Logic for Computational Effects: work in progress

Gordon Plotkin and John Power[*]
School of Informatics
University of Edinburgh
King's Buildings
Mayfield Road
Edinburgh EH9 3JZ
Scotland
Email: gdp@inf.ed.ac.uk, ajp@inf.ed.ac.uk

Abstract

We outline a possible logic that will allow us to give a unified approach to reasoning about computational effects. The logic is given by extending Moggi's computational λ-calculus by basic types and a signature, the latter given by constant symbols, function symbols, and operation symbols, and by including a μ operator. We give both syntax and semantics for the logic except for μ. We consider a number of sound and complete classes of models, all given in category-theoretic terms. We illustrate the ideas with some of our leading examples of computational effects, and we observe that operations give rise to natural modalities.

1 Introduction

We are attempting to develop a formal logical system for reasoning about computational effects in a unified manner. Our starting point is Eugenio Moggi's computational λ-calculus or λ_c-calculus [5, 6]. The λ_c-calculus terms provide an underlying call-by-value functional programming language. We add basic types together with a signature consisting of constant symbols, function symbols, and operation symbols. It is the operations in the signature that yield computational effects. The λ_c-calculus also provides a basic logic, which we extend to obtain a logic for computational effects. Crucially, the extension includes equations and an induction principle for the operations. The equations, which arise from observations, allow us to use the usual tools of logic in developing a proof theory and thereby a mechanisation of

[*]This work has been done with the support of EPSRC grants GR/M56333 and GR/N64571/01. Keywords: computational λ-calculus, signature, operations, modalities, $Freyd$-categories.

the logic. We shall deal with recursion later. So our development of a logical system for computational effects focuses on operations and on the relationship between equations and observations. Such operations include binary $\vee$ for modelling nondeterminism, *read* and *write* for modelling interactive input/output, and *lookup* and *update* for modelling side-effects (see [11, 12] for details). Our techniques are inherently semantic, using category theory. We seek to give an integrated, unified analysis of a programming logic with operational and denotational semantics, that applies to a range of computational effects and the constructions one makes on them, that is readily extendable to further computational features, and that is amenable to dealing with the special features of each of the effects, such as those arising from locality in addressing state.

2 Syntax

The syntax for the programming language that forms the λ_c-calculus may be taken to be identical to that for the simply typed λ-calculus [14]. So it has type constructors

$$\sigma ::= 1 \mid \sigma_1 \times \sigma_2 \mid \sigma \to \tau$$

and term constructors

$$e ::= * \mid \langle e, e' \rangle \mid \pi_i(e) \mid \lambda x.e \mid e'e \mid x$$

where x ranges over variables, $*$ is of type 1, with π_i existing for $i = 1$ or 2, all subject to the evident typing. The λ_c-calculus has two predicates: an equality predicate exactly as in the simply typed λ-calculus and a unary predicate $(-)\downarrow$ for "definedness" or "effect-freeness". The rules for the latter say $* \downarrow$, $x \downarrow$, $\lambda x.e \downarrow$ for all e, if $e \downarrow$ then $\pi_i(e) \downarrow$, and similarly for $\langle e, e' \rangle$, and that definedness is closed under equality. There are two classes of rules for $=$. The first class say that $=$ is a congruence. And the second class are rules for the basic constructions and for unit, product and functional types. The rules are closed under substitution of effect-free terms for variables. It follows from the rules for both predicates that types together with equivalence classes of terms in context form a category, with a subcategory determined by effect-free terms.

The only aspect of the λ_c-calculus that goes beyond the standard simply typed λ-calculus is the predicate $(-)\downarrow$ together with associated sophistication in the rules for $=$. The λ_c-calculus has typically been treated either as an equational logic or as an higher order intuitionistic logic, both of which were considered in [5, 6]. We primarily plan to focus on the latter, extending predicate logic (either intuitionistic or classical) by a μ (and a ν) predicate constructor $\mu X(x_1 : \sigma_1, \cdots, x_n : \sigma_n).\phi$ as in the modal μ-calculus in order to model temporal properties of programs. We also expect to modify the λ_c-calculus as we proceed, for instance by considering the predicate $\leq$ rather than $=$ in order to incorporate recursion. And we expect to weaken the axioms for $(-)\downarrow$ as, in some situations, the axiom $x\downarrow$ seems unnatural. We shall also add further type constructors such as those corresponding to symmetric monoidal structure as used in studying locality [8, 11].

A signature consists of (base) types, function symbols, predicate symbols for the programming language, and operation symbols. The constant, function, and predicate symbols are to be considered and modelled using effect-free terms in context, while the operation symbols form arbitrary terms that will not in general be effect-free.

Example 1 *Suppose one wishes to consider an idealised language for the combination of global state with nondeterminism. One might add to the λ_c-calculus a type Nat for natural numbers, function symbols 0, succ, and pred, for natural numbers, and a predicate symbol $= 0$. Then one adds operation symbols for nondeterminism and global state such as operation symbols $\vee$ for binary nondeterminism and update and lookup for state. The equational axioms to be added to the λ_c-calculus are those generated by the combination of nondeterminism and global state, as for instance in [11, 13]. One can give a systematic account of the combination of nondeterminism and global state in these terms [1, 2]. If one adds further type constructors as mentioned above, we have semantic evidence that suggests an extension to local state by adding another operation block subject to natural axioms [11], but the most elegant way to achieve that requires further investigation.*

We have many examples of such signatures and associated equations in [1, 2, 10, 11, 12, 13]. But to date, we do not have a systematic way to generate the equations from an abstract formulation of the notion of observation. Part of our ongoing work will be devoted to providing such a formulation and such a construction, as the notion of observation is computationally natural while an equational presentation allows us more easily to adopt the usual principles of proof theory, and a proof theory is central to providing a computational tool to prove that programs satisfy their specifications.

In our analysis, we plan to introduce a modality $[f]$ for each operation symbol f, where $[f](\phi_1, \cdots, \phi_n)(x)$ is defined to be

$$\forall y_1, \cdots, y_n : \sigma.(x = f(y_1, \cdots, y_n) \to \phi_1(y_1) \wedge \cdots \wedge \phi_n(y_n))$$

and a dual $\langle f \rangle$ defined by

$$\exists y_1, \cdots, y_n : \sigma.(x = f(y_1, \cdots, y_n) \wedge (\phi_1(y_1) \vee \cdots \vee \phi_n(y_n)))$$

These modalities fit with our constructions of and results about operational semantics for the λ_c-calculus together with algebraic operations in [10].

It requires some thought to see how best to extend these modalities to infinitary operations. It will involve a delicate analysis of the notion of arity, as we want to include a binary operation $\vee$ in studying nondeterminism but we also want to include arities such as types In and Out, typically to be modelled by natural numbers, in studying interactive input/output. This requires care as we seek finitistic proof theory in order to allow for easy mechanisation, and we want to isolate our use of higher order structure, as studies such as those involved with data refinement require [3], data refinement providing an application of our logic.

These modalities are different to those considered by Pitts [9] then Moggi [7]. They do not have the λ_c-calculus as an underlying language and logic, and their modalities inherently involve use of a monad, which is only implicit for us. But, subject to the reformulation those points require, their modalities may be derivable from ours; the converse is not true as they do not have our systematic account of operations. Our modalities do allow us to extend Hennessy-Milner logic, and we may consider a modal version of the logic.

3 Semantics

The most direct sound and complete class of models for the λ_c-calculus is given by (faithful) closed $Freyd$-categories [14]. These, by construction, generalise cartesian closed categories exactly by asserting the existence of a class of arrows in the ambient category subject to axioms that ensure that if one models effect-freeness of $\Gamma \vdash e : \sigma$ by an arrow in the class, one obtains a sound and complete class of models.

Thus, instead of having a cartesian closed category, one has a category C together with a subcategory B containing the same objects as C, with B having finite products, their extending along the inclusion $J : B \longrightarrow C$ to a weakened form of product, such that, for every object X of B, the functor $J(- \times X) : B \longrightarrow C$ has a right adjoint $X \to - : C \longrightarrow B$. It is evident how to model types and terms in context in a faithful closed $Freyd$-category; the predicate $(-)\downarrow$ is modelled for a term in context by the assertion that the arrow lies in B, and $=$ is modelled for two terms in context by the assertion that the two induced arrows are equal. This interpretation canonically extends to intuitionistic predicate logic, interpreting formulae by sets of arrows in the style of Kripke-Joyal semantics [4]: one can model classical logic by restricting B to be Set. Another way to model the logic is by means of a fibration.

Several natural questions arise. By syntactic construction, sound and complete classes of models exist for our logic. But can one fix B, e.g., fixing B to be Set, and still find a sound and complete class of models? How do models for fragments of the logic interact with models of the full logic? For instance, subject to some delicacy with types, the operation symbols taken together with their equations yield (enriched) Lawvere theories [1, 2, 13], which in turn yield models for the full logic. Do these models form a sound and complete class? The equations between operations yield equations between programs [12]. What additional equations are implied by this process? How do these issues extend to locality, wherein *Set* is replaced by a presheaf category and the logic naturally restricts to intutionistic logic?

Closed *Freyd*-categories were not the first sound and complete class of models given for the λ_c-calculus: the first class was given by starting with a base category B with finite products, asserting the existence of a strong monad T on B, and asserting that T have Kleisli exponentials (and satisfy the "mono requirement") [5, 6]. The λ_c-calculus was then modelled in the Kleisli category for T; the predicates were modelled by the evident subobjects of TX and $TX \times TX$. The construction of the Kleisli category characterises closed *Freyd*-categories. Both definitions together with two other sound and complete classes of models for the λ_c-calculus and the relationships between them, are explained in [14].

There are good reasons to consider different classes of models for the λ_c-calculus. For instance, one often seeks to analyse programming languages in terms of sublanguages. So one would like constructs that, given two signatures together with their equations, combine them. Several such constructs exist, two of them studied in [1, 2]. For a corresponding semantical analysis, it is natural to start with a fixed base category B, extend it in two ways, then study possible combinations of the two extensions. That is more in the spirit of monads than that of closed *Freyd*-categories. In fact, a generalisation of Lawvere theory is even better in some ways. These are questions to be addressed in association with our logic. Our overview paper [13] lists further natural semantic questions, which in turn give rise to questions about our logic: for instance, how to deal with *handle* for exceptions, which does not behave as an operation in the sense we consider here.

Further, we mentioned above the possibility of dropping the λ_c-calculus axiom $x \downarrow$. What are the models then? We have a class of models for $(-) \downarrow$ with such a deleted axiom: given a closed *Freyd*-category, one can consider the subgraph (in fact an ideal) of B given by those maps $f : X \longrightarrow Y$ such that

$$\begin{tikzcd}
TX \arrow[r, "Tf"] & TY \\
X \arrow[u, "\eta_X"] \arrow[r, "f"'] & Y \arrow[u, "\eta_Y"']
\end{tikzcd}$$

has a unique diagonal fill-in from TX to Y, where TX is defined to be

$1 \to X$. These maps can equivalently be described as the maps $g : X \longrightarrow Y$ in C for which the map $(1 \to g) : (1 \to X) \longrightarrow (1 \to Y)$ in B factors through η_Y. But we have not yet considered a sound and complete class of models; and the models we have described involve B although its arrows no longer appear as the semantics of any syntactic entity.

References

[1] Hyland, M., G. D. Plotkin, G. D. and Power, A. J. (2002) Combining computational effects: commutativity and sum. *Proceedings of IFIP Conf. On Theoretical Computer Science*, Montreal, Canada, 25–30 August pp. 474–484. Kluwer, Dordrecht.

[2] Hyland, M., Plotkin, G. D. and Power, A. J. Combining computational effects: sum and tensor, submitted.

[3] Kinoshita, Y. and Power, A.J. (1999) Data Refinement in Call-by-Value Languages. *Proceedings of CSL 99, Lecture Notes in Computer Science 1683*, Madrid, Spain, 20–25 September, pp. 562–576. Springer-Verlag, Berlin.

[4] Mac Lane, S. and Moerdijk, I. (1992) *Sheaves in Geometry and Logic*, Springer-Verlag, Berlin.

[5] Moggi, E. (1989) Computational lambda-calculus and monads. *Proc. LICS '89*, Asilomar, California, 5–8 June, pp 14–23, IEEE Press, Washington.

[6] Moggi, E. (1991) Notions of computation and monads. *Information and Computation*, **93**, 55–92.

[7] Moggi, E. (1995) A Semantics for Evaluation Logic. *Fundamenta Informaticae*, **22**, 117–152.

[8] O'Hearn, P. W. and Tennent, R. D. (1997) *Algol-like Languages*, Birkhauser, Boston.

[9] Pitts, A. M. (1991) Evaluation Logic. *Proc. 4th Higher Order Workshop, Banff 1990, Workshops in Computing, 283*, Alberta, Canada, 10–14 September, pp 162–189, Springer-Verlag, Berlin.

[10] Plotkin, G. D. and Power, A.J. (2001) Adequacy for Algebraic Effects. *Proc. FOSSACS 2001, Lecture Notes in Computer Science 2030*, Genova, Italy, 2–6 April, pp 1–24, Springer-Verlag, Berlin.

[11] Plotkin, G. D. and Power, A. J. (2002) Notions of Computation Determine Monads, *Proc. FOSSACS 2002, Lecture Notes in Computer Science 2303*, Grenoble, France, 8–12 April, pp 342–356, Springer-Verlag, Berlin.

[12] Plotkin, G. D. and Power, A. J. (to appear) Algebraic Operations and Generic Effects. *Applied Categorical Structures.*

[13] Plotkin, G. D. and Power, A. J.(2002) Computational effects and operations: an overview. *Proc. 6th Workshop Domains, Birmingham, Electronic Notes in Theoretical Computer Science 73*, Birmingham, United Kingdom, 16–19 September, Elsevier (http:www.elsevier.nl/locate/entcs/).

[14] Power, A. J. (2001) Models of the computational λ-calculus. *Proc. MFCSIT 2000, Electronic Notes in Theoretical Computer Science 40*, Cork, Ireland, 20–21 July, Elsevier (http:www.elsevier.nl/locate/entcs/).