], but the above version is sufficient here. One illuminating view is in terms of models for a Lawvere theory in a category other than Set, cf [20, 21]. There is nothing special about the finitariness of Σ: everything holds equally for infinitary operations, providing C has correspondingly infinitary products, as all our examples do. It is, moreover, routine to verify that the induced monad T always has a natural associated strength, induced by the universal property of products. Leading examples of interesting categories C are Set, $Poset$, ω-Cpo, presheaf categories $[W, Set]$, and functor categories of the form $[W, \omega\text{-}Cpo]$ for a small category of worlds W, cf [17].

Example 1. Exceptions The monad $- + E$ for exceptions on Set is induced by E nullary operations, with no equations. These operations model the raising of exceptions, but do not model a 'handle' operation. This distinction is consistent with the fact that raising exceptions is algebraic [20], while (significantly!) handling exceptions is not. In this paper, we only consider algebraic operations; non-algebraic operations such as handle are of a different character, which remains to be understood.

Example 2. Interactive Input/Output The monad $TX = \mu Y.(O \times Y + Y^I + X)$ for interactive I/O on Set is induced by operations $read : X^I \longrightarrow X$ (which is infinitary) and $write : X \longrightarrow X^O$, with no equations [15]. The corresponding generic effects are $e_r : 1 \longrightarrow TI$ and $e_w : O \longrightarrow T1$; the corresponding extension of call-by-value PCF would have datatypes In and Out (they could both be $Char$, a character type) and programs $\mathbf{read} : In$ and $\mathbf{write}\ M : 1$ for $M : Out$.

Example 3. Nondeterminism Let C be the category of ω-cpo's. Then the category of algebras for the convex power-domain [5, 18, 1] is the category of semilattices in C (i.e., structures with an associative, commutative, idempotent binary operation), equipped with a least element $\perp$. Similar facts are true of the upper and lower power-domains, except that each requires an additional equational axiom in the setting of [10].

Example 4. Probabilistic Nondeterminism The probabilistic power-domain [7, 8, 4] can be treated algebraically in several equivalent ways. One is via a random choice operator $x +_r y$ meaning "do x with probability r, or y with probability $1-r$." Taken together with a bottom element this has an axiomatisation over the category of ω-cpo's that fits within the framework of [10]. The equivalent generic effect is $(e_p)_r : 1 \to T(2)$. In programming languages considered in the literature one sees both explicit choice operators and a random 'die' $\mathbf{rand}_r : \mathit{Bool}$.

3 Global State

In this section, we show how the side-effects monad, which is used to model global state, is generated by operations for lookup and update subject to computationally natural equations.

Let L be a finite set, to be regarded as a set of locations, and let V be a countable set, to be regarded as the set of values. For instance, V may be taken to be the set of natural numbers. One defines the set S of states of a language to be the set V^L of functions from locations to values. So S is a countable set. The restriction to finite L is deliberate and is needed for the proofs of our results; rather than use an infinite set to deal with the availability of an unbounded number of locations, we prefer to use a presheaf semantics as in Section 4. Observe that one includes the case of any countable set S by putting $L = 1$ and $V = S$; we shall need L explicit in order to analyse local state.

Now assume we have a category C with countable products and coproducts. Consider the monad T on C given by $(S \otimes -)^S$, where A^X means the product of X copies of the object A of C, and $X \otimes A$ means the coproduct of X copies of A. A map in the Kleisli category from A to B is equivalent to giving a map in C from $S \otimes A$ to $S \otimes B$, thus allowing a change of state.

We seek to express the category $(S \otimes -)^S$-Alg as the category of (Σ, E)-algebras, for computationally natural Σ and E, in the category C. In order to give this result, we define a category $GS(C)$, which, by its description, is the category of (Σ, E)-algebras in C for evident (Σ, E), so that the evident forgetful functor $U : GS(C) \longrightarrow C$ has a left adjoint given by $(S \otimes -)^S$. It follows that (Σ, E)-Alg is isomorphic to $(S \otimes -)^S$-Alg.

Our operations will consist of a lookup operation $l : A^V \longrightarrow A^L$ and an update operation $u : A \longrightarrow A^{L \times V}$; these are equivalent to families of operations as previously considered. Given a V-indexed family of elements of A, the infinitary lookup operation takes a location loc, finds out what its value is in the current state of the computation, and computes the element of A determined

by that value. Given an element of A together with a location loc and a value v, the update operation updates the state by insisting that loc take value v, and then allowing the computation to run. The corresponding generic effects are $e_l : \underline{L} \longrightarrow T\underline{V}$ and $e_u : \underline{L} \times \underline{V} \longrightarrow T1$, where $\underline{L} = L \otimes 1$ and $\underline{V} = V \otimes 1$. In a corresponding extension of call-by-value PCF, one would have datatypes Loc and Val (the latter might be Nat) and program constructions $!M : Val$ for $M : Loc$ and $(M := N) : 1$ for $M : Loc$ and $N : Val$.

We take care here to give the result for a category C with axiomatic structure rather than just for Set, as we expect the more general result to be required for modelling the combination of side effects with other computational effects. Our results also extend routinely to the situation where C is a V-category (see [9]) and the set of values is replaced by an object of V. It remains to be seen how best to handle complex situations such as storable procedures [11] or linked lists.

Definition 1. *Given a category C with countable products, a finite set L, and a countable set V, we define the category $GS(C)$ as follows: an object consists of*

- *an object A of C*
- *a lookup map $l : A^V \longrightarrow A^L$, and*
- *an update map $u : A \longrightarrow A^{L \times V}$*

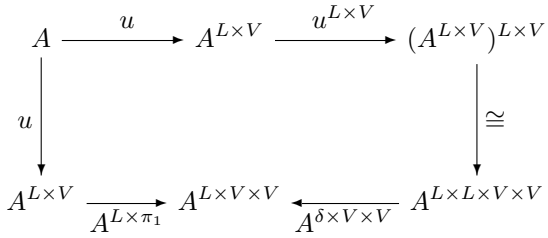
subject to commutativity of two classes of diagrams. First, we have four interaction diagrams as follows:

$$\begin{array}{ccccc}
A & \xrightarrow{u} & A^{L\times V} & \xrightarrow{\cong} & (A^V)^L \\
{\scriptstyle A^t}\downarrow & & & & \downarrow{\scriptstyle l^L} \\
A^L & \xleftarrow[A^\delta]{} & A^{L\times L} & \xleftarrow[\cong]{} & (A^L)^L
\end{array}$$

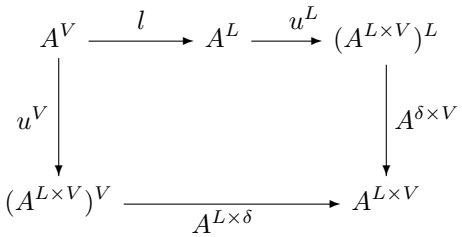
where $\delta : L \longrightarrow L \times L$ and $t : L \longrightarrow 1$ are the diagonal and terminal maps, and the lower unlabelled isomorphism matches the outer L of $(A^L)^L$ with the first L of $A^{L\times L}$,

$$\begin{array}{ccccc}
(A^V)^V & \xrightarrow{l^V} & (A^L)^V & \xrightarrow{\cong} & (A^V)^L \\
\downarrow{\scriptstyle \cong} & & & & \downarrow{\scriptstyle l^L} \\
A^{V\times V} & & & & (A^L)^L \\
{\scriptstyle A^\delta}\downarrow & & & & \downarrow{\scriptstyle \cong} \\
A^V & \xrightarrow[l]{} & A^L & \xleftarrow[A^\delta]{} & A^{L\times L}
\end{array}$$

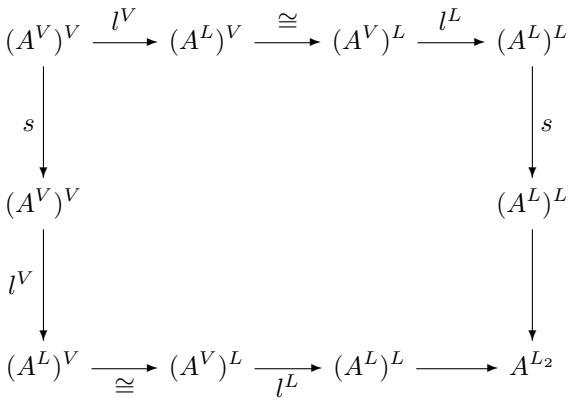
where the unlabelled isomorphisms match the outer V of $(A^V)^V$ with the first V of $A^{V\times V}$ and similarly for L, cf [9],



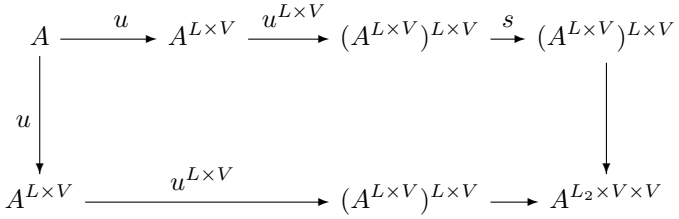
where the unlabelled isomorphism matches the outside L with the first L and similarly for V, and



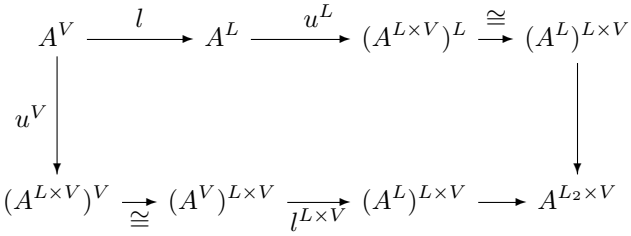
suppressing two isomorphisms. We also have three commutation diagrams as follows:



where s signifies 'swap' maps and L_2 denotes the set of ordered pairs of distinct elements of L, with the unlabelled maps both given by the same canonical map,



where s again signifies a swap map and with the unlabelled maps again given by the same canonical map, and



where, again, the unlabelled maps are given by the same canonical map. The rest of the structure of $GS(C)$ as a category is evident: for instance, a map from (A, u, l) to (A', u', l') is a map $f : A \longrightarrow A'$ in C subject to commutativity of f with l and l' and commutativity of f with u and u'.

The above constitutes our formal definition of the category of algebras. For any category C with countable products, one can routinely give an equational language for which equations between infinitary terms of the language correspond to commutative diagrams in the category. One has a V-ary function symbol l_{loc} for each loc in L and a unary function symbol $u_{loc,v}$ for each loc in L and v in V. The seven commutative diagrams in the definition of $GS(C)$ can be expressed equationally as the following seven axiom schema involving infinitary expressions respectively:

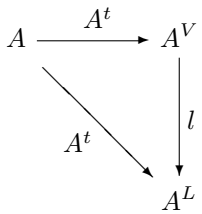
1. $l_{loc}(u_{loc,v}(x))_v = x$
2. $l_{loc}(l_{loc}(t_{vv'})_v)_{v'} = l_{loc}(t_{vv})_v$
3. $u_{loc,v}(u_{loc,v'}(x)) = u_{loc,v'}(x)$
4. $u_{loc,v}(l_{loc}(t_{v'})_{v'}) = u_{loc,v}(t_v)$
5. $l_{loc}(l_{loc'}(t_{vv'})_{v'})_v = l_{loc'}(l_{loc}(t_{vv'})_v)_{v'}$ where $loc \neq loc'$
6. $u_{loc,v}(u_{loc',v'}(x)) = u_{loc',v'}(u_{loc,v}(x))$ where $loc \neq loc'$
7. $u_{loc,v}(l_{loc'}(t_{v'})_{v'}) = l_{loc'}(u_{loc,v}(t_{v'}))_{v'}$ where $loc \neq loc'$.

It can be shown that this axiom system is *Hilbert-Post complete*, meaning that it has no equationally consistent extensions; thus we have all the equations for global state. The schema induce program assertions. Here are those corresponding to the third and sixth:

3*. $(l := x; \mathbf{let}\ y\ \mathbf{be}\ !l\ \mathbf{in}\ M) = (l := x; M[x/y])$
6*. $(l \neq m) \supset (l := x; m := y) = (m := y; l := x)$

where $x, y : Val$, $l, m : Loc$ and $M; N$ abbreviates $(\lambda x : 1.N)(M)$ with x fresh.

Proposition 1. *For any object (A, l, u) of $GS(C)$, the diagram*



commutes.

Proof. Use two applications of the first axiom and one application of the second axiom.

In the equational logic, this proposition is: $l_{loc}(x)_v = x$.

We henceforth assume that C has both countable products and countable coproducts.

Proposition 2. *For any object X of C, the object $(S \otimes X)^S$ together with the maps*

$$u : (S \otimes X)^S \longrightarrow ((S \otimes X)^S)^{L \times V}$$

determined by composition with the function from $L \times V \times V^L$ to V^L that, given (loc, v, σ), "updates" $\sigma : L \longrightarrow V$ by replacing its value at loc by v and

$$l : ((S \otimes X)^S)^V \longrightarrow ((S \otimes X)^S)^L$$

determined by composition with the function from $L \times V^L$ to $V \times V^L$ that, given (loc, σ), "looks up" loc in $\sigma : L \longrightarrow V$ to determine its value, and is given by the projection to V^L, satisfy the commutative diagrams required to give an object of $GS(C)$.

The definitions of u and l in the proposition correspond to the equations

$$u(loc, v, x)(\sigma) = x(\sigma[v/loc])$$

and

$$l(loc, (x_v)_v)(\sigma) = x_{\sigma(loc)}(\sigma).$$

Theorem 1. *The forgetful functor $U : GS(C) \longrightarrow C$ exhibits the category $GS(C)$ as monadic over C, with monad $(S \otimes -)^S$.*

Proof. We first show that the left adjoint to U is the functor $(S \otimes -)^S$, with algebra structure on $(S \otimes X)^S$ given by the proposition, and with the unit of the adjunction given by the canonical map $\eta_X : X \longrightarrow (S \otimes X)^S$. Given an algebra (A, l, u) and a natural number n, let

$$u_n : (L \times V)^n \otimes A \longrightarrow A$$

denote the canonical map induced by n applications of u, and let

$$l_n : A^{V^n} \longrightarrow A^{L^n}$$

denote the canonical map induced by n applications of l.

Given an arbitrary map $f : X \longrightarrow A$, and recalling that $S = V^L$, define $\overline{f} : (S \otimes X)^S \longrightarrow A$ to be the composite of $(S \otimes f)^S : (S \otimes X)^S \longrightarrow (S \otimes A)^S$ with

$$(V^L \otimes A)^{V^L} \twoheadrightarrow ((L \times V)^L \otimes A)^{V^L} \xrightarrow{u_L^{V^L}} A^{V^L} \xrightarrow{l_L} A^{L^L} \twoheadrightarrow A$$

where the unlabelled maps are the evident structural maps. We need to prove four commutativities: one showing that $\overline{f}$ composed with η_X is f, two to show that $\overline{f}$ is an algebra map, and a final one to show that, given any algebra map $g : (S \otimes X)^S \longrightarrow A$, the map $\overline{g\eta}$ equals g.

For the unit axiom, first observe that the commutation axioms generalise to allow l and u to be replaced by l_n and u_m for arbitrary natural numbers n and m. The unit axiom follows by induction on the size of L using these generalised versions of the first two commutation axioms and the first interaction axiom.

One can see the proof of commutativity of $\overline{f}$ with u by first considering the case where L has precisely one element, when the proof is easy using the third and fourth interaction axioms, and Proposition 1. The proof for arbitrary L is essentially the same, but also requires the generalised commutation axioms.

Commutativity of $\overline{f}$ with l is straightforward: it requires the second interaction diagram together with generalised versions of the first commutation diagram. And the final commutativity follows from routine calculation, just using naturality. So $(S \otimes -)^S$ is indeed left adjoint to U.

Finally, it follows routinely from Beck's monadicity theorem that U is monadic.

4 Local State

We now consider local state in terms of operations and equations, extending those for global state in a principled fashion. In order to do that, we first discuss how to model local state. Following [17], as further studied in [12], we do not model local state in terms of a category with axiomatically given structure as we have done for global state, but rather restrict attention to a particular presheaf category $[I, Set]$ where I is the category of finite sets and injections. We hope that our results will generalise to functor categories $[I, C]$ where C has axiomatically given structure.

Note that I is equivalent to the category of natural numbers and monomorphisms; I does not have finite coproducts, and in particular, the sum of two natural numbers does not act as their binary coproduct. However, I does have an initial object, so by duality, I^{op} has a terminal object. The Yoneda embedding embeds I^{op} into the presheaf category $[I, Set]$, which is cartesian closed as a locally presentable category, allowing us to use the general theory of operations and equations of [10].

Finite products in $[I, Set]$ are given pointwise, and the closed structure, given functors $X, Y : I \longrightarrow Set$, is given by

$$(Y^X)n = [I, Set](X - \times I(n, -), Y-)$$

i.e., the set of natural transformations from $X - \times I(n, -)$ to Y. The terminal object of $[I, Set]$ is $I(0, -)$. There is a convenient additional 'convolution' symmetric monoidal closed structure on $[I, Set]$. The closed structure $[X, Y]$ corresponding to the convolution monoidal product is

$$[X, Y]n = [I, Set](X-, Y(n + -))$$

In particular, by the Yoneda lemma, $[I(m,-),Y]n = Y(n+m)$. Moreover, the functor $[X,-]$ has a canonical strength with respect to the cartesian closed structure of $[I,Set]$. We shall use a combination of both symmetric monoidal closed structures here.

We no longer model state by a set; instead, we index it according to the world in which it is defined. So, given a set of values V, state is modelled by the functor $S : I^{op} \longrightarrow Set$ given by $Sn = V^n$. Note that S is not an object of $[I,Set]$. Observe that the functor S is the composite of the inclusion $I^{op} \longrightarrow Set^{op}$ with $V^{(-)} : Set^{op} \longrightarrow Set$.

The monad for local state is

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

where $\int$ denotes a coend, which is a complicated form of colimit involving a universal dinatural map [9, 13]. This construction is a simplified version of one in Levy's thesis [12]; the idea is that in a state with n locations, a computation can create $m-n$ new locations and return a value (e.g., a function) that depends on them (and so one also needs to know S at all of m). In the case $V = 1$ it reduces to the monad for local names in [23]; it would be interesting to know the relationship with the monads for local state in [22]. The behaviour of T on injective maps $f : n \longrightarrow n'$ is as follows: decompose n' as the sum $n + n''$, note that $S(p + n'') = Sp \times Sn''$, and use covariance of X. So the map

$$(\int^{m\epsilon(n/I)} (Sm \times Xm))^{Sn} \times Sn \times Sn'' \longrightarrow \int^{m''\epsilon((n+n'')/I)} (Sm'' \times Xm'')$$

evaluates at Sn, then maps the m-th component of the first coend into the $(m + n'')$-th component of the second, using the above isomorphism for S and functoriality of X. The monad T routinely has strengths with respect to both symmetric monoidal closed structures.

We denote the inclusion of I into Set, which is $I(1,-)$, by the notation L, as it represents locations, and we overload notation by letting $V : I \longrightarrow Set$ denote the constant functor at V, representing values. As L is not a mere set but rather a set indexed by a world, we need more refined notions of signature, operations, and equations in order to allow L and V to be arities as we had for global state. Note that our definition of L_2 as in the previous section extends here, where L_2 may be seen as the functor from I to Set that sends a finite set to the set of ordered pairs of distinct elements.

Ideally, we should like to use either the cartesian closed structure or the convolution monoidal closed structure of $[I,Set]$ in order to present the monad T as generated by algebraic structure as in [10]. But the equations we need for l and u are those for global state, and they are inherently cartesian, using diagonals and projections. So, if we were to use only one of the structures, it would need to be the cartesian closed one. But our construction of a lifting, as is essential to our proof of the main theorem, requires the block map b to have domain $[L,A]$ given by the linear convolution closed structure (and this is computationally

natural as $[L, A](n) \cong A(n+1)$). So, we use a combination of the two kinds of structure here. Such a combination is not considered in [10], but that theory does apply in that, from our description, one can routinely induce unenriched algebraic structure, albeit complicated. However, it may ultimately be better to develop the theory of [10] to allow for a pair of enrichments interacting with each other, as here, then use that generalised theory in describing such phenomena as local state or as arise in modelling the π-calculus [3].

For this paper, we proceed by analogy with global state, by defining a category $LS([I, Set])$ which should be of the form (Σ, E)-Alg in a sense to be made precise in one of the ways outlined above. The relationship between our modelling of global and local state will be clear. Observe that, as $V : I \longrightarrow Set$ is a constant functor, we have

$$(A^V)- = [V, A]- = (A-)^V$$

We shall only use the notation A^V.

Definition 2. *We define the category $LS([I, Set])$ as follows: an object consists of*

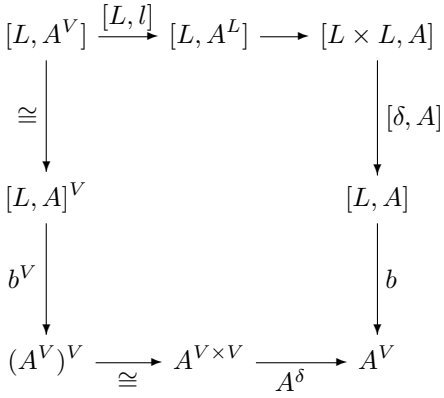
- *an object A of $[I, Set]$*
- *a lookup map $l : A^V \longrightarrow A^L$*
- *an update map $u : A \longrightarrow A^{L \times V}$*
- *a block map $b : [L, A] \longrightarrow A^V$*

subject to commutativity of six interaction diagrams and six commutativity diagrams. The interaction diagrams consist of the four interaction diagrams for global state, together with

$$\begin{array}{ccccccc}
[L, A] & \xrightarrow{[L, u]} & [L, A^{L \times V}] & \xrightarrow{\cong} & [L, A^L]^V & \longrightarrow & [L \times L, A]^V \\
\Big\downarrow{\scriptstyle b} & & & & & & \Big\downarrow{\scriptstyle [\delta, A]^V} \\
A^V & & \xrightarrow[(A^t)^V]{} & & (A^V)^V & \xleftarrow[b^V]{} & [L, A]^V
\end{array}$$

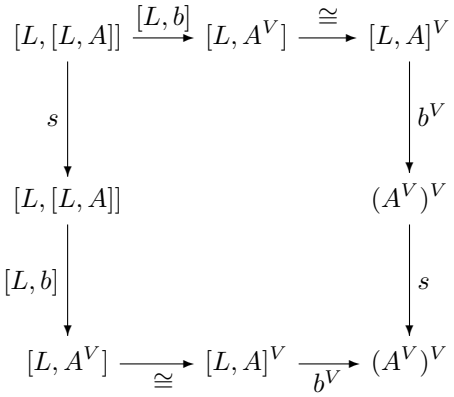
where the horizontal unlabelled map is given by a canonical distributivity law of $X \otimes -$ over product together with the fact that the unit for the tensor product is

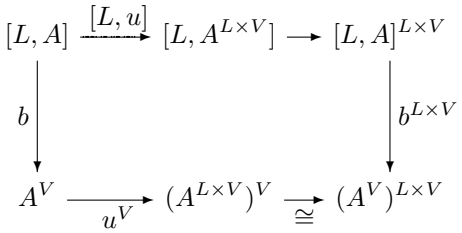
the terminal object, and



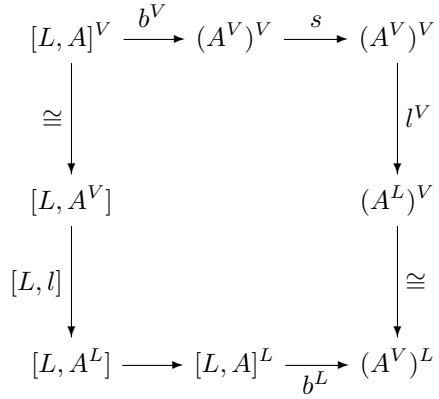
where the horizontal unlabelled map is determined as in the first diagram.

The commutation diagrams are those for global state together with





and



We do not have a formal syntactic analogue of these diagrams as equations, but we intend to produce one in further work. In outline, for the diagrams inherited from global state, we imagine interpreting the previous equations internally, as regards the *loc* indices. And for the new diagrams, we could imagine something along the lines of introducing operations b_v and expressing the above equations syntactically as

1. $b_v\langle u_{loc,v'}(t)\rangle_{loc} = b_{v'}\langle t\rangle_{loc}$
2. $b_v\langle l_{loc}(t_{v'})_{v'}\rangle_{loc} = b_v\langle t_v\rangle_{loc}$
3. $b_v\langle b_{v'}\langle t\rangle_{loc'}\rangle_{loc} = b_{v'}\langle b_v\langle t\rangle_{loc}\rangle_{loc'}$
4. $b_v\langle u_{loc',v'}(t)\rangle_{loc} = u_{loc',v'}(b_v\langle t\rangle_{loc})$
5. $b_v\langle l_{loc'}(t_{v'})_{v'}\rangle_{loc} = l_{loc'}(b_v\langle t_{v'}\rangle_{loc}\rangle_{v'}$

notationally differentiating linear abstraction and cartesian indexing (note that the terms t may contain occurrences of variables such as *loc*).

For an extension of call-by-value PCF we can add a block construct

$$\textbf{block new } l := x; M \textbf{ end} : \sigma$$

for $M : \sigma$. A construct corresponding to the generic effect $e_b : V \to TL$ (see below) appears in [22], viz **ref** $M : Loc$ for $M : Val$, creating a new reference. One can also add an equality test on locations. Here is a sample program assertion, corresponding to the fourth equation:

$$(\textbf{block new } l := x; m := y; M \textbf{ end}) = (m := y; \textbf{block new } l := x; M \textbf{ end})$$

In order to prove that the free algebra is given by the monad for local state, we need to put an algebra structure on TX for arbitrary X in $[I, Set]$. It is simplest to express this, using the theory of [20], in terms of generic effects. So we give maps $e_l : L \longrightarrow TV$, $e_u : L \times V \longrightarrow T1$ and $e_b : V \longrightarrow TL$ with the understanding that $l : (TX)^V \longrightarrow (TX)^L$ is defined using composition with e_l in the Kleisli category $Kl(T)$, and similarly for e_u and e_b. They are defined as follows:

$$(e_l)_n : n \longrightarrow [Sn, Sn \times V]$$

is defined by $(e_l)_n(p,\sigma) = (\sigma, \sigma(p))$

$$(e_u)_n : n \times V \longrightarrow [Sn, Sn]$$

is defined by $(e_u)_n(p, v, \sigma) = \sigma[v/p]$, and

$$(e_b)_n : V \longrightarrow (\int^{m\epsilon(n/I)} (Sm \times m))^{Sn}$$

is defined by $(e_b)_n(v,\sigma) = ((\sigma, v), 1)\epsilon S(n+1) \times (n+1)$.

Proposition 3. *For any object X of $[I, Set]$, the object TX together with the maps l, u and b as defined above, satisfy the commutative diagrams required to give an object of $LS([I, Set])$.*

Theorem 2. *The forgetful functor $U : LS([I, Set]) \longrightarrow [I, Set]$ exhibits the category $LS([I, Set])$ as monadic over $[I, Set]$ with monad T as above.*

Proof. The proof is essentially the same as that for Theorem 1. The key construction is that given a map$f : X \longrightarrow A$ where A is an algebra, f extends to an algebra map $\overline{f}$ given by the composite of $Tf : TX \longrightarrow TA$ with, on the n-th component,

- a structural map

$$(\int^{m\epsilon(n/I)} (V^m \times Am))^{V^n} \longrightarrow (\int^{m\epsilon(n/I)} (V^{m-n} \times (m \times V)^n \times Am))^{V^n}$$

- a map $(\int^{m\epsilon(n/I)} (V^{m-n} \times (m \times V)^n \times Am))^{V^n} \longrightarrow (\int^{m\epsilon(n/I)} (V^{m-n} \times Am))^{V^n}$ given by n applications of u_m
- a map $(\int^{m\epsilon(n/I)} (V^{m-n} \times Am))^{V^n} \longrightarrow (An)^{V^n}$ given on the $(i : n \to m)$-th component by a composite of applications of b_p as p varies from n to $m-1$
- a map $(An)^{V^n} \longrightarrow (An)^{n^n}$ given by n applications of l_n, and
- a structural map $(An)^{n^n} \longrightarrow An$.

The two new interaction axioms are used to prove that $\overline{f}$ respects b.

Unlike the case of global state there is a further natural sixth axiom for block, viz: $b_v\langle x\rangle_l = x$. This is *false* in our model, although the corresponding program assertion (**block new** $l := x; M$ **end**) $= M$ (l not free in M) does hold operationally. Relevant logical relation techniques for finding a monadic semantics are given in [22, 23]; we conjecture that, with the additional axiom, the axioms for local state will prove Hilbert-Post complete.

References

1. S. O. Anderson and A. J. Power, A Representable Approach to Finite Nondeterminism, in *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. M. P. Fiore, E. Moggi, and D. Sangiorgi, A Fully-Abstract Model for the pi-Calculus, in *Proc. LICS '96*, pp. 43–54, Washington: IEEE Press, 1996.
4. R. Heckmann, Probabilistic Domains, in *Proc. CAAP '94*, LNCS, Vol. 136, pp. 21-56, Berlin: Springer-Verlag, 1994.
5. M. C. B. Hennessy and G. D. Plotkin, Full Abstraction for a Simple Parallel Programming Language, in *Proc. MFCS '79* (ed. J. Beĉvár), LNCS, Vol. 74, pp. 108-120, Berlin: Springer-Verlag, 1979.
6. J. M. E. Hyland, G. D. Plotkin, and A. J. Power, Combining Computataional Effects: Commutativity and Sum, submitted, 2002.
7. C. Jones, *Probabilistic Non-Determinism*, Ph.D. Thesis, University of Edinburgh, Report ECS-LFCS-90-105, 1990.
8. C. Jones and G. D. Plotkin, A Probabilistic Powerdomain of Evaluations, in *Proc. LICS '89*, pp. 186–195, Washington: IEEE Press, 1989.
9. G. M. Kelly, *Basic Concepts of Enriched Category Theory*, Cambridge: Cambridge University Press, 1982.
10. G. M. Kelly and A. J. Power, Adjunctions whose Counits are Coequalizers, and Presentations of Finitary Enriched Monads, in *J. Pure Appl. Algebra*, Vol. 89, pp. 163–179, 1993.
11. P. B. Levy, Call-by-Push-Value: A Subsuming Paradigm, Ph.D. thesis, Queen Mary College, 2001.
12. P. B. Levy, Call-by-Push-Value, in *Proc. TLCA '99* (ed. J.-Y. Girard), LNCS, Vol. 1581, pp. 228-242, Berlin: Springer-Verlag, 1999.
13. S. Mac Lane, *Categories for the Working Mathematician*, Springer-Verlag, 1971.
14. E. Moggi, Computational Lambda-Calculus and Monads, in *Proc. LICS '89*, pp. 14–23, Washington: IEEE Press, 1989.
15. E. Moggi, *An Abstract View of Programming Languages*, University of Edinburgh, Report ECS-LFCS-90-113, 1989.
16. E. Moggi, Notions of computation and monads, *Inf. and Comp.*, Vol. 93, No. 1, pp. 55–92, 1991.
17. P. W. O'Hearn and R. D. Tennent, *Algol-like Languages*, Progress in Theoretical Computer Science, Boston: Birkhauser, 1997.
18. G. D. Plotkin, *Domains*, URL: http://www.dcs.ed.ac.uk/home/gdp, 1983.
19. G. D. Plotkin and A. J. Power, Adequacy for Algebraic Effects, in *Proc. FOSSACS 2001* (eds. F. Honsell and M. Miculan), LNCS, Vol. 2030, pp. 1–24, Berlin: Springer-Verlag, 2001.
20. 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.
21. A. J. Power, Enriched Lawvere Theories, in *Theory and Applications of Categories*, pp. 83–93, 2000.
22. I. Stark, *Names and Higher-Order Functions*, Ph.D. thesis, University of Cambridge, 1994.
23. I. Stark, Categorical Models for Local Names, in *Lisp and Symbolic Computation*, Vol. 9, No. 1, pp. 77–107, 1996.

and