Computational effects and operations: an overview

Gordon Plotkin and John Power [1]

Laboratory for the Foundations of Computer Science
University of Edinburgh
King's Buildings, Edinburgh EH9 3JZ, SCOTLAND

Abstract

We overview a programme to provide a unified semantics for computational effects based upon the notion of a countable enriched Lawvere theory. We define the notion of countable enriched Lawvere theory, show how the various leading examples of computational effects, except for continuations, give rise to them, and we compare the definition with that of a strong monad. We outline how one may use the notion to model three natural ways in which to combine computational effects: by their sum, by their commutative combination, and by distributivity. We also outline a unified account of operational semantics. We present results we have already shown, some partial results, and our plans for further development of the programme.

1 Introduction

Part of the enterprise of the semantics of programming languages is to separate out and analyse their features. One such is that of side-effects, the "side" indicating that they occur "on the side" while polymorphically computing something else (or, in the case of commands, nothing at all). Side-effects concern the store, but one can see other features similarly as polymorphic effects: examples are various forms of nondeterminism, printing, or jumps of various kinds. These *computational effects* form the focus of our investigation.

Computational effects invariably arise from operations such as a nondeterministic choice operation, operations for writing or reading, or operations for looking up or updating state. So we need a unified account of operations. Once we have such an account, we need to extend it to an account of dynamics, for instance given by structural operational semantics. We also need to understand the logic of effects, particularly their equational logic and its

[1] This work is supported by EPSRC grant GR/M56333.

relationship with observations. And we need a unified account of modularity, i.e., the various ways in which computational effects combine. We have begun a programme of research in this direction, and this paper outlines our achievements to date and some questions on which we are currently working.

An initial attempt to provide a unified semantic account of computational effects was made by Eugenio Moggi in [15,16,17], more recently described in [2]. Moggi primarily considered computational effects, which he called notions of computation, in the setting of call-by-value λ-calculus, with examples drawn primarily from the programming language ML. His central semantic construct was that of a strong monad on a category with finite products. His ideas have been adopted in functional programming [2], in particular in the development of the language *Haskell*. Paul Levy has also extended some of his ideas to call-by-name and to a combined calling mechanism in [14]. Our approach relates closely to Moggi's, but while he emphasised the construction of an object TX of computations of type X as primitive, we give operations a more primitive role, with TX treated as derived.

Our central notion is that of a Lawvere theory [23,7], particularly countable enriched ones. The study of ordinary Lawvere theories is equivalent to the study of universal algebra or to the study of equational theories or to the study of finitary monads on Set [1]. The study of countable enriched Lawvere theories is more general: in terms of strong monads on a category with finite products, it is equivalent to demanding that the category be locally countably presentable as a cartesian closed category and that the monad have countable rank. All the leading examples of base categories satisfy these properties: such categories include Set, $Poset$, ωCpo, all presheaf categories and, if we weaken the countability condition, all Grothendieck toposes. Some restricted categories of domains are not included. All Moggi's leading examples of monads have countable rank, except for the continuations monad. So we include the monads for exceptions, side-effects, nondeterminism, interactive input/output, and probabilistic nondeterminism, as well as a monad for local state. Several of the monads are easier to describe in terms of enriched Lawvere theories, and the concept of enriched Lawvere theory lends itself quite directly to accounts of the operations and of modularity [7].

An ordinary Lawvere theory amounts to a clone of operations and equations, and it is typically described as being freely generated by operations and equations. The generalisation to countable Lawvere theories allows the operations to be of countable arity and correspondingly for the equations. The further generalisation to enrichment allows more sophisticated arities. For instance, in the case of enrichment over $Poset$, the two element poset $\leq$ with $\bot \leq \top$ may act as an arity. It also allows correspondingly sophisticated notions of operations and equations, as we outline in Section 2. All the computational effects we consider are easily described as countable enriched Lawvere theories freely generated by such operations and equations. In all cases, the operations are computationally simple and natural, for instance *lookup* and *update* for

side-effects, and *read* and *write* for interactive input/output. The equations are similarly natural and yield corresponding programming language equations.

The correspondence, given by the Yoneda embedding, between arrows of the Lawvere theory and algebraic operations plays a subtle role here. Nondeterminism typically appears in a programming language as a polymorphic syntactic construction $M \,\mathbf{or}\, N : \sigma$ (for $M, N : \sigma$) modelled by a correspondingly polymorphic algebraic operation $\vee : TX \times TX \longrightarrow TX$ in the base category, where TX is the free algebra on X. This corresponds to a map $2 \longrightarrow 1$ in the Lawvere theory, or equivalently a map $1 \longrightarrow T2$ in the base category. We call the latter map the *generic effect* corresponding to the operation. But in examples other than the nondeterministic ones, in particular in infinitary examples such as interactive input/output, it is the generic effects, e.g., $read : 1 \longrightarrow TI$ and $write : O \longrightarrow T1$ where I is an object of inputs and O is an object of outputs, that appear more directly in a programming language. The underlying mathematics is analysed in [21], with more analysis of its programming significance in [22]. We give a very brief outline in Section 2.

The notion of countable enriched Lawvere theory provides us with a natural way to describe how computational effects may be combined. Typically, one takes the disjoint union of the operations, together with all the equations, and adds further equations relating the two classes of operations: one might add no equations, yielding the *sum* of effects; one might demand the two families of operations commute with each other, yielding the *commutative combination* of effects; or one might ask for *distributivity* of one family of operations over the other, or perhaps of each family over the other. Except for distributivity, such combinations are investigated in [7]; we outline the situation in Section 3.

A natural question, and one we have considered, is whether one can provide a unified treatment of structural operational semantics for computational effects. In fact, we can provide a unified *definition* of a structural operational semantics [20], but it is not entirely satisfactory yet. It works well for nondeterminism, probabilistic nondeterminism, printing, and combinations thereof, but it does not agree with a reasonable operational semantics for side-effects, and it does not provide an operational semantics for *handle* at all: crucially, while *handle* is an operation, it is not an algebraic one as the others mentioned so far are. We outline our work to date in Section 4.

Finally, we outline some further issues in Section 5. We consider operations such as *handle* which one might call *deconstructors*, regarding our operations as *constructors* (of effects). We are grateful to Andrzej Filinski for explaining the notion of deconstructor to us. We also outline a possible systematic way in which to extend our modelling to local phenomena such as the extension from global to local state [22]. And we outline some ideas about how to model observations and some ideas for giving an enriched notion of equational theory.

2 Enriched Lawvere Theories

In this section, we recall the definition of countable enriched Lawvere theory, give some examples, and explain the relationship with monads [7]. We do not define the notion of locally countably presentable category here [1], or what it means for a category to be locally countably presentable as a cartesian closed category. So we take the definition for granted. But we do need to recall the notion of *cotensor* [10].

The notion of cotensor is the most natural enrichment of the construction A^X for an object A of a category and a set X. Given an object A of a V-category C and given an object X of V, the cotensor A^X satisfies the defining condition that there is an isomorphism in V

$$C(B, A^X) \cong C(B, A)^X$$

V-natural in B. For instance, taking V to be $Poset$, cotensors allow us to describe not only objects such as $A \times A$ in a locally ordered category, but also objects such as $A^{\leq}$. This possibility allows us, in describing theories, to consider a greater range of arities than those given by countable sets and to incorporate inequations in the context of an elegant, coherent body of mathematics.

Given a category V that is locally countably presentable as a cartesian closed category, for instance ωCpo, one defines $V_{\aleph_1}$ to be a skeleton of the full sub-V-category of V determined by the countably presentable objects of V. It is equivalent to the free V-category with countable tensors on 1 [10,23].

Definition 2.1 A *countable Lawvere V-theory* is a small V-category L with countable cotensors together with a strict countable-cotensor preserving identity-on-objects V-functor $I : V_{\aleph_1}^{op} \longrightarrow L$. A *model* of L in a V-category C with countable cotensors is a countable-cotensor preserving V-functor $M : L \longrightarrow C$.

For any countable Lawvere V-theory L and any V-category C with countable cotensors, we thus have the V-category $Mod(L, C)$ of models of L in C; the maps of the underlying category are given by all V-natural transformations (and the naturality condition implies that they respect countable cotensors).

The usual way to describe countable Lawvere V-theories is by means of V-sketches, with the Lawvere V-theory given freely on the V-sketch. The notion of V-sketch is implicit in [12], but there does not seem to be a thorough examination of precisely that concept in the literature: we plan to provide one, together with a definition of equational V-theory, in future work. The idea is that to give a V-sketch amounts to giving operations and equations, the difference from ordinary sketches only lying in the extended notion of arity. Barr and Wells' book [1] treats ordinary sketches, i.e., the case of $V = Set$, in loving detail.

Example 2.2 The countable Lawvere theory $L_{I/O}$ for interactive input/output is the free countable Lawvere theory generated by operations $read : I \longrightarrow 1$

and $write : 1 \longrightarrow O$, where I is a countable set of *inputs* and O of *outputs*. So, interactive input/output is more directly modelled by the countable Lawvere theory than by the corresponding monad $TX = \mu Y.(O \times Y + Y^I + X)$.

Observe that the notion of enriched Lawvere theory makes operations primary, as they generate the theory. Arrows of the Lawvere theory induce, via the Yoneda embedding, operations on free models, which, as we have said in the introduction, is how operations such as $\vee$ used to model nondeterminism or probabilistic nondeterminism typically appear [5,8,9,18,19]. In other cases, especially when one has infinitary operations, for instance for interactive input/output, where I and O are typically infinite, the generic effects appear more directly in a programming language than do the corresponding operations on free models. For instance, a language with interactive input/output will typically contain types In and Out, and will contain programs $\mathbf{read} : In$ and $\mathbf{write}\, M : 1$ for $M : Out$. To give an arrow $c \longrightarrow d$ of a Lawvere theory is equivalent to giving a map (the generic effect) $d \longrightarrow Tc$ in the base category, where TX is the free algebra on X. The programs **read** and **write** have semantics given by maps $e_r : 1 \longrightarrow TI$ and $e_w : O \longrightarrow T1$, corresponding to arrows $I \longrightarrow 1$ and $1 \longrightarrow O$ in the Lawvere theory. The correspondence between operations and generic effects is analysed in [21], along with other equivalent formulations of the notion of operation, and [22] contains more analysis of examples, especially for state, albeit written in the slightly different terms of [11].

An ordinary sketch, and hence a countable Lawvere theory, yielding the side-effects monad is essentially given in [22].

Example 2.3 The countable Lawvere theory L_S for side-effects (when S is Val^{Loc} for a finite set Loc and a countable set Val) is the free countable Lawvere theory generated by operations $update : 1 \longrightarrow Loc \times Val$ and $lookup : Val \longrightarrow Loc$ subject to the seven natural equations listed in [22], four of them specifying interaction equations for *update* and *lookup* and three of them specifying commutation equations. For any category C with countable products and countable coproducts, the canonical comparison functor from $Mod(L_S, C)$ to T-Alg is an equivalence of categories, where T is the monad on C defined by $TX = (\bigsqcup_S X)^S$.

An enriched Lawvere theory is generated by operations subject to equations. Operations appear directly in describing programming languages, but equations do not. Typically, one has a notion of observation, then says two programs are equal if they are contextually equivalent relative to that notion of observation. So, in future work, we should like to develop a theory for the construction of enriched Lawvere theories from operations and observations, rather than from operations and equations as implicit in the notion of sketch. However, equations are significant in their own right as they help to provide a proof system with which to reason about semantics: in order to prove that two

complicated programs are equal, one wants a small finite number of equational axioms with which to do so; so the equations, for instance those for side-effects mentioned above, should play a role. For instance, one of the equations for side-effects is

$$update_{loc,v}(update_{loc,v'}(x)) = update_{loc,v'}(x)$$

and the corresponding program assertion is

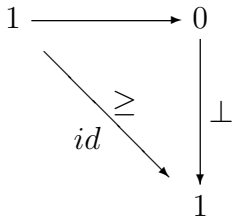
$$(l := x; \mathbf{let}\ y\ \mathbf{be}\ !l\ \mathbf{in}\ M) = (l := x; M[x/y])$$

The complications involved with higher order types will mean that such equations will not be complete for the language, but they will do part of the desired job. More analysis of the use of such equations for side-effects appears in [22].

Example 2.4 Ignoring partiality, the countable Lawvere theory L_P corresponding to a power-domain is the countable Lawvere theory freely generated by a binary operation $\vee : 2 \longrightarrow 1$ subject to equations for associativity, commutativity and idempotence, i.e., the countable Lawvere theory for a semilattice.

For an example of a countable Lawvere V-theory that does not arise freely from an unenriched countable Lawvere theory, let V be ωCpo, and consider a countable Lawvere theory for partiality.

Example 2.5 The countable Lawvere ωCpo-theory $L_\perp$ for partiality is the theory freely generated by a nullary operation $\perp\colon 0 \longrightarrow 1$ subject to the condition that there is an inequality



where the unlabelled map is the unique map determined because 0 is the initial object of $V_{\aleph_1}$ and therefore the terminal object of $V_{\aleph_1}^{op}$. A model of $L_\perp$ in ωCpo is exactly an ω-cpo with least element.

We have introduced a countable Lawvere theory L_P for a semilattice. We use the same notation to denote the countable Lawvere ωCpo-theory for a semilattice: the generators and equations are the same, but the ωCpo-theory has more objects as there are countably presentable ω-cpos other than flat ones, and these additional objects generate additional maps. It is an open problem to give an explicit description of all the countably presentable objects of ωCpo. The countable Lawvere ωCpo-theory for a semilattice is just the free countable Lawvere ωCpo-theory on the countable Lawvere theory for a semilattice.

This definition allows us to make immediate reference to the sum of effects. Using the terminology we shall define, we can make the following definition.

Example 2.6 The countable Lawvere ωCpo-theory L_N for nondeterminism is given by the sum of the countable Lawvere ωCpo-theories L_P for a semilattice and $L_\perp$ for partiality.

Another non-trivial example of a computationally natural countable Lawvere ωCpo-theory is given by probabilistic nondeterminism [5,8,9]. More detail appears in [22], albeit in the mathematical terms of [11].

Now we have some examples, we compare the notion of countable enriched Lawvere theory with that of strong monad. Given a countable Lawvere V-theory and a V-category C with countable cotensors, there is a canonical forgetful V-functor $U : Mod(L, C) \longrightarrow C$, and, when $C = V$, this forgetful V-functor has a left V-adjoint, exhibiting $Mod(L, V)$ as equivalent to the V-category T_L-Alg for the induced V-monad T_L on V.

Conversely, given a V-monad T with countable rank on V, the V-category $Kl(T)^{op}_{V_{\aleph_1}}$ determined by restricting $Kl(T)$ to the objects of $V_{\aleph_1}$ is a countable Lawvere V-theory L_T. To give a V-enriched V-monad is equivalent to giving a strong monad on V, so in order to make the comparison with Moggi's definition a little more direct, we express the main abstract result in terms of strong monads [23].

Theorem 2.7 *If V is locally countably presentable as a cartesian closed category, the constructions of T_L from L and of L_T from T induce an equivalence of categories between the category of countable Lawvere V-theories on V and the category of strong monads on V with countable rank. Moreover, the comparison V-functor is an equivalence of V-categories from $Mod(L, V)$ to T_L-Alg.*

3 Combining Computational Effects

In this section, following [7], we consider natural combinations of countable enriched Lawvere theories corresponding to natural combinations of computational effects [7]. There are three such combinations of primary importance here: the sum, the commutative combination, and a distributive combination.

The simplest of these is the sum. The category of countable enriched Lawvere theories is cocomplete, so we simply consider the sum in that category. We have already mentioned in the previous section that the countable Lawvere ωCpo-theory L_N for nondeterminism is the sum $L_P + L_\perp$ of the Lawvere ωCpo-theories L_P for a semilattice and $L_\perp$ for partiality. That is typical of the way partiality interacts with computational effects other than side-effects. Another class of examples is given by exceptions.

Proposition 3.1 *Given a set E, if L_E denotes the countable Lawvere theory*

for E nullary operations, and if L is any countable Lawvere theory, the monad T_{L_E+L} is given by $T_L(- + E)$.

The construction sending a monad T to $T(- + E)$ has been called the exceptions monad transformer [2,3]. The sum of countable Lawvere V-theories may also be characterised in terms of the categories of models, similarly to the characterisation of the commutative combination we present below. Details appear in [7].

The second combination is given by the commutative combination of theories. It is most elegantly described by a universal property in terms of categories of models, but we shall start by giving a more direct description as follows.

Definition 3.2 Given countable Lawvere V-theories L and L', the countable Lawvere V-theory $L \otimes L'$ is defined to be the countable Lawvere V-theory generated by the disjoint union of $L(A, B)$ and $L'(A, B)$ for each (A, B), respecting composition and identities of L and L', and, suppressing canonical isomorphisms, subject to commutativity of

$$\begin{array}{ccc} L(A,B) \times L'(A',B') & \longrightarrow & L(A \times B', B \times B') \times L'(A \times A', A \times B') \\ \downarrow & & \downarrow{\scriptstyle comp} \\ L(A \times A', B \times A') \times L'(B \times A', B \times B') & \xrightarrow[comp]{} & L(A \times A', B \times B') \end{array}$$

The construction giving the commutative combination is part of a symmetric monoidal structure on the category of countable Lawvere V-theories, and it is definable by the following universal property.

Theorem 3.3 *For any small V-category C with countable cotensors, the V-categories $Mod(L \otimes L', C)$ and $Mod(L, Mod(L', C))$ are coherently equivalent.*

The leading class of examples of a commutative combination of computational effects corresponds to the side-effects monad transformer [2] as follows.

Theorem 3.4 *Let L_S denote the countable Lawvere V-theory for side-effects, and let L denote any countable Lawvere V-theory. Then the monad $T_{L_S \otimes L}$ is isomorphic to $(S \times T_L -)^S$.*

The final combination of primary interest is given by distributivity. For instance, one requires a nondeterministic choice operation $\vee$ to model nondeterminism and a probabilistic choice operation $+$ to modelling probabilistic nondeterminism, and one wants distributivity of one over the other [5,8,9]. Another example of distributivity occurs when one has a pair of nondeterministic operations $\vee$ and $+$, such as for internal and external nondeterminism, and one wants distributivity of each over the other [6].

Informally, it is clear what is required here, and in the case of $V = Set$, we have a reasonable account. More generally, in the enriched setting, we have an outline treatment using an enriched notion of operad. But our account in the enriched setting is not definitive yet, so resolution of that is future work for us.

4 Operational Semantics

Lawvere theories, in contrast to monads, make operations primitive. So one may hope that they support a unified structural operational semantics for a calculus extending the computational λ-calculus introduced in [15]. We have made progress in this direction [20] as follows.

Consider a Lawvere theory generated by operations f_α subject to equations. We extend the usual notion of *value* for the computational λ-calculus to a notion of *effect value*, where an effect value is defined inductively to be a value V or a term of the form $f_\alpha(t_1, \cdots, t_n)$ where f_α has arity n and each t_i is an effect value. Each closed term t evaluates to an effect value, essentially by carrying along the operations f_α that appear in t. The details of both small-step and *collection* big-step semantics appear in [20], together with adequacy results with respect to the usual semantics of the computational λ-calculus in the Kleisli category for the induced monad T.

For nondeterminism, the induced small-step semantics is essentially the same as the usual one. The induced big-step semantics looks somewhat different owing to the use of effect values. It simply does not have a rule like

$$\frac{t_i \Rightarrow V_i}{t_1 \vee t_2 \Rightarrow V_i} \ (i = 1, 2)$$

but rather it carries along the $\vee$ while evaluating t_1 and t_2, yielding a term of the form $t(V_1, \cdots, V_n)$, where t is built using many copies of $\vee$. The equations only arise at the end as one identifies effect values semantically, e.g., $V \vee V = V$ because $\{V\} \cup \{V\} = \{V\}$. So the unified big-step semantics we propose is recognisably equivalent to the usual big-step semantics for nondeterminism but is formally a little different.

Similar remarks hold for probabilistic nondeterminism and printing. We believe that the idea should also work for infinitary operations, provided one allows corresponding infinitary syntax, and that should induce an operational semantics for generic effects. Moreover, this unified semantics extends to recursion [20]. But we do not obtain a reasonable semantics for side-effects, the problem being that all equations are ignored until the end, whereas side-effects seem to require use of them in defining a structural operational semantics. So we have still to investigate that; the distributive laws of [25,13] might help. Finally, we note that we do not have any systematic operational semantics for deconstructors.

5 Further Work

We end with an outline of our current work and plans for further questions we want to address.

For most of the computational effects we have studied, for instance side-effects, nondeterminism, and interactive input/output, the enriched Lawvere theories are generated by all of the interesting primitive operations subject to natural equations. But, as we have said, that is strikingly false for exceptions, as we only need the *raise* operation in order to generate the enriched Lawvere theory, making no use of the (non-algebraic) *handle* operation that is central to analysis of exceptions. A similar situation arises in modelling $PROLOG$. So we plan to extend our general analysis in order to incorporate such deconstructors: they should somehow be modelled in relation to the corresponding constructors, perhaps as inverses of some kind.

In [22], we showed how a monad for local state can be described in terms of operations and equations, extending the enriched Lawvere theory for global state. But we did not make precise the general nature of those operations and equations: they require a use of linear structure that we do not fully understand yet. We plan to investigate that structure, in particular with an eye towards a general mechanism allowing the extension of semantics from global definition to local definition, for instance yielding a semantics for local exceptions.

We have satisfactory accounts of the sum and commutative combinations of computational effects [7], but we do not yet have as satisfactory an account of distributive combinations. So we plan to complete that work, perhaps with particular attention devoted to specific examples.

The general structural operational semantics we have outlined in this paper following [20] yields a standard operational semantics for nondeterminism and a sensible one for probabilistic nondeterminism but, as explained in Section 4, not for side-effects or deconstructors. So we plan to define a more subtle operational semantics that includes those examples.

The relationship between ordinary Lawvere theories, finitary monads on Set, universal algebra, and equational theories has been thoroughly understood for several decades: see [1] for most of it, with [4] also relevant. Enriched Lawvere theories are defined and shown to be equivalent to finitary enriched monads in [23], and [11] (see also [24]) defines and gives an equivalence between the latter and enriched universal algebra. But there is not yet a definition of enriched equational theory together with a theorem proving it equivalent to the other notions; so we plan to provide that. It should agree with the notion of single-sorted finite cotensor sketch implicit in [12]. We should also like to extend the setting of the paper to include realisability toposes.

We also have not incorporated a treatment of observations into our analysis yet. In describing a programming language, one has syntax, including operations, and a notion of observation, the latter yielding equations determined

by contextual equivalence. We believe there is a account of this to be found at the level of generality of this paper by use of an object of observations, closely related to the use of an answer type R as used to define a continuations monad R^{R^-}. This amounts to defining a Lawvere theory not as being free on a sketch but rather in a mathematical formulation of the idea of being generated by operations subject to observational equivalence.

More generally, we should like to extend this analysis to other calling mechanisms than call-by-value, which has been the focus of our work to date. Paul Levy's work [14] on call-by-name might be helpful here, and of course call-by-need should be investigated too. And beyond that, the ideas should apply to languages built on other bases than the λ-calculus such as process calculi. The theory of operational semantics of [25] has already been developed for process calculi, so we hope to relate the two theories in this generality.

References

[1] M. Barr and C. Wells, *Category Theory for Computing Science*, Prentice-Hall, 1990.

[2] N. Benton, J. Hughes, and E. Moggi, *Monads and Effects*, APPSEM '00 Summer School, 2000.

[3] P. Cenciarelli and E. Moggi, *A Syntactic Approach to Modularity in Denotational Semantics*, CWI Technical Report, 1993.

[4] P. Freyd, *Algebra-Valued Functors in General and Tensor Products in Particular*, Colloq. Math. Wroclaw **14** (1966) 89–106.

[5] R. Heckmann, *Probabilistic Domains*, Proc. CAAP **94**, Lecture Notes in Computer Science **136** 21–56.

[6] M. Hennessy, *Algebraic Theory of Processes*, Cambridge, Massachussetts: MIT Press, 1988.

[7] M. Hyland, G.D. Plotkin, and A.J. Power, *Combining Computational Effects: Commutativity and Sum*, Submitted.

[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*, Proc. LICS **4** (1989) 186–195.

[10] G. M. Kelly, *Basic Concepts of Enriched Category Theory*, Cambridge: Cambridge University Press, 1982.

[11] G. M. Kelly and A. J. Power, *Adjunctions whose Counits are Coequalisers, and Presentations of Finitary Enriched Monads*, J. Pure Appl. Algebra **89** (1993) 163–179.

[12] Y. Kinoshita, A. J. Power, and M. Takeyama, *Sketches*, J. Pure Appl. Algebra **143** (1999) 275–291.

[13] M. Lenisa, A. J. Power, and H. Watanabe, *Distributivity for Endofunctors, Pointed and Co-Pointed Endofunctors, Monads and Comonads*, Proc. CMCS **2000** Electronic Notes in Thoeret. Comp. Sci. **33**, 2000.

[14] P. B. Levy, *Call-by-Push-Value: A Subsuming Paradigm*, Proc. TLCA **99**, Lecture Notes in Computer Science **1581** 228–242.

[15] E. Moggi, *Computational lambda-calculus and monads*, Proc. LICS **89** (1989) 14–23.

[16] E. Moggi, *An abstract view of programming languages*, University of Edinburgh, Report ECS-LFCS-90-113, 1989.

[17] E. Moggi, *Notions of computation and monads*, Inf. and Comp. **93** (1991) 55–92.

[18] G. D. Plotkin, *A Powerdomain Construction*, SIAM J. Comput. **5** (1976) 452–487.

[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*, Proc. FOSSACS **2001**, Lecture Notes in Computer Science **2030** 1–24.

[21] G. D. Plotkin and A. J. Power, *Semantics for Algebraic Operations*, Proc. MFPS **17**, Electronic Notes in Thoeret. Comp. Sci. **45**, 2001.

[22] G. D. Plotkin and A. J. Power, *Notions of Computation Determine Monads*, Proc. FOSSACS **2002**, Lecture Notes in Computer Science (to appear).

[23] A. J. Power, *Enriched Lawvere Theories*, Theory and Applications of Categories (2000) 83–93.

[24] E. P. Robinson, *Variations on Algebra: Monadicity and Generalisations of Equational Theories*, Festshrift for Rod Burstall, Formal Aspects of Computing (to appear).

[25] D. Turi and G. D. Plotkin, *Towards a Mathematical Operational Semantics*, Proc. LICS **12** (1997) 280–291.