Extensional Universal Types for Call-by-Value

Kazuyuki Asada

Research Institute for Mathematical Sciences, Kyoto University, Japan
asada@kurims.kyoto-u.ac.jp

Abstract. We propose $\lambda_c 2_\eta$*-calculus*, which is a second-order polymorphic call-by-value calculus with *extensional* universal types. Unlike product types or function types in call-by-value, extensional universal types are genuinely right adjoint to the weakening, i.e., β-equality and η-equality hold for not only values but all terms. We give monadic style categorical semantics, so that the results can be applied also to languages like Haskell. To demonstrate validity of the calculus, we construct concrete models for the calculus in a generic manner, exploiting "relevant" parametricity. On such models, we can obtain a reasonable class of monads consistent with extensional universal types. This class admits polynomial-like constructions, and includes non-termination, exception, global state, input/output, and list-non-determinism.

1 Introduction

Polymorphic lambda calculi like System F [11,30] have been widely studied and also used in some practical programing languages like ML and Haskell as their semantical background. With universal types in them, we can abstract terms defined uniformly for each type into one term, which improves usability, efficiency, readability and safety. Moreover in impredicative polymorphic lambda calculi, we can encode various datatypes like products, coproducts, initial algebras, and final coalgebras with universal types. These datatypes have merely *weak* universal properties in System F, but they are truly universal if we assume relational parametricity [31,34,29,1,16,7].

As well as purely functional calculi, extensions of polymorphic lambda calculi to call-by-name calculi/languages with some computational effects have been studied: with a fixed-point operator [28,4,5], and with first-class continuations [14].

While call-by-value is one of the most frequently used evaluation strategy in practical programing languages, extensions to call-by-value polymorphic languages raise a subtle problem on the treatment of universally typed *values*. Suppose that we have a type-abstracted term $\Lambda\alpha.M$. Then, is this a value at all? Or should we treat this as a value, only when M is a value? It corresponds to the choice whether we evaluate M under the type abstraction, or do not (as function closure). This problem does not occur in modern ML, because, due to *value restriction*, M must always be a value.

When we set $\Lambda\alpha.M$ as a value only when so is M, then we can obtain extensional universal types, i.e., "type" η-equality, for some reasonable class of

effects. We call the η-equality $\Lambda\alpha.M\alpha = M$ ($\alpha \notin \mathrm{FTV}(M)$) for type abstraction *type η-equality*, and also the β-equality $(\Lambda\alpha.M)\,\sigma = M\,[\sigma/\alpha]$ for type abstraction *type β-equality*. Clearly, if we treat $\Lambda\alpha.M$ as a value for any term M, then type η-equality does not hold in general, as function types in call-by-value.

We expect that such type η-equality can be useful for program transformation to optimize programs. Also we can use it when reasoning with *parametricity for call-by-value*, for type η-equality is often used in parametric reasoning.

In the present paper we propose a second-order polymorphic call-by-value lambda calculus with extensional universal types. We give syntax and its sound and complete categorical semantics, and describe how we can construct concrete models with a variety of computational effects.

Our main contribution is to show that, for some parametric models or semantic setting, we can obtain a reasonable class of effects which are compatible with extensional universal types. The class includes non-termination, exception, global state, input/output, and list-non-determinism. We can not show at present whether (commutative) nondeterminism (like finite or infinite powersets, multisets and probabilistic non-determinism) and continuations belong to the class of models or not.

We use Moggi's monadic semantics [25,26,27,2] rather than direct-style semantics for call-by-value calculi [10,21]. One reason is that it is easier to represent the class of effects considered in the present paper. The monads modeling the effects mentioned above are respectively lifting monad $(-)+1$, exception monads $(-)+E$, global state monads $((-)\times S)^S$, input monads $\mu\beta.\,(-)+\beta^U$, output monads $\mu\beta.\,(-)+(U\times\beta)$, and the list monad $\mu\beta.1+(-)\times\beta$. As these examples, this class is characterized as that of all "polynomial" monads constructed by constants, products, separated sum, powers, final coalgebras, and *linearly* initial algebras; the last construction is explained in Section 6.4.

One more reason to use monadic semantics is because we can apply the results to call-by-name (meta-)languages with monads like Haskell, as well as to call-by-value languages as object languages like ML. In (some simplified) Haskell, only fixed-point operators involving the non-termination effect exists in the call-by-name object language, and other effects are treated via monadic style translations with various monads. The semantics which we give with "relevant" parametricity and fixed-point operators in Section 6 is close to such subset of Haskell, with extensional universal types.

1.1 Related Work

Harper et al. studied two kinds of call-by-value (and two kinds of call-by-name) extensions of the higher order polymorphic lambda calculus System F_ω with first-class continuations [12]. The difference between the two call-by-value extensions is exactly as above, i.e., whether, for a type-abstracted term, we evaluate the inside of the body or not. They took an operational semantics-based approach, and noted there the failure of proving the preservation theorem which asserts that a closed answer-typed term is evaluated to a closed answer-typed term. Thus first-class continuations raise difficulty to obtain extensional universal types.

Recently in [22], Møgelberg studied polymorphic FPC, a call-by-value functional language with recursive types and hence a fixed-point operator. The universal types in this language are not designed to be extensional. However, it seems that we can make it extensional because the effect used there is nontermination, though the problems to adjust other features of the language like recursive types are not trivial.

An extension of System F with parametricity to call-by-push-value [20] was also studied [23]. The paradigm of call-by-push-value is similar to that of the monadic metalanguages, and it might be possible to express our results in terms of call-by-push-value. Call-by-push-value can be also used to translate call-by-name languages as done in [24].

1.2 Outline

In Section 2, we introduce two *second order* computational lambda calculi. One is $\lambda_{\mathrm{c}}2_{\eta}$-calculus which has extensionality on universal type, and the other is $\lambda_{\mathrm{c}}2$-calculus which does not.

Section 3 is devoted to preliminaries for semantics in later sections. In Section 4, we give categorical semantics for the calculi given in Section 2. These sections 3 and 4 may be skipped by readers who are not particularly interested in general semantic framework.

In Section 5, we start to look at concrete models for $\lambda_{\mathrm{c}}2_{\eta}$-calculus. First we describe a class of monads which admit extensional universal types, then we show in two sections that they indeed form concrete models. In Section 5, we treat parts which hold in relatively general, i.e., only by categorical property and by parametricity. In Section 6, we consider more specific models with fixed-point operators and "relevant" parametricity.

Lastly, we give some concluding remarks in Section 7.

2 Second Order Computational Lambda Calculi

In this section, we introduce two calculi: *second order computational λ-calculus* ($\lambda_{\mathrm{c}}2$-*calculus* for short) and *second order computational λ-calculus with extensional universal types* ($\lambda_{\mathrm{c}}2_{\eta}$-*calculus* for short). These are extensions of Moggi's computational lambda calculus (λ_{c}-calculus), listed in Figure 1, with universal types.

We give the $\lambda_{\mathrm{c}}2$-calculus in Figure 2, and the $\lambda_{\mathrm{c}}2_{\eta}$-calculus in Figure 3. Note that the classes of values include e.g. $\pi_1 \langle V, V' \rangle$. A "value" here means an "effect-free" term, rather than a canonical form.

The differences between the $\lambda_{\mathrm{c}}2$-calculus and the $\lambda_{\mathrm{c}}2_{\eta}$-calculus are not in the definitions of types or terms but only in equation theories, i.e., the values and the axioms. The universal types in the $\lambda_{\mathrm{c}}2$-calculus satisfy type β-equality for any term and type η-equality for only values, while those in the $\lambda_{\mathrm{c}}2_{\eta}$-calculus satisfy type β- and η-equality for any term. We call such universal types with full type η-equality *extensional* universal types. In the paper we put the focus on the

$$\begin{array}{rrl}
\text{Types} & \sigma & ::= b \mid \sigma \to \sigma \mid 1 \mid \sigma \times \sigma \\
\text{Terms} & M & ::= x \mid c^\sigma \mid \lambda x^\sigma.M \mid MM \mid * \mid \langle M, M\rangle \mid \pi_0 M \mid \pi_1 M \\
\text{Values} & V & ::= x \mid c^\sigma \mid \lambda x^\sigma.M \mid * \mid \langle V, V\rangle \mid \pi_0 V \mid \pi_1 V \\
\text{Evaluation Contexts} & E & ::= [-] \mid EM \mid VE \mid \langle E, M\rangle \mid \langle V, E\rangle \mid \pi_0 E \mid \pi_1 E
\end{array}$$

where b ranges over base types, and c^σ ranges over constants of type σ

Typing Rules:

$$\frac{}{\Gamma \vdash x : \sigma}\,(x{:}\sigma \in \Gamma) \qquad \frac{}{\Gamma \vdash c^\sigma : \sigma} \qquad \frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash \lambda x^\sigma.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$$

$$\frac{}{\Gamma \vdash * : 1} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle : \sigma \times \tau} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \pi_0 M : \sigma} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \pi_1 M : \tau}$$

Axioms:

$$\begin{array}{lll}
(\lambda x^\sigma.M)\, V = M\,[V/x] & & \pi_i\,\langle V_0, V_1\rangle = V_i \quad (i = 0, 1) \\
\lambda x^\sigma.Vx = V \quad (x \notin \mathrm{FV}\,(V)) & & \langle \pi_0 V, \pi_1 V\rangle = V \\
V = * \quad (V : 1) & & (\lambda x^\sigma.E\,[x])\, M = E\,[M] \quad (x \notin \mathrm{FV}\,(E))
\end{array}$$

Fig. 1. The λ_{c}-calculus

$\lambda_{\mathrm{c}} 2_\eta$-calculus, to demonstrate how many effects are consistent with extensional universal types.

In Section 6, we give relevant parametric models with fixed-point operators as concrete models for the $\lambda_{\mathrm{c}} 2_\eta$-calculus. And there is a reasonably wide class of monads on the models, including non-termination, exception, global state, input, output, and list-non-determinism.

Such fixed-point operators are not included in the syntax of the $\lambda_{\mathrm{c}} 2_\eta$-calculus. Also, the $\lambda_{\mathrm{c}} 2_\eta$-calculus includes no proper axioms which induce computational effects, as well as the λ_{c}-calculus. However, with such models, we will be able to extend the $\lambda_{\mathrm{c}} 2_\eta$-calculus with suitable axioms and terms which induce such computational effects, and also with call-by-value fixed-point operators [15]. Alternatively, we will also be able to define "second order monadic metalanguages" which have call-by-name fixed-point operators [32] and in which we can simulate such extended $\lambda_{\mathrm{c}} 2_\eta$-calculi by use of corresponding monads.

3 Preliminaries for Semantics

This section is devoted to preliminaries for some category theoretical notions.

Notation: '$\Rightarrow$' is used for exponentials in a CCC (cartesian closed category). An identity on A is written also as just A. For 2-cells, '$*$' and '$\circ$' mean horizontal and vertical compositions respectively.

For a functor p being a fibration, we say an object X is *over* an object I (resp. an arrow f is *over* an arrow u) if $pX = I$ (resp. $pf = u$). A functor

$$\begin{array}{lll} \text{Types } \sigma ::= \ldots \mid \alpha \mid \forall\alpha.\sigma & & \text{Values } V ::= \ldots \mid \Lambda\alpha.M \\ \text{Terms } M ::= \ldots \mid \Lambda\alpha.M \mid M\sigma & \text{Evaluation Contexts } E ::= \ldots \mid E\sigma & \end{array}$$

Typing Rules: All the rules in the λ_{c}-calculus in which a kind context Ξ is added to the all contexts, and:

$$\frac{\Xi, \alpha \mid \Gamma \vdash M : \sigma}{\Xi \mid \Gamma \vdash \Lambda\alpha.M : \forall\alpha.\sigma} \ (\alpha \notin \mathrm{FTV}(\Gamma)) \qquad \frac{\Xi \mid \Gamma \vdash M : \forall\alpha.\sigma}{\Xi \mid \Gamma \vdash M\tau : \sigma\,[\tau/\alpha]}$$

Axioms: All the axioms in the λ_{c}-calculus where the evaluation contexts are extended as above, and:

$$(\Lambda\alpha.M)\,\sigma = M\,[\sigma/\alpha] \qquad \Lambda\alpha.V\alpha = V \quad (\alpha \notin \mathrm{FTV}(V))$$

Fig. 2. The $\lambda_{\mathrm{c}}2$-calculus, extended from the λ_{c}-calculus

$$\begin{array}{lll} \text{Types } \sigma ::= \ldots \mid \alpha \mid \forall\alpha.\sigma & & \text{Values } V ::= \ldots \mid \Lambda\alpha.V \mid V\sigma \\ \text{Terms } M ::= \ldots \mid \Lambda\alpha.M \mid M\sigma & \text{Evaluation Contexts } E ::= \ldots \mid E\sigma & \end{array}$$

Typing Rules are the same as those in the second order λ_{c}-calculus ($\lambda_{\mathrm{c}}2$-calculus).
Axioms: All the axioms in the λ_{c}-calculus where the evaluation contexts are extended as above, and:

$$(\Lambda\alpha.M)\,\sigma = M\,[\sigma/\alpha] \qquad \Lambda\alpha.M\alpha = M \quad (\alpha \notin \mathrm{FTV}(M))$$

Fig. 3. The $\lambda_{\mathrm{c}}2_{\eta}$-calculus, extended from the λ_{c}-calculus

$p : \mathbb{E} \longrightarrow \mathbb{B}$ is called a *fibration* if for any arrow $u : J \longrightarrow I$ in $\mathbb{B}$ and object X over I, there is an object u^*X over J and an arrow $\bar{u}X : u^*X \longrightarrow X$ over u which is *cartesian*: for any object Z over K, arrow $h : Z \longrightarrow X$ in $\mathbb{E}$ and arrow $v : K \longrightarrow J$ in $\mathbb{B}$ s.t. $ph = u \circ v$, there is unique arrow $g : Z \longrightarrow u^*X$ over v satisfying $h = \bar{u}X \circ g$.

For most of basic fibred category theory in the context of categorical semantics, we refer to [19]. In the present paper we use fibred-category-theoretical notions only *with a fixed base category*. $\mathbf{Fib}_{\mathbb{B}}$ is the 2-category of all fibrations, fibred functors and fibred natural transformations over a fixed base category $\mathbb{B}$.

A *fibred monad* (over $\mathbb{B}$) is an *internal* monad *in* the 2-category $\mathbf{Fib}_{\mathbb{B}}$ [33]. For a fibred monad T, we can construct the Kleisli adjunction $U \vdash F : p \longrightarrow p_T$ *in* $\mathbf{Fib}_{\mathbb{B}}$ (cf. [17]). For a fibration $(p, \otimes, \mathrm{I}, \alpha, \lambda, \rho)$ with fibred products, for which we use the tensor notation, a *strong fibred monad* (T, η, μ, τ) over p is a fibred monad (T, η, μ) and fibred natural transformation $\tau : \otimes \circ (p \times T) \Longrightarrow T \circ \otimes$ satisfying the four equations: $\lambda * T = (T * \lambda) \circ (\tau * \langle \mathrm{C}_{\mathrm{I}}, p \rangle)$, $(\tau * (\otimes \times p)) \circ \big(\alpha * \big(p^2 \times T\big)\big) = (T * \alpha) \circ (\tau * (p \times \otimes)) \circ (\otimes * \tau)$, $\eta * \otimes = \tau \circ (\otimes * (p \times \eta))$, $\tau \circ (p \times \mu) = (\mu * \otimes) \circ (T * \tau) \circ (\tau * (p \times T))$, where $\mathrm{C}_{\mathrm{I}} := \mathrm{I} \circ ! : p \longrightarrow 1 \longrightarrow p$ is the constant fibred functor of I.

4 Categorical Semantics for Second Order Computational Lambda Calculi

In this section, we give categorical semantics for the $\lambda_c 2$-calculus and the $\lambda_c 2_\eta$-calculus. As mentioned in the introduction, we use monadic style semantics rather than direct style semantics for call-by-value. They are equivalent [10], since we include into the definitions below the equalizing requirement, which asks each component η_A of the unit of a monad T is an equalizer of $T\eta_A$ and η_{TA}.

First we define a "λ_c-version of polymorphic fibration", which models the λ_c-calculus and type variables, but does not model universal types.

Definition 1 *A* polymorphic λ_c-model *consists of*

(i) *a cartesian polymorphic fibration, i.e. a fibration $p : \mathbb{E} \longrightarrow \mathbb{B}$ which has products in the base category, generic object Ω and fibred products,*

(ii) *a fibred strong monad T on p satisfying the equalizing requirement fiberwise, and*

(iii) *fibred Kleisli exponentials, i.e., a family of right adjoint functors $X \multimap (-)$ to the composite functors $F_I((-) \times X) : \mathbb{E}_I \stackrel{(-)\times X}{\longrightarrow} \mathbb{E}_I \stackrel{F_I}{\longrightarrow} (\mathbb{E}_T)_I$ where I is an object in $\mathbb{B}$ and X is over I in p, and which satisfies the Beck-Chevalley condition:*
for any $u : J \longrightarrow I$ in $\mathbb{B}$ and X over I in p, the natural transformation from $u^* \circ (X \multimap (-))$ to $(u^*X \multimap (-)) \circ {u^*}_T$ induced by the natural isomorphism from $F_J((-) \times u^*X) \circ u^*$ to ${u^*}_T \circ F_I((-) \times X)$ is isomorphic. □

If we have the first and the second item in the above definition, and if the fibration is a fibred CCC, then we get the third item of polymorphic λ_c-model for free by composition of the Kleisli adjunction and the adjunction defining cartesian closedness. All the examples in the present paper are such models.

In the following, we use Ω for a generic object of a fibration, if it exists and there is no ambiguity.

Definition 2 *Let p be a polymorphic λ_c-model. By change-of-base of the Kleisli embedding $F : p \longrightarrow p_T$ along $(-) \times \Omega : \mathbb{B} \longrightarrow \mathbb{B}$, we have a fibred functor $F^\Omega : p^\Omega \longrightarrow {p_T}^\Omega$. In addition, the reindexing functors induced by the projections give a "weakening" fibred functor $\pi^*_{(-),\Omega} : p \longrightarrow p^\Omega$. Then,*

- Kleisli simple Ω-products *is a fibred right adjoint functor to the composite of these two fibred functors, and*
- *a* $\lambda_c 2$-model *is a polymorphic λ_c-model p which has Kleisli simple Ω-products.* □

As the case of fibred Kleisli exponentials for a fibred CCC, if we have a polymorphic λ_c-model p which has simple Ω-products, then we obtain Kleisli simple Ω-products in a quite similar way, and hence a $\lambda_c 2$-model for free. So any $\lambda 2$-fibration [19] which has the second item in Definition 1 forms a $\lambda_c 2$-model.

Definition 3 *A* $\lambda_c 2_\eta$-model *is a polymorphic λ_c-model p whose Kleisli fibration p_T has simple Ω-products.* □

It is easily seen that all $\lambda_{\mathrm{c}}2_{\eta}$-models are $\lambda_{\mathrm{c}}2$-models.

An interpretation of the $\lambda_{\mathrm{c}}2_{\eta}$-calculus in a $\lambda_{\mathrm{c}}2_{\eta}$-model is defined by inductions on type formation and on typing rules. This can be done in a quite similar way to that of λ_{c}-calculus [25], with referring to that of System F [19], too.

On the type abstraction rule, $[\![\Xi \mid \Gamma \vdash \Lambda\alpha.M : \forall\alpha.\sigma]\!] : [\![\Gamma]\!] \longrightarrow [\![\forall\alpha.\sigma]\!] := \prod [\![\sigma]\!]$ in $(p_T)_{[\![\Xi]\!]}$ is defined as the transposition of the composite $[\![\Xi, \alpha \mid \Gamma \vdash M : \sigma]\!] \circ$ ("canonical isomorphism") $: \pi^*[\![\Gamma]\!] \cong [\![\Gamma]\!] \longrightarrow [\![\sigma]\!]$ in $(p_T)_{[\![\Xi,\alpha]\!]} = (p_T)_{[\![\Xi]\!]\times\Omega}$ under simple Ω-products adjointness.

On the type application rule, first we have an arrow $[\![\Xi \mid \Gamma \vdash M : \forall\alpha.\sigma]\!] : [\![\Gamma]\!] \longrightarrow [\![\forall\alpha.\sigma]\!] := \prod [\![\sigma]\!]$ in $(p_T)_{[\![\Xi]\!]}$ with its transposition $m : \pi^*[\![\Gamma]\!] \longrightarrow [\![\sigma]\!]$ in $(p_T)_{[\![\Xi]\!]\times\Omega}$, and also have an arrow $[\![\tau]\!]^\sharp : [\![\Xi]\!] \longrightarrow \Omega$ in $\mathbb{B}$, where $(-)^\sharp$ is the correspondence induced by the generic object. Then $[\![\Xi \mid \Gamma \vdash M\tau : \sigma\,[\tau/\alpha]\,]\!] : [\![\Gamma]\!] \longrightarrow [\![\sigma\,[\tau/\alpha]\,]\!]$ in $(p_T)_{[\![\Xi]\!]}$ is defined as the composite $[\![\Gamma]\!] \cong \left\langle \mathrm{id}_{[\![\Xi]\!]}, [\![\tau]\!]^\sharp \right\rangle^* \pi^*[\![\Gamma]\!] \longrightarrow \left\langle \mathrm{id}_{[\![\Xi]\!]}, [\![\tau]\!]^\sharp \right\rangle^* [\![\sigma]\!] \cong [\![\sigma\,[\tau/\alpha]\,]\!]$, where the last isomorphism is the canonical isomorphism from a semantic type substitution lemma proved routinely, and the middle arrow is the reindexing of m by $\left\langle \mathrm{id}_{[\![\Xi]\!]}, [\![\tau]\!]^\sharp \right\rangle$.

Theorem 4 *The class of all λ_c2_{η}-models are sound and complete for the λ_c2_{η}-calculus with respect to the above interpretation.* □

Proof. *Soundness follows routinely by induction. For completeness, a term model can be constructed in the same way as those for System F [19] and for the λ_c-calculus [10].* ■

Now we shall introduce a subclass of the class of $\lambda_{\mathrm{c}}2_{\eta}$-models. All concrete models we give in the present paper belong to this class.

Definition 5 *A* monadic $\lambda_{\mathrm{c}}2_{\eta}$-model *is a polymorphic λ_c-model such that p and p_T has simple Ω-products and the Kleisli embedding $F : p \longrightarrow p_T$ preserves them.* □

The notion of monadic $\lambda_{\mathrm{c}}2_{\eta}$-models is natural for monadic style translation. The preservation of simple Ω-products by the identity-on-objects functor F means that we use the same universal types before and after the monadic style translation. If we are interested in extensional universal types only for call-by-value languages themselves, then we should expand the semantics to the class of $\lambda_{\mathrm{c}}2_{\eta}$-models. However, if we are also interested in monadic metalanguages, then it is very simple and hence important to use the same universal types between the value (non-effect) language (base category of a monad) and various effectful languages (Kleisli categories), since in a monadic metalanguage we may use more than one monad, as we do so in Haskell.

Proposition 6 *Let p be a polymorphic λ_c-model such that p has simple Ω-products.*

(1) If p_T has simple Ω-products, then the fibred right adjoint functor $U : p_T \longrightarrow p$ preserves simple Ω-products.

(2) p is a monadic $\lambda_c 2_\eta$-model, (i.e., p_T has simple Ω-products and the fibred functor $F : p \longrightarrow p_T$ preserves simple Ω-products,) if and only if the underlying fibred endofunctor T of the monad preserves simple Ω-products. □

By this proposition, it turns out that, in order to find a monadic $\lambda_c 2_\eta$-model, we only have to pay attention to the underlying endofunctor of a monad, without considering η nor μ. This is because the canonical arrow $T(\prod A) \longrightarrow \prod TA$ respects η and μ, since the reindexings preserve them. This simplification is very useful as we use in the next section.

5 Concrete Models

In this section, we start to study concrete monadic $\lambda_c 2_\eta$-models.

In order to obtain monadic $\lambda_c 2_\eta$-models, we use Proposition 6 (2). For a $\lambda 2$-fibration and a fibred strong monad on it satisfying the equalizing requirement, if the underlying fibred endofunctor of the monad preserves simple Ω-products, i.e., if $T(\prod A) \cong \prod TA$ holds, then they form a monadic $\lambda_c 2_\eta$-model. So we analyze what kind of fibred functors preserve simple Ω-products in $\lambda 2$-fibrations.

5.1 The Class of Monads

We describe here the class of monads considered in the present paper. For the sake of simplicity, we concentrate on the underlying endofunctors of monads.

First let us consider the following class of functors. In order to consider initial algebras and final coalgebras, we consider multi-ary functors as well as unary endofunctors.

$$T ::= \gamma \mid C \mid 1 \mid T \times T \mid T + T \mid T^C \mid \mu\gamma.T \mid \nu\gamma.T$$

This class is constructed inductively by projections, (i.e., variables γ,) constant functors, finite products, binary coproducts, powers, (i.e., exponentials whose domains are constants,) initial algebras, and final coalgebras. Basically we would like to consider something like the above class, but there is a problem.

All we need to show is that the underlying fibred endofunctors of fibred monads preserve simple Ω-products, so it is sufficient to prove that the constructions defining inductively the above class (in the fibred setting) preserve simple Ω-products. However, it is shown in Section 6.1 that coproducts do not necessarily commute with universal quantifier in System F even with parametricity, and so we need some special morphism in a $\lambda 2$-fibration considered here. In the paper, we use models having fixed-point operators to resolve it. So, in order to avoid well-known conflict between fixed-point operators and coproducts, we consider linear (more precisely, *relevant*) models by which we can relax relational parametricity inducing coproducts as in [28,5], and replace coproducts with separated sums and initial algebras with *linearly initial algebras*. Linearly initial algebras are, roughly, something which are initial algebras only in a linear model. These notions of separated sums and linearly initial algebras will be explained in Section 6.3 and 6.4 respectively.

Now let us describe the class of monads considered in the present paper. It is the class of all fibred monads whose underlying fibred endofunctors are included in the following class (1).

$$T ::= \gamma \mid 1 \mid T \times T \mid T^C \mid \nu\gamma.T \mid C \mid T \oplus T \mid \mu^{\circ}\gamma.T \tag{1}$$

In the above, $\oplus$ is separated sum, and $\mu^{\circ}\gamma.T$ is linearly initial algebras. These constructions form fibred functors, see Sections 5.2, 6.3, and 6.4 for final coalgebras, separated sums, and linearly initial algebras respectively.

From now on we show that simple Ω-products are preserved by (i): products, powers, final coalgebras, (ii): constant fibred functors, (iii): separated sums, and linearly initial algebras.

On the constructions of (i), we can show that they preserve simple Ω-products by their categorical universal property, because they are right adjoint as well as simple Ω-products. For constants of (ii), however, we need more *property* like parametricity, and for separated sums and linearly initial algebras of (iii), we need additionally more *structures* like fixed-point operators. We consider products, powers, final coalgebras in the next subsection, constants are treated in the next, and separated sums and linearly initial algebras are postponed to the next section.

In the paper, we do not mind whether such constructions as above are available or not, and do only show that they preserve simple Ω-products if they exist. If we try to show existence of e.g. parameterized initial algebras for multi-ary functor, to treat the list monad and input monads, then we need to introduce the notion of "fibrations and fibred functors *enriched* over a monoidal *fibration*", and perhaps need to use *fibrations with indeterminates* [17]. For space reason we postpone such detailed work elsewhere, which is less problematic because the existence of the constructions represented in syntax is well known by polymorphic encoding with parametricity.

5.2 Products, Powers and Final Coalgebras

Here, we investigate constructions which preserve simple Ω-products by only universal property.

Lemma 7 *Let $\mathbb{B}$ be a cartesian category, and K be any object of $\mathbb{B}$.*

(1) The subcategory of $\mathbf{Fib}_{\mathbb{B}}$ consisting of all fibrations having simple K-products, and all fibred functors preserving simple K-products, is cartesian subcategory.
(2) Let p be a fibration having simple K-products and fibred finite products. Then the fibred functors $\times : p \times p \longrightarrow p$, and $1 : 1 \longrightarrow p$ preserve simple K-products.
(3) Moreover assume that p is a fibred CCC. Then for any X over 1 in p, the "power" fibred functor $X \Rightarrow (-) : p \longrightarrow p$ preserves simple K-products. □

We can also add final coalgebras into the above list, if they exist sufficiently in the sense described below. The same things hold from here to Definition 8 for both initial algebras and final coalgebras in the dual way, so we do with initial algebras, since more examples of effects use initial algebras.

Let p, q be fibrations with the same base category $\mathbb{B}$, and $F : q \times p \longrightarrow p$ be a fibred functor. We say that $F : q \times p \longrightarrow p$ *has initial algebras with parameters*, if for any X over I in q, the endofunctor $F(X, -) : \mathbb{E}_I \longrightarrow \mathbb{E}_I$ has initial algebra $(\mu FX, \alpha_X : F(X, \mu FX) \longrightarrow \mu FX)$, and if the reindexings preserve them, i.e., for any X over I in q and any arrow $u : J \longrightarrow I$ in $\mathbb{B}$, the unique algebra map from the initial algebra $\mu F(u^*X)$ to the algebra $F(u^*X, u^*(\mu FX)) \cong u^*F(X, \mu FX) \stackrel{u^*\alpha_X}{\longrightarrow} u^*(\mu FX)$ is isomorphism.

If F is in the case, then the assignment which maps an object X over I in q to the object μFX over I in p extends to the unique fibred functor $\mu F : q \longrightarrow p$ such that the family of maps $(\alpha_X : F(X, \mu FX) \longrightarrow \mu FX)_X$ forms into a fibred natural transformation from $F(-, \mu F-)$ to μF.

Definition 8 *For a fibred functor $F : q \times p \longrightarrow p$ having initial algebras with parameters, there is the* initial algebras fibred functor $\mu F : q \longrightarrow p$ *as above.*

Similarly, if $F : q \times p \longrightarrow p$ has final coalgebras with parameters*, which is defined in the dual way, then we have the* final coalgebras fibred functor $\nu F : q \longrightarrow p$ *with the fibred natural transformation from νF to $F(-, \nu F-)$.* □

Lemma 9 *Let $\mathbb{B}$ be a cartesian category, K be an object of $\mathbb{B}$, p, q be fibrations having simple K-products, and $F : q \times p \longrightarrow p$ be a fibred functor having final coalgebras with parameters. Then, if F preserves simple K-products, the fibred functor $\nu F : q \longrightarrow p$ also preserves simple K-products.* □

Typical usage of the above is to get endofunctors with $q = p$ (or p^n), but we will use a coKleisli fibration for q with linear models in Section 6.4.

5.3 Constant

In this short subsection, the construction using constants is added.

For a $\lambda 2$-fibration p and any object X over 1, the fibred functor $X : 1 \longrightarrow p$ preserves simple Ω-products if p satisfies suitable parametricity, including relational parametricity, linear parametricity, and focal parametricity.

In this case of constants we do not need additional arrows differently from the case in the next section, because we can adopt the (unique) projection as the inverse arrow of the canonical (diagonal) arrow $X \longrightarrow \prod X$.

At this point, we can add global state monads $((-) \times S)^S$ and output monads $\mu^\circ\beta.(-) + (U \times \beta) \cong (-) \times (\mu^\circ\beta.1 + (U \times \beta))$ to the class of monads which form monadic $\lambda_c 2_\eta$-models.

6 Separated Sums and Linearly Initial Algebras

We continue to show how we construct monads compatible with extensional universal types. In this section we consider separated sums and linearly initial algebras. For space reason, we give only a sketch.

6.1 Basic Ideas

First we describe basic ideas used in later, and also why we use separated sums and linearly initial algebras instead of coproducts and initial algebras. Contrary to the case of constants, for coproducts and initial algebras we have to use more limited class of models with additional arrows. First let us see the reason for coproducts in a syntactic way.

Naively thinking, to get the desired term of the type $\forall\alpha.\,(\sigma+\tau)\to\forall\alpha.\sigma+\forall\alpha.\tau$, we can think of a term like

$$\begin{aligned}&\lambda u{:}\forall\alpha.\,(\sigma+\tau)\,.\ \text{case } u1 \text{ of} \qquad (2)\\ &\quad \text{in}_0\ a'\to \text{in}_0\ \big(\Lambda\alpha.\text{case } u\alpha \text{ of } (\text{in}_0\ a\to a)\ \mid\ (\text{in}_1\ b\to \text{“this case nothing”})\big)\\ &\quad \mid\text{in}_1\ b'\to \text{in}_1\ \big(\Lambda\alpha.\text{case } u\alpha \text{ of } (\text{in}_0\ a\to \text{“this case nothing”})\ \mid\ (\text{in}_1\ b\to b)\big)\end{aligned}$$

where “this case nothing”’s mean that the cases of the coproducts are not realized, if we assume parametricity. In fact, we can prove in the Plotkin-Abadi logic that, for any term u of a type $\forall\alpha.\,(\sigma+\tau)$, every type instantiation of u has the same index of the coproduct.

However, this is just a reasoning in logic, and there is no assurance to be able to construct such terms as “this case nothing”. In fact, there is no term of the type $\forall\alpha.\,(\sigma+\tau)\to\forall\alpha.\sigma+\forall\alpha.\tau$ in System F for certain σ and τ: for the case when σ is α and τ is $\alpha{\to}0$, the type $\forall\alpha.\,(\alpha+(\alpha{\to}0))\to\forall\alpha.\alpha+\forall\alpha.\,(\alpha{\to}0)$ in System F corresponds to the proposition $\forall\alpha.\,(\alpha\vee\neg\alpha)\Rightarrow\forall\alpha.\alpha\vee\forall\alpha.\neg\alpha$ in second order intuitionistic logic, and the inhabitation contradicts the soundness of second order classical logic. So we have to add more terms to realize the above “this case nothing”.

To solve this problem, we use non-termination effects, with which we can replace “this case nothing”’s with bottoms.

In the next place, let us think about initial algebras. For e.g. the list monad, we may think a desired term f of the type $\forall\alpha.\mu\beta.1+\sigma\times\beta\longrightarrow\mu\beta.1+(\forall\alpha.\sigma)\times\beta$ as the following.

$$\begin{aligned}&\text{let } f=\lambda u{:}\,(\forall\alpha.\mu\beta.1{+}\sigma{\times}\beta)\,.\ \text{case } u1 \text{ of Nil}\to\text{Nil} \mid \text{Cons}(a',as')\to\\ &\quad \text{Cons}\big(\ \Lambda\alpha.\text{case } u\alpha \text{ of } (\text{Nil}\to\text{“this case nothing”})\ \mid\ (\text{Cons}(a,as)\to a)\\ &\qquad ,\ f\big(\Lambda\alpha.\text{case } u\alpha \text{ of } (\text{Nil}\to\text{“this case nothing”})\ \mid\ (\text{Cons}(a,as)\to as)\big)\ \big)\end{aligned}$$

The two “this case nothing”’s are the same as that in the case of coproducts, and this is just because we use coproducts in the definition of lists. The essential here is the occurrence of f in the definition of f. This is not induction which follows from the universality of initial algebras, but recursion by fixed-point operators.

For fixed-point operators, we employ separated sums and linearly initial algebras instead of coproducts and initial algebras respectively. These cause no problem to construct monads, as we use these in Haskell in fact, because these also have universal property, though limited into a linear model. In the following subsections, we show that separated sums and linearly initial algebras preserve simple Ω-products in relevant parametric models with fixed-point operators.

6.2 Linear Parametric Models

From now on, we consider domain theoretic models. Let us begin with describing what kind of models we use.

We consider a PILL_Y *model*[6] l having fibred products. PILL_Y models are $\lambda 2$-version of linear categories which can also model fixed-point operators, see loc. cit. for details. The requirement on fibred products is not strong at all, since they can be obtained for free if we assume linear parametricity [5], and in fact we assume a bit stronger one, i.e., relevant parametricity in the next subsection. For the linear exponential fibred comonad ! on l, let $U : l \rightleftarrows p : L$ be its coKleisli fibred adjunction, where U is the right, so fiberwise identity-on-objects.

Then p is a $\lambda 2$-fibration: It is well-known that the coKleisli category of a linear model with cartesian products is a CCC, see e.g. [3]. A generic object is shared with l, since U is fiberwise identity-on-objects and the base is shared. Simple Ω-products are for free like fibred products.

We take this $\lambda 2$-fibration p as the base fibration of monadic $\lambda_{\mathrm{c}} 2_{\eta}$-models. Then, the fibred monads T on p studied below are what we described in Section 5.1.

Since U is a fibred right adjoint functor, it preserves simple Ω-products, which follow from just the universal property, and irrelevant to the fact that Ω is a generic object or that U preserves it.

On the other hand, we do assume that the left fibred adjoint functor L preserves simple Ω-products. The reason for assuming this is to show the compatibility of separated sums and of linearly initial algebras with simple Ω-products. Thanks to this assumption, we can add the non-termination lifting monad to the class of monads compatible with simple Ω-products, but this is just a (welcome) secondary product. This is a reasonable assumption, since lifting monads usually preserve simple Ω-products in a parametric setting, as opposed to powerset monads.

6.3 Relevant Parametricity for Separated Sum

The use of fixed-point operators involves two matters, i.e., we have to use separated sums instead of coproducts in p, and have to use linear parametricity.

In fact, the use of separated sums is less problematic. Assuming linear parametricity, there are fibred coproducts in l, so we have also fibred separated sums $\oplus$ in p as $U \circ (+) \circ L^2$. Here, $(+) \circ L^2$ has the same kind of universal property as Kleisli exponentials, and by this we can construct familiar monads like e.g.

exception monads, the list monad, and input monads. When we use separated sums for exception monads, it can be viewed also in terms of *linearly used effects* [13].

Now we show that these separated sums commute with simple Ω-products. We basically use the idea of the above term (2). After replacing "this case nothing"'s with bottoms, still there remain two problems. The first is, in (2), we use weakening rules for a' and b', and the second is the use of a contraction rule for the two u's in $u1$ and $u\alpha$. (The two $u\alpha$'s are essentially the same, since this is from the distributivity of monoidal products over coproducts in l.)

The problem of weakening is resolved as the following. To prove that separated sums preserve simple Ω-products, it suffices to prove $\prod^{\circ}(!A+!B) \cong \prod^{\circ}!A+\prod^{\circ}!B$ in l, where $\prod^{\circ}$ is simple Ω-products in l. This is because, we can then show that $\prod(A \oplus B) \stackrel{\text{def}}{=} \prod U(LA+LB) \cong U\prod^{\circ}(LA+LB) = U\prod^{\circ}(!A+!B) \cong U\left(\prod^{\circ}!A+\prod^{\circ}!B\right) \cong U(L\prod UA+L\prod UB) = \prod A \oplus \prod B$, noting that U and L preserve simple Ω-products, and U is identity-on-objects. So, we can use weakening rules by virtue of these two !'s.

On the other hand, to use a contraction rule, we need to strengthen a type theory to allow contraction, i.e., to the "relevant" one. Fortunately, lifting monads are usually *relevant monads* as in [18]. *Relevant lambda calculi* are extension of linear lambda calculi with contraction term formation rules. Then *relevant Plotkin-Abadi Logic* is simply linear Plotkin-Abadi logic [5] on top of such the second order relevant lambda calculus. As a result of this extension of the class of linear terms, the class of *admissible relations* is also extended, for instance we can use graph relations of such "relevant terms".

Finally, we need one more rule for forming admissible relations: for an admissible relation ρ between σ and τ, and a term $\langle f, g\rangle : \sigma' \times \tau' \multimap \sigma \times \tau$ of linear function type, the "inverse-image" relation $(x : \sigma', y : \tau').\rho(f\langle x, y\rangle, g\langle x, y\rangle)$ is also an admissible relation between σ' and τ'. This is modeled with the intuition that admissible relations between σ and τ are subalgebras of the product of σ and τ. This rule is used in the parametric reasoning to prove the equality between the identity on $\prod^{\circ}(!A+!B)$ and the composite term through $\prod^{\circ}!A+\prod^{\circ}!B$ involving the isomorphism $\prod^{\circ}(!A+!B) \cong \prod^{\circ}!A+\prod^{\circ}!B$.

We introduce some of models (l, p, ...) for such relevant parametricity which satisfy the assumptions thus far: the models (PFam(AP$(D)_{\perp}$), PFam(AP(D)), ...) of linear Plotkin-Abadi logic constructed from domain theoretic PERs, which is described in [8]. Hence,

Proposition 10 *The separated sums in PFam$(AP(D))$ preserve simple Ω-products.*□

6.4 Linearly Initial Algebras

In this last subsection, we consider about linearly initial algebras.

Once we determine to use fixed-point operators, there is an easier way than considering of such a complicate term as in Section 6.1. That is, we can use the fact that if both initial algebras and final coalgebras exist for sufficiently many endofunctors, then the initial algebras and final coalgebras are canonically

isomorphic for such endofunctors if and only if fixed-point operators exists [9,5]. Then Lemma 9 is applicable.

All the remaining matter is that we have to use linearly initial algebras instead of initial algebras. Now let q be a fibration which has the same base category and generic object Ω as those of l and p, and has simple Ω-products. Then for a fibred functor $F : q \times p \longrightarrow p$ preserving simple Ω-products, the composite fibred functor $F' := L \circ F \circ (q \times U) : q \times l \longrightarrow q \times p \longrightarrow p \longrightarrow l$ also preserves simple Ω-products.

Now if F' has initial algebras with parameters and also final coalgebras with parameters in the sense of Section 5.2, then the $\mu F'$ and $\nu F'$ are naturally isomorphic thanks to the fixed-point operators as mentioned above. So by Lemma 9, $\mu F'$ preserves simple Ω-products. Hence the *linearly initial algebras* fibred functor $U \circ \mu F' : q \longrightarrow l \longrightarrow p$ also preserves simple Ω-products.

Proposition 11 *Let $F : PFam(AP(D))^{n+1} \longrightarrow PFam(AP(D))$ be a fibred functor such that F' defined as above has initial algebras with parameters and final coalgebras with parameters. If F preserves simple Ω-products, then the linearly initial algebras fibred functor $U \circ \mu F' : PFam(AP(D))^{n} \longrightarrow PFam(AP(D))$ also preserves simple Ω-products.* □

Theorem 12 *Let T be a fibred strong monad on $PFam(AP(D))$ satisfying the equalizing requirement. If the underlying fibred endofunctor of T is included in the class (1) in Section 5.1, then T and the $\lambda 2$-fibration $PFam(AP(D))$ form a monadic $\lambda_c 2_\eta$-model.* □

7 Concluding Remark

We have given the second order computational λ-calculus with extensional universal types, which is a call-by-value lambda calculus with universal types satisfying η-equality. Then we have formulated its sound and complete categorical semantics, and also reasonable characterization in terms of monadic metalanguages. Finally, we have seen concrete domain theoretic models, in a somewhat general way with relevant parametricity. Such models can accommodate many familiar effects constructed polynomially to extensional universal types.

In Section 6, we have taken the domain theoretic approach. On the other hand, we can also take a dependent type theoretic approach, by which we can avoid "this case nothing" in the term (2) in Section 6, using *strong* coproducts. Moreover, by *inductive* initial algebras in the sense of [19], we can deal with some kinds of initial algebras including the list monad and input monads. In this way, we can see that the typical models for parametricity constructed from recursion theoretic PERs (see e.g. loc. cit.) also have a similar class of monads which form $\lambda_c 2_\eta$-models. These will be treated in a separate paper.

It is interesting to clarify whether we can include powerset monads and/or continuations monads into the class of models, and whether all lifting monads commute with parametric simple Ω-products.

Also, it is an interesting challenge to investigate principles of parametric polymorphism in the two call-by-value calculi in the paper.

Acknowledgements

I would like to thank Masahito Hasegawa for many helpful discussions and comments on earlier drafts. I am also grateful to Shin-ya Katsumata, Ichiro Hasuo, and Naohiko Hoshino for useful comments and discussions. Also I thank anonymous reviewers for helpful comments.

References

1. Martín Abadi, Luca Cardelli, and Pierre-Louis Curien. Formal parametric polymorphism. *TCS: Theoretical Computer Science*, 121, 1993.
2. Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In Gilles Barthe, Peter Dybjer, Luís Pinto, and João Saraiva, editors, *Advanced Lectures from Int. Summer School on Applied Semantics, APPSEM 2000 (Caminha, Portugal, 9–15 Sept. 2000)*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer-Verlag, Berlin, 2002.
3. P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In *CSL*, pages 121–135, 1994.
4. Gavin M. Bierman, Andrew M. Pitts, and Claudio V. Russo. Operational properties of lily, a polymorphic linear lambda calculus with recursion. *Electr. Notes Theor. Comput. Sci.*, 41(3), 2000.
5. Lars Birkedal, Rasmus E. Møgelberg, and Rasmus L. Petersen. Linear Abadi & Plotkin logic. *Logical Methods in Computer Science*, 2(5), November 2006.
6. Lars Birkedal, Rasmus E. Møgelberg, and Rasmus L. Petersen. Category-theoretic models of linear Abadi and Plotkin logic. *Theory and Applications of Categories*, 20(7):116–151, 2008.
7. Lars Birkedal and Rasmus Ejlers Møgelberg. Categorical models for Abadi and Plotkin's logic for parametricity. *Mathematical Structures in Computer Science*, 15(4):709–772, 2005.
8. Lars Birkedal, Rasmus Ejlers Møgelberg, and Rasmus Lerchedahl Petersen. Domain-theoretical models of parametric polymorphism. *Theor. Comput. Sci*, 388(1-3):152–172, 2007.
9. Peter Freyd. Recursive types reduced to inductive types. In John Mitchell, editor, *Proceedings of the Fifth Annual IEEE Symp. on Logic in Computer Science, LICS 1990*, pages 498–507. IEEE Computer Society Press, June 1990.
10. Carsten Führmann. Direct models of the computational lambda calculus. *Electr. Notes Theor. Comput. Sci*, 20, 1999.
11. Jean-Yves Girard. Interpretation fonctionelle et elimination des coupures de l'arithmetique d'ordre superieur. These D'Etat, Universite Paris VII, 1972.
12. Robert Harper and Mark Lillibridge. Operational interpretations of an extension of F_ω with control operators. *J. Funct. Program.*, 6(3):393–417, 1996.
13. Masahito Hasegawa. Linearly used effects: Monadic and cps transformations into the linear lambda calculus. In Zhenjiang Hu and Mario Rodríguez-Artalejo, editors, *FLOPS*, volume 2441 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2002.
14. Masahito Hasegawa. Relational parametricity and control. *Logical Methods in Computer Science*, 2(3), 2006.
15. Masahito Hasegawa and Yoshihiko Kakutani. Axioms for recursion in call-by-value. *Higher-Order and Symbolic Computation*, 15(2-3):235–264, 2002.

16. Ryu Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 4(1):71–109, 1994.
17. Claudio Alberto Hermida. *Fibrations, Logical Predicates and Indeterminates*. PhD thesis, University of Edinburgh, 1993.
18. Bart Jacobs. Semantics of weakening and contraction. *Ann. Pure Appl. Logic*, 69(1):73–106, 1994.
19. Bart Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics 141. Elsevier, 1999.
20. Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.
21. Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *INFCTRL: Information and Computation (formerly Information and Control)*, 185, 2003.
22. Rasmus Ejlers Møgelberg. Interpreting polymorphic fpc into domain theoretic models of parametric polymorphism. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 372–383. Springer, 2006.
23. Rasmus Ejlers Møgelberg and Alex Simpson. Relational parametricity for computational effects. In *LICS*, pages 346–355. IEEE Computer Society, 2007.
24. Rasmus Ejlers Møgelberg and Alex Simpson. Relational parametricity for control considered as a computational effect. *Electr. Notes Theor. Comput. Sci.*, 173:295–312, 2007.
25. Eugenio Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Laboratory for Foundations of Computer Science, University of Edinburgh, 1988.
26. Eugenio Moggi. Computational lambda-calculus and monads. In *LICS*, pages 14–23. IEEE Computer Society, 1989.
27. Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
28. Gordon D. Plotkin. Type theory and recursion (extended abstract). In *LICS*, page 374. IEEE Computer Society, 1993.
29. Gordon D. Plotkin and Martín Abadi. A logic for parametric polymorphism. In Marc Bezem and Jan Friso Groote, editors, *TLCA*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 1993.
30. John C. Reynolds. Towards a theory of type structure. In Bernard Robinet, editor, *Symposium on Programming*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974.
31. John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523, 1983.
32. Alex K. Simpson and Gordon D. Plotkin. Complete axioms for categorical fixed-point operators. In *LICS*, pages 30–41, 2000.
33. Ross Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2:149–168, 1972.
34. Philip Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*. Springer Verlag, 1989.

Erratum

In Section 4 we defined two kinds of categorical models called $\lambda_c 2_\eta$-*models* and *monadic* $\lambda_c 2_\eta$-*models*, where the latter is stronger notion than the former; and in Theorem 4 we stated that the $\lambda_{\mathrm{c}} 2_\eta$-calculs is sound and complete for the $\lambda_{\mathrm{c}} 2_\eta$-models. It turns out that this does not hold, and instead we have to assume *monadic* $\lambda_c 2_\eta$-*models* rather than $\lambda_{\mathrm{c}} 2_\eta$-models; this is necessary in the sense that the calculus is also complete for the monadic $\lambda_{\mathrm{c}} 2_\eta$-models, as the term model becomes a monadic $\lambda_{\mathrm{c}} 2_\eta$-model. Thus a correct theorem instead of Theorem 4 is:

Theorem *The* $\lambda_c 2_\eta$*-calculus is sound and complete with respect to the monadic* $\lambda_c 2_\eta$*-models.* □

What is overlooked in a proof of the soundness is the lemma that, for any value V, its interpretation $[\![V]\!]$ is a value (i.e., in the image of the Kleisli embedding); its proof itself is straightforward by induction on V (once we assume monadic $\lambda_{\mathrm{c}} 2_\eta$-models).

All the parts except for Theorem 4 in the paper are correct—where first of all we do not use the notion of $\lambda_{\mathrm{c}} 2_\eta$-models (but use the stronger notion of monadic $\lambda_{\mathrm{c}} 2_\eta$-models)—, especially including Sections 5 and 6 on construction of concrete monadic $\lambda_{\mathrm{c}} 2_\eta$-models.

Remark *Once we apply the above erratum, we use only the notion of monadic* $\lambda_c 2_\eta$*-models and do not use that of* $\lambda_c 2_\eta$*-models at all; then it seems better to use the terminology* $\lambda_{\mathrm{c}} 2_\eta$-models *for the notion of monadic* $\lambda_c 2_\eta$*-models, (though in this erratum we always use the terminology in the original paper to avoid confusion).* □