Universal Properties of Impure Programming Languages

Sam Staton
Computer Laboratory, University of Cambridge

Paul Blain Levy
School of Computer Science, University of Birmingham

Abstract

We investigate impure, call-by-value programming languages. Our first language only has variables and let-binding. Its equational theory is a variant of Lambek's theory of multicategories that omits the commutativity axiom.

We demonstrate that type constructions for impure languages — products, sums and functions — can be characterized by universal properties in the setting of 'premulticategories', multicategories where the commutativity law may fail. This leads us to new, universal characterizations of two earlier equational theories of impure programming languages: the premonoidal categories of Power and Robinson, and the monad-based models of Moggi. Our analysis thus puts these earlier abstract ideas on a canonical foundation, bringing them to a new, syntactic level.

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]

Keywords Multicategories, Premonoidal categories, Monads

1. Introduction

In this paper we introduce a new equational account of impure programs. This account is analogous to the $\beta\eta$-equality for the pure typed λ-calculus. It provides a simple reasoning tool that is justified both syntactically and semantically, by compositional interpretations in a variety of models. It is given a canonical status through universal properties and representability.

Let us stress that the $\beta\eta$-equality of the pure λ-calculus is not appropriate in the impure setting under the call-by-value semantics. For instance, if M has side effects then the η-law for products does not hold: $M \not\equiv \langle \#1\, M, \#2\, M\rangle$. For this reason, the categorical notion of product does not immediately apply to call-by-value programming languages.

1.1 An equational account of let-binding: premulticategories

To move to the impure setting, we must reassess the nature of substitution, which we do by introducing the new notion of *premulticategory* (§2). In the $\beta\eta$-equality of a pure λ-calculus, substitution is essential (e.g. $(\lambda x.M)\, N \equiv M[{}^{N}/_{x}]$). In a call-by-value language, substitution is more intricate. The Standard ML expression

$$\texttt{let val}\ x = N\ \texttt{in}\ M\ \texttt{end}$$

is (in some sense) equivalent to $M[{}^{N}/_{x}]$ in the pure fragment of ML, whereas in the general it means 'evaluate N, call the result x, and continue as M'. This sequencing plays a fundamental role, and it forms the basis of our definition of premulticategory.

Our starting point is a language with at least two typing rules

$$\frac{-}{x{:}A \vdash x : A} \qquad \frac{\Delta \vdash t : A \qquad \Gamma, x{:}A, \Gamma' \vdash u : B}{\Gamma, \Delta, \Gamma' \vdash \texttt{let val}\ x = t\ \texttt{in}\ u\ \texttt{end} : B}$$

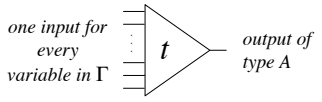
and satisfying three equations

$$\texttt{let val}\ x{=}x\ \texttt{in}\ t\ \texttt{end} \equiv t \qquad \texttt{let val}\ x{=}t\ \texttt{in}\ x\ \texttt{end} \equiv t\ (x \notin \mathrm{fv}(t))$$

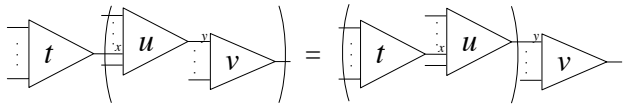
$$\begin{aligned}&\texttt{let val}\ x = t\ \texttt{in let val}\ y = u\ \texttt{in}\ v\ \texttt{end end}\\ \equiv\ &\texttt{let val}\ y = (\texttt{let val}\ x = t\ \texttt{in}\ u\ \texttt{end})\ \texttt{in}\ v\ \texttt{end} \quad (x \notin \mathrm{fv}(v)).\end{aligned}$$

Any reasonable call-by-value programming language will have an analogue of `let val`. Roughly speaking it is the essence of A-Normal Form and Single-Static-Assignment (e.g. [10]).

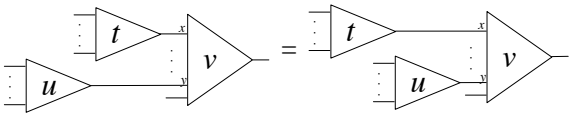
It is often helpful to understand a term in context $\Gamma \vdash t : A$ as a generalized function which takes a valuation for the context Γ and returns a result of type A. If we thus draw a term graphically



then we can understand `let val` as composition of functions, and a single variable is just a wire. The third law says



The three laws are thus the laws of identity and associativity for composing multivariate functions. The classical theory of multicategories also proposes a commutativity law:



$$\begin{aligned}&\texttt{let val}\ x = t\ \texttt{in let val}\ y = u\ \texttt{in}\ v\ \texttt{end end} \quad (x \notin \mathrm{fv}(u),\\ \equiv\ &\texttt{let val}\ y = u\ \texttt{in let val}\ x = t\ \texttt{in}\ v\ \texttt{end end} \qquad y \notin \mathrm{fv}(t))\end{aligned}$$

This law does not hold for impure programs: consider the case where $t \stackrel{\text{def}}{=} \texttt{print "hello "}$ and $u \stackrel{\text{def}}{=} \texttt{print "world"}$. Our equational theory of `let val` is thus a non-commutative variant of multicategories, which we call 'premulticategories' (§2).

(Jeffrey [18] resolves the inaccuracy of the graphical calculus by introducing special control arrows.)

POPL'13, January 23–25, 2013, Rome, Italy.

1.2 First class types via representability

We argue that type and term constructions should be defined in the context of the simple language with `let val`, in a canonical and universal way. To this end, we give universal properties for products (§3), sums (§4) and call-by-value functions (§6) in impure languages. One important consequence of universal properties is that although there may be many isomorphic implementations of (say) a product type, there is exactly one isomorphism between any two implementations, which means that there is a canonical way to translate between them.

We focus on product types for now, but we will later deal with sum types and function types. The concept of 'product' is already present in the little language with `let val`, for we can describe terms '$\Gamma, x{:}\,(A \times B) \vdash t : C$' as terms $\Gamma, x_1{:}\,A, x_2{:}\,B \vdash t' : C$. The principle of representability says that this concept determines the product types, if they exist. Concretely, representability requires that there is a term

$$x{:}\,A, y{:}\,B \vdash \langle x, y\rangle : A \times B$$

which is universal in the following sense. The term induces a family of functions between sets:

$$P_{\Gamma,C} : \{t \mid \Gamma, z{:}\,A \times B \vdash t : C\} \to \{u \mid \Gamma, x{:}\,A, y{:}\,B \vdash u : C\}$$

where $P_{\Gamma,C}(t) \stackrel{\text{def}}{=} \texttt{let val}\, z \texttt{ = } \langle x, y\rangle \,\texttt{in}\, t \,\texttt{end}$, and this family of functions is required to be a natural family of bijections. The bijectivity condition means that each $P_{\Gamma,C}$ has an inverse, in other words that there is an assignment

$$\frac{\Gamma, x{:}\,A, y{:}\,B \vdash u : C}{\Gamma, z{:}\,A \times B \vdash P^{-1}_{\Gamma,C}(u) : C}$$

satisfying some equations. In Standard ML, we can define $P^{-1}_{\Gamma,C}$ by pattern matching: let $P^{-1}_{\Gamma,C}(u) \stackrel{\text{def}}{=} \texttt{let val}\, \langle x, y\rangle \texttt{ = } z \,\texttt{in}\, u \,\texttt{end}$.

The technique of representability is syntactic, in that it suggests constructors and destructors for product types. It is also semantic, because it only determines the types up-to unique isomorphism of types. It gives a universal property: there is a unique isomorphism between any two implementations.

The principle of representability is a fundamental idea in abstract mathematics. Hermida [14] gives an account of representability for multicategories. One of our main technical contributions is to develop the theory of representability in the non-commutative setting of premulticategories (§7) and to demonstrate its relevance to the principles of programming languages (§3–6).

1.3 Explaining monads and premonoidal categories

Two important precursors to this work are Moggi's work on monads [29, 30] and the line of work on premonoidal categories and Freyd categories [12, 33, 34] begun by Power and Robinson. This earlier work has been very influential in programming languages research. Our criticism of it is that its analysis of impure computation is inextricably tied with its treatment of types. Using premonoidal categories, we show that impure computation can be separated from types in these models (§8).

Moggi's notion of λ_C-model [29, Def. 2.6] is defined in terms of strong monads. It axiomatizes in one fell swoop the structure of impure programming languages with product types and function types.

One of the contributions of Power and Robinson's work [33] is to decouple the function types from Moggi's notion of λ_C-model. They do this by directly axiomatizing categorical structures of 'computations' which they call premonoidal categories. This is in contrast to Moggi's development which centres around monads on a category of 'values'. The notion of premonoidal category axiomatizes (again, in one fell swoop) the structure of impure languages with product types. Freyd categories adjoin the distinction between computations and values to the theory of premonoidal categories.

In this paper we isolate the principles of impure computation from type structure by introducing the notion of premulticategory and the related notion of Freyd multicategory, a premulticategory with a distinguished class of values. Our main theorems recover the earlier constructions by using universal properties to introduce type structure to this framework:

- A premonoidal category [33] is a premulticategory with products (Theorem 26);
- A Freyd category [33, 34] is a Freyd multicategory with products (Proposition 33).
- A λ_C-model [29] is a Freyd multicategory with products and function spaces (§8.3).

Coherence from representability. We now give a more technical summary of the connections between premonoidal categories and premulticategories. In a programming language, the types

$$(A * B) * C \qquad \text{and} \qquad A * (B * C) \tag{1}$$

are not identical but they are isomorphic in a canonical way. In the approach of Power and Robinson, the isomorphism between the two types (1) is *given* as part of the data for a premonoidal category [33]. This 'coherence' approach has proved extremely successful in many areas of mathematics, but it appears several steps away from programming language syntax. In our work, the isomorphism (1) does not need to be given: instead, it can be *derived* from the universal property of products. This is a non-commutative version of Hermida's result [14]: to give a monoidal category is to give a multicategory with tensor products.

The situation is actually more complicated. Power and Robinson require that the given isomorphism (1) is 'central', which informally means that it doesn't matter when it is executed. Although this requirement is reasonable from a pragmatic point of view, it does not seem to arise from a requirement of higher category theory: premonoidal categories do not seem to be pseudomonoids in a monoidal bicategory. This problem is remedied by our representability result, which provides a principled explanation of the centrality requirements in premonoidal categories.

In summary, whereas the original definition of premonoidal category is several steps away from programming language syntax, our new characterization essentially *is* programming language syntax. This is because rather than axiomatizing everything in one fell swoop, we characterize type constructions as universal properties over a basic framework for impure computation.

2. A basic equational account of impure programs: premulticategories

We begin by analyzing the basic structure of an impure programming language with variables and let-binding. The equations which we propose are three of the four equations of multicategories.

Recall that an ordinary category comprises a collection of objects, a collection of morphisms between the objects, a selection of identity morphisms, and an associative composition operation. Roughly speaking, a premulticategory is similar except that the morphisms do not go from one object to another. Rather, they go from a context Γ to an object A. This is a natural way to study typed programming languages, since a program in a context Γ has a type A.

At this stage, we do not ask for the objects to be closed under operations (product types, unit types) nor for any special morphisms (printing, arithmetic). Nonetheless we certainly do not forbid these

things at this stage: a premulticategory can have more content than the basic structural requirements.

As well as the syntactic examples of premulticategories that arise from programming languages, we have more concrete examples coming from set theory, pointed sets, and we explain how a strong monad gives rise to a premulticategory.

2.1 Definitions: premulticategories and centrality

Notational convention. Informally, the language of premulticategories can be thought of as a language for terms in context. Formally, a morphism $\Gamma \to A$ goes from a list Γ of objects to an object. We introduce the following informal convention. We will often write a list (A, B, C) annotated with variables $(x : A, y : B, z : C)$. This allows us to informally index the list using variables rather than numbers, e.g. we can write y for the second element of the list. By doing this we can reason about concatenated lists without fumbling with arithmetic on indices. In everything we do, the informal variables can be translated into numbers at the expense of readability.

The cornerstone of this paper is the following definition of premulticategory. This is a non-commutative version of 'multicategory', an idea first extensively investigated by Lambek [21].

Definition 1. A *premulticategory* is given by the following data.

- A collection of objects, ranged over by A, B etc.
- For each list of objects Γ and each object A a collection of morphisms $\Gamma \to A$ is given. If $t : \Gamma \to A$ then we write $\Gamma \vdash t : A$.
- *Identity morphisms:* For each object A a morphism $\mathrm{id}_A : (A) \to A$ must be given. We notate this requirement informally as a rule:

$$\frac{}{x{:}\,A \vdash x : A}$$

- *Composition:* Given a morphism $t : \Delta \to A$ and morphism $u : (\Gamma, A, \Gamma') \to B$, if the length of Γ is i then a morphism $(u \circ_i t) : (\Gamma, \Delta, \Gamma') \to B$ must be given. We notate this requirement informally as a rule:

$$\frac{\Delta \vdash t : A \qquad \Gamma, x{:}\,A, \Gamma' \vdash u : B}{\Gamma, \Delta, \Gamma' \vdash t \rightsquigarrow x.\, u : B}$$

  The notation $(t \rightsquigarrow x.\, u)$ can be understood as shorthand for $\texttt{let val}\ x\ \texttt{=}\ t\ \texttt{in}\ u\ \texttt{end}$: "execute t, bind the result to x, and continue as u".

The data is subject to the following equations.

- Identity laws.

$$\frac{\Gamma, x{:}\,A, \Gamma' \vdash t : B}{\Gamma, x{:}\,A, \Gamma' \vdash x \rightsquigarrow x.\, t \equiv t : B} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash t \rightsquigarrow x.\, x \equiv t : A}$$

  In case the notation is unclear: the first law means that for any morphism $t : (\Gamma, A, \Gamma') \to B$, the composite of id_A with t at position i (where i is the length of Γ) is equal to t.
- Associativity law:

$$\frac{\Gamma_3 \vdash t : A \qquad \Gamma_2, x{:}\,A, \Gamma_2' \vdash u : B \qquad \Gamma_1, y{:}\,B, \Gamma_1' \vdash v : C}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_2', \Gamma_1' \vdash (t \rightsquigarrow x.\, u) \rightsquigarrow y.\, v \equiv t \rightsquigarrow x.\, (u \rightsquigarrow y.\, v) : C}$$

  More formally this law is notated $v \circ_i (u \circ_j t) = (v \circ_i u) \circ_{i+j} t$, where i is the length of Γ_1 and j is the length of Γ_2.

Commutativity and centrality. In general, given morphisms $t : \Gamma_1 \to A$ and $u : \Gamma_2 \to B$, we say that t *commutes with* u if it doesn't matter which order t and u are executed. To be precise, t commutes with u if the following equations hold (for all v):

$$\frac{\Delta_1, x{:}\,A, \Delta_2, y{:}\,B, \Delta_3 \vdash v : C}{\Delta_1, \Gamma_1, \Delta_2, \Gamma_2, \Delta_3 \vdash t \rightsquigarrow x.\, u \rightsquigarrow y.\, v \equiv u \rightsquigarrow y.\, t \rightsquigarrow x.\, v : C}$$

$$\frac{\Delta_1, y{:}\,B, \Delta_2, x{:}\,A, \Delta_3 \vdash v : C}{\Delta_1, \Gamma_2, \Delta_2, \Gamma_1, \Delta_3 \vdash t \rightsquigarrow x.\, u \rightsquigarrow y.\, v \equiv u \rightsquigarrow y.\, t \rightsquigarrow x.\, v : C}$$

If t commutes with all u, then we say that t is *central*. In other words, t is central if it doesn't matter when it is executed. A premulticategory is a *multicategory* if all morphisms are central. (Lambek [21] calls this the axiom of commutativity.)

2.2 Examples of premulticategories

Multicategories have attracted significant recent interest in category theory, logic and algebra (e.g. [7, 14, 15, 22]), and in the semantics of continuation-passing [27] and of state [26, 35]. Premulticategories have not been studied before, to our knowledge.

First simple example: multicategory of sets. We begin by describing the multicategory of sets, but let us first set up some notation. Given a list Γ of sets, we write $\prod\Gamma$ for the product of those sets. For instance, $\prod(A, B, C) = \{(a, b, c) \mid a \in A, b \in B, c \in C\}$. If we view Γ as a context, then $\prod\Gamma$ is the set of valuations for that context.

In the multicategory of sets, the objects are sets, and a morphism $\Gamma \to A$ is a function $\prod\Gamma \to A$. Thus a morphism $\Gamma \to A$ is something which assigns a result in A to every valuation of Γ. The identity morphisms are the identity functions. Composition is by composition of functions: given morphisms $t : \Delta \to A$ and $u : (\Gamma, x{:}\,A, \Gamma') \to B$ and given valuations $\vec{a} \in \prod\Gamma$, $\vec{b} \in \prod\Delta$, $\vec{c} \in \prod\Gamma'$, let $(t \rightsquigarrow x.\, u)(\vec{a}, \vec{b}, \vec{c}) \stackrel{\text{def}}{=} u(\vec{a}, t(\vec{b}), \vec{c})$. This is a multicategory: all morphisms are central.

Second simple example: multicategory of pointed sets. In the multicategory of pointed sets, the objects are sets A equipped with a distinguished element $\bot_A$. A morphism $t : \Gamma \to A$ is a function $t : \prod\Gamma \to A$ that is strict in each argument: if $\vec{a} \in \prod\Gamma$ is such that $a_i = \bot$ for some i, then $t(\vec{a}) = \bot_A$. Identities and composition are as in the multicategory of sets; it is easy to check that these yield functions that are strict in each argument. All morphisms are central.

Third simple example: premulticategory of 'stateful' functions. Let S be a fixed set, thought of as a set of states. We form a premulticategory whose objects are sets and where a morphism $\Gamma \to A$ is a function $(\prod\Gamma) \times S \to A \times S$. The idea is that a 'stateful' function takes as an argument a valuation of its context and also an initial state; it returns a result and also a final state.

Composition is defined as follows. Given $t : \Delta \to A$ and $u : (\Gamma, x{:}\,A, \Gamma') \to B$ and given valuations $\vec{a} \in \prod\Gamma$, $\vec{b} \in \prod\Delta$, $\vec{c} \in \prod\Gamma'$ and an element $s \in S$, let $(t \rightsquigarrow x.\, u)(\vec{a}, \vec{b}, \vec{c}, s) \stackrel{\text{def}}{=} u(\vec{a}, a, \vec{c}, s')$ where $(a, s') = t(\vec{b}, s)$. This premulticategory is not a multicategory unless S has at most one element. The central morphisms $t : \Gamma \to A$ are the functions that do not change the state, i.e. for each valuation $\vec{a} \in \prod\Gamma$ there is an element $t_1(\vec{a})$ of A such that for all states $s \in S$ we have $t(\vec{a}, s) = (t_1(\vec{a}), s)$.

Connection with programming languages. Well-typed terms of a programming language such as Standard ML form a premulticategory, modulo observational equivalence. Composition $(t \rightsquigarrow x.\, u)$ amounts to the ML syntax $\texttt{let val}\ x\ \texttt{=}\ t\ \texttt{in}\ u\ \texttt{end}$. The identity and associativity laws are simple observational equations under this interpretation. Not all morphisms are central. For instance

$$\mathsf{print}\ \texttt{"hello"} \rightsquigarrow x.\, \mathsf{print}\ \texttt{" world"} \rightsquigarrow y.\, (x, y) \not\equiv \mathsf{print}\ \texttt{"world"} \rightsquigarrow x.\, \mathsf{print}\ \texttt{" hello"} \rightsquigarrow y.\, (x, y).$$

Connection with deductive systems. As Lambek observed [21], multicategories are closely related to deductive systems such as the sequent calculus. The identity morphisms correspond to the axiom, and composition corresponds to cut. The identity and associativity laws provide a starting point for a cut-elimination procedure. However, whereas the commutativity axiom plays a natural role in equivalence of proofs, we are rejecting it in the context of programming languages. Moreover, we are not interested in cut elimination, since cut/sequencing plays a crucial role. For instance, we cannot eliminate sequencing from the following program without changing its meaning.

$(\mathsf{print}\ \text{"Enter your name:"}) \rightsquigarrow x.\, \mathsf{readline} \rightsquigarrow y.\, \mathsf{print}\ \text{"Hello "}\hat{}\, y$

Connection with algebra. The words 'commutativity' and 'centrality' are used by analogy with algebra. Let $(M, \cdot, e)$ be a monoid, i.e. a set with an associative binary operation $\cdot$ and unit e. We can build a premulticategory from M: it has one object $*$ and a morphism $\Gamma \to *$ for each element of M. The identity morphism is e and composition $m \rightsquigarrow x.\, n$ is monoid multiplication, $m \cdot n$.

A morphism m in this premulticategory is central if and only if it is central in the usual algebraic sense: $\forall n.\, m \cdot n = n \cdot m$. The premulticategory is a multicategory if and only if the monoid is commutative: $\forall m, n.\, m \cdot n = n \cdot m$.

This is a simple motivating example. Let Σ be an alphabet, and let Σ^* be the monoid of strings over the alphabet. The unit is the empty string and multiplication is concatenation. When we think of this as a premulticategory, we can think of the object $*$ as the `unit` type in ML, and we can think of elements s of the monoid Σ^* as commands $\Gamma \vdash \mathsf{output}\, s : *$, with the equation

$$\Gamma \vdash (\mathsf{output}\, s \rightsquigarrow x.\, \mathsf{output}\, t) \equiv \mathsf{output}\, (st) : *.$$

Examples from monads. Moggi proposed monads as models of impure programming languages [30]. Let-binding plays a key role in his metalanguage. This can be explained by noting that every strong monad induces a premulticategory. Examples of monads on the category of sets include the state monad $(S \Rightarrow ((-) \times S))$, the continuations monad $((-) \Rightarrow R) \Rightarrow R$, and the exceptions monad $((-) \uplus E)$. All of these computational effects can be accommodated in premulticategories.

Recall that a monad on the (ordinary) category of sets is given by an assignment of a set $T(A)$ to each set A, a family of functions $\eta_A : A \to T(A)$ and an assignment from functions $f : A \to T(A)$ to functions $f^* : T(A) \to T(B)$, all satisfying some axioms. Given a monad T, we can define a 'Kleisli' premulticategory of T-functions. The objects are sets, and a morphism $\Gamma \to A$ is a function $\prod \Gamma \to T(A)$. The identities are defined using η. Composition is defined as follows: given $t : \Delta \to A$ and $u : (\Gamma, x : A, \Gamma') \to B$ and valuations $\vec{a} \in \prod \Gamma$, $\vec{b} \in \prod \Delta$, $\vec{c} \in \prod \Gamma'$, let

$$(t \rightsquigarrow x.\, u)(\vec{a}, \vec{b}, \vec{c}) \stackrel{\text{def}}{=} u^*(str(\vec{a}, t(\vec{b}), \vec{c}))$$

where $str : \prod(\Gamma, T(A), \Gamma') \to T(\prod(\Gamma, A, \Gamma'))$ is a *strength* which is uniquely defined for any set-theoretic monad [30, Prop. 3.4]. This construction can be carried out for any strong monad on a symmetric monoidal category.

The Kleisli premulticategory induced by a monad is a multicategory if and only if the monad is commutative in the sense of [19]. Commutative monads have special relevance in the study of linear logic (e.g. [4, 16]).

Any monoid $(M, \cdot, e)$ induces a monad $M \times (-)$ on the category of sets. The premulticategory induced by the monoid embeds in the Kleisli premulticategory induced by its monad, and the three notions of commutativity for monoids, monads and premulticategories coincide in this situation.

While many notions of computation can be described by monads, there is a difference between premulticategories and monads: a strong monad cannot be defined without having a product structure and a construction T. We argue that these constructions on types ($\times$ and T) have nothing to do with impure computation. In Section 8.3 we show that these constructions ($\times$ and T) have universal properties when they exist, so that they need not be specified at all.

2.3 Structural laws

Our language for premulticategories doesn't have any structural laws. In a composite $t \rightsquigarrow x.\, u$, there is a linearity constraint: the variables in t must be distinct from the variables in u (except for x). Moreover, the context cannot necessarily be reordered. There are many situations in programming languages where non-linearity is essential, and so we now explain how to introduce the structural laws into the theory of premulticategories.

If $\Gamma = (A_1, \ldots, A_m)$ and $\Delta = (B_1, \ldots, B_n)$ are lists of objects then we define a *renaming* $f : \Gamma \to \Delta$ to be a function $m \to n$ such that $A_i = B_{f(i)}$ for $i \le m$. Renamings compose as functions. (See also e.g. [9, §II.1].)

Definition 2. A premulticategory is *cartesian* when for every morphism $t : \Gamma_1 \to A$ and every renaming $f : \Gamma_1 \to \Gamma_2$ a morphism $t[f] : \Gamma_2 \to A$ is given,

$$\frac{\Gamma_1 \vdash t : A}{\Gamma_2 \vdash t[f] : A}\ (f : \Gamma_1 \to \Gamma_2)$$

such that $t[\mathrm{id}_\Gamma] = t$ and $t[f; g] = (t[f])[g]$ when both sides are defined, and satisfying the rule

$$\frac{\Delta_1 \vdash t : A \qquad \Gamma_1, x : A, \Gamma_1' \vdash u : B}{\Gamma_2, \Delta_2, \Gamma_2' \vdash (t \rightsquigarrow x.\, u)[g_1 \backslash f / g_2] \equiv t[f] \rightsquigarrow x.\, u[g_1 \backslash \mathrm{id}_A / g_2] : B}$$

for any $f : \Delta_1 \to \Delta_2$, $g_1 : \Gamma_1 \to (\Gamma_2, \Gamma_2')$ and $g_2 : \Gamma_1' \to (\Gamma_2, \Gamma_2')$, where the renamings $(g_1 \backslash f / g_2) : (\Gamma_1, \Delta, \Gamma_1') \to (\Gamma_2, \Delta_2, \Gamma_2')$ and $(g_1 \backslash \mathrm{id}_A / g_2) : (\Gamma_1, x : A, \Gamma_1') \to (\Gamma_2, x : A, \Gamma_2')$ are defined in the obvious way.

Special kinds of renaming correspond to the classical structural laws. The bijective renamings of the form $(\Gamma, x : A, y : B, \Gamma') \to (\Gamma, y : B, x : A, \Gamma')$ determine the exchange laws; the injective renamings of the form $\mathsf{wk}_x : (\Gamma, \Gamma') \to (\Gamma, x : A, \Gamma')$ determine the weakening laws; and the surjective renamings of the form $\mathsf{ctrct}_x^{y,z} : (\Gamma, y : A, z : A, \Gamma') \to (\Gamma, x : A, \Gamma')$ determine the contraction laws. All renamings can be built from morphisms of these three kinds (e.g. [6, 20]). We can work with a subset of the structural laws by focusing on a particular well-behaved class of renamings:

Definition 3 (c.f. [22], Def 2.2.21)**.** A premulticategory is *symmetric* when for every morphism $t : \Gamma_1 \to A$ and every bijective renaming $f : \Gamma_1 \to \Gamma_2$ a morphism $t[f] : \Gamma_2 \to A$ is given such that the rules in Definition 2 are satisfied.

(We emphasize that 'symmetric' and 'commutative' are not synonyms.)

In a cartesian premulticategory, some morphisms interact particularly well with the renaming structure (c.f. [12]). We say that a morphism $t : \Delta \to A$ is *discardable* if, for all $u : (\Gamma, \Gamma') \to B$,

$$\Gamma, \Delta, \Gamma' \vdash t \rightsquigarrow x.\, (u[\mathsf{wk}_x]) \equiv u[\mathsf{wk}_\Delta] : B$$

and $t : \Delta \to A$ is *copyable* if, for all $u : (\Gamma, y : A, z : A, \Gamma') \to B$,

$$\Gamma, \Delta, \Gamma' \vdash t \rightsquigarrow x.\, (u[\mathsf{ctrct}_x^{y,z}]) \equiv (t \rightsquigarrow y.\, t \rightsquigarrow z.\, u)[\mathsf{ctrct}_\Delta^{\Delta,\Delta}] : B.$$

Definition 4. A *cartesian multicategory* is a cartesian premulticategory in which every morphism is central, discardable and copyable.

(Various authors have investigated general categorical frameworks that include symmetric multicategories, cartesian multicategories, and many more elaborate examples [5, 7, 15, 22], but

those frameworks are based on simultaneous substitution and do not seem to work well for premulticategories.)

3. Product types and tensor products

In the previous section we introduced an equational theory for programs with let-binding. We did this without making any assumptions about the type constructions in the language.

We now explain what it means for a language to have product/record types. We do this by using the universal property of tensor products, which is well understood for multicategories [14]. This universal property has the immediate consequence that any two implementations of the product type are canonically isomorphic. In this sense, 'having products' is seen as a property and not additional structure. (We discuss a limitation of this view in §5.2.)

In this section and in the following two sections on sums and functions we will deal with particular concepts. In Section 7 we provide general notions of representability and show that these particular concepts are instances of the general notions.

3.1 Tensor products: definition

Consider a list $\Delta = (A_1, \dots, A_n)$ of objects in a premulticategory. The list Δ can be thought of as a specification for a product type $(A_1 * \dots * A_n)$. The list is not actually a product type because it is not actually an object. A tensor product for Δ is a single object that represents it.

Definition 5. A *tensor product* for a list of objects $\Delta = (A_1, .., A_n)$ is an object $\otimes\Delta$ together with a morphism $\Delta \to \otimes\Delta$, which we notate

$$\frac{}{x_1{:}A_1, \dots, x_n{:}A_n \vdash \langle x_1, \dots, x_n\rangle : \otimes\Delta}$$

together with an operation on morphisms

$$\frac{\Gamma, x_1{:}A_1, \dots, x_n{:}A_n, \Gamma' \vdash t : B}{\Gamma, y{:}\otimes\Delta, \Gamma' \vdash y \mathrel{-\!\!\in} \langle x_1 \dots x_n\rangle.\, t : B} \tag{2}$$

for each pair of lists Γ, Γ' and each object B. The notation $y \mathrel{-\!\!\in} \langle x_1 \dots x_n\rangle.\, t$ should be understood as pattern matching a variable of product type into its components $x_1 \dots x_n$, binding them in t.

We require the morphism $\Delta \to \otimes\Delta$ to be central (that is, $\langle x_1..x_n\rangle \rightsquigarrow y.\, t \rightsquigarrow z.\, u \equiv t \rightsquigarrow z.\, \langle x_1..x_n\rangle \rightsquigarrow y.\, u$) and we impose the following two conditions:

$$\frac{\Gamma, x_1{:}A_1, \dots, x_n{:}A_n, \Gamma' \vdash t : B}{\Gamma, x_1{:}A_1, \dots, x_n{:}A_n, \Gamma' \vdash \langle \vec{x}\rangle \rightsquigarrow y.\, (y \mathrel{-\!\!\in} \langle \vec{x}\rangle.\, t) \equiv t : B}$$

$$\frac{\Gamma, y{:}\otimes\Delta, \Gamma' \vdash u : B}{\Gamma, y{:}\otimes\Delta, \Gamma' \vdash y \mathrel{-\!\!\in} \langle x_1, \dots, x_n\rangle.\, (\langle x_1, \dots, x_n\rangle \rightsquigarrow y.\, u) \equiv u : B}$$

The first equation says that if we form a tuple and then extract its elements, nothing happens. The second equation says that if we extract the elements of tuple and then build the tuple again, nothing happens.

In Section 7.2.2 we will explain how this definition amounts to representability in a more abstract sense.

3.2 Examples of tensors

In the multicategory of sets, every list of objects has a tensor product. The tensor product of a list Δ is the product set $\prod\Delta$, that is, the set of valuations for Δ. The universal morphism $\Delta \to \otimes\Delta$ is the identity function. Essentially the same representation works in the premulticategory of stateful functions and the premulticategory of arising from a monad on the category of sets.

Recall that a pointed set is a set A equipped with a chosen element $\bot_A \in A$. In the multicategory of pointed sets, every list $\Delta = (A_1, \dots, A_n)$ has a tensor product given by the 'smash product', which is a quotient of the free pointed set on the product of Δ,

$$\otimes\Delta = (\{\bot_{\otimes\Delta}\} \cup \textstyle\prod\Delta)/_\sim$$

where $\sim$ is generated by $\bot_{\otimes\Delta} \sim (a_1, \dots, \bot_{A_i}, \dots, a_n)$.

Pointed sets are a simple algebraic theory. More generally, one can build a multicategory from *any* algebraic theory, with algebras as objects and multilinear maps as morphisms. A famous example is the of multicategory of vector spaces which also has tensor products. A well-known characterization theorem says that, for a given algebraic theory, the multicategory of algebras has tensor products if and only the corresponding free-algebra monad is commutative (c.f. [25]).

Connection with programming languages. The syntax for tensor products is very similar to the syntax in ML. In Standard ML, the object $\otimes\Delta$ would be written $(A_1 * \dots * A_n)$, and the assignment $y \mathrel{-\!\!\in} \langle x_1, \dots, x_n\rangle.\, t$ is written as follows:

$$\texttt{let val}\ (x_1, \dots, x_n) = y\ \texttt{in}\ t\ \texttt{end}.$$

The centrality of the universal morphism and the two equations are straightforward observational equivalences. Thus we can build a premulticategory with all tensor products whose objects are types and whose morphisms are terms in context modulo observational equivalence.

If the empty list $()$ has a tensor product $\otimes()$ then this behaves like the `unit` type in ML. For instance, the sequencing notation $(t\ ;\ u)$ can be understood as shorthand for $t \rightsquigarrow x.\, x \mathrel{-\!\!\in} \langle\rangle.\, u$.

A convenient programming practice is to write tuples with terms in the components. By fixing a left-to-right order of evaluation, we have the following derived typing rule:

$$\frac{\Gamma_1 \vdash t_1 : A_1 \quad \dots \quad \Gamma_n \vdash t_n : A_n}{\Gamma_1..\Gamma_n \vdash \langle t_1..t_n\rangle \stackrel{\text{def}}{=} t_1 \rightsquigarrow x_1.\, \dots t_n \rightsquigarrow x_n.\, \langle x_1..x_n\rangle : \otimes\Delta} \tag{3}$$

Connection with logic. The rule for matching (2) is the left rule for tensor products in the sequent calculus presentation of linear logic. The right rule of the sequent calculus looks like the derived rule (3) although there is nothing canonical about it in the non-commutative setting: the order of evaluation of $t_1 \dots t_n$ matters unless they are central.

3.3 Basic first results

We now state some basic results that can be derived from the definition of tensor product. They are instances of general results in Section 7.2.

Proposition 6. *Matching associates and commutes with composition:*

$$\frac{\Gamma_1, \Delta, \Gamma_1' \vdash t : B \qquad \Gamma_2, z{:}B, \Gamma_2' \vdash u : C}{\Gamma_2, \Gamma_1, y{:}\otimes\Delta, \Gamma_1', \Gamma_2' \vdash (y \mathrel{-\!\!\in} \langle \vec{x}\rangle.\, t) \rightsquigarrow z.\, u \equiv y \mathrel{-\!\!\in} \langle \vec{x}\rangle.\, (t \rightsquigarrow z.\, u) : C}$$

$$\frac{\Gamma_1 \vdash t : B \qquad \Gamma_2, \Delta, z{:}B, \Gamma_2' \vdash u : C}{\Gamma_2, y{:}\otimes\Delta, \Gamma_1, \Gamma_2' \vdash y \mathrel{-\!\!\in} \langle \vec{x}\rangle.\, (t \rightsquigarrow z.\, u) \equiv t \rightsquigarrow z.\, (y \mathrel{-\!\!\in} \langle \vec{x}\rangle.\, u) : C}$$

Proposition 7. *For any list $\Delta = (x_1{:}A_1, \dots, x_n{:}A_n)$ of objects, any object R and any morphism $r{:}\Delta \to R$ there is a unique morphism $t{:}(y{:}\otimes\Delta) \to R$ such that $\langle x_1, \dots, x_n\rangle \rightsquigarrow y.\, t \equiv r$. The unique morphism t is central if r is central.*

As a corollary of this result we achieve our main theorem:

Theorem 8. *Tensor products are unique up to unique isomorphism, and the unique mediating isomorphisms are central.*

In other words, there is a canonical way to translate between any two implementations of product types.

4. Sums and labelled variants

In the previous section we demonstrated that products can be given a universal property in a model of an impure programming language. In this section we show that sum types (labelled variants) can also be given a universal property in a similar way.

4.1 Definitions: constructor contexts and sums

We fix an infinite set of 'constructors'. A *constructor context* is a finite partial function from constructors to objects of a premulticategory. A *sum* for a constructor context Δ is an object $\oplus\Delta$ and a family of morphisms $\{\Delta(\mathrm{c}) \to \oplus\Delta\}_{\mathrm{c}\in dom(\Delta)}$:

$$\frac{}{x{:}\,\Delta(\mathrm{c}) \vdash \mathrm{c}(x) : \oplus\Delta}\ (\mathrm{c} \in dom(\Delta))$$

that is universal in the sense that each morphism is central and there is an operation on morphisms

$$\frac{\Gamma, x_{\mathrm{c}}{:}\,\Delta(\mathrm{c}), \Gamma' \vdash t_{\mathrm{c}} : A \quad (\mathrm{c} \in dom(\Delta))}{\Gamma, y{:}\,\oplus\Delta, \Gamma' \vdash \mathsf{match}\, y\, \mathsf{as}\, (\mathrm{c}(x_{\mathrm{c}}) \Rightarrow t_{\mathrm{c}})_{\mathrm{c}\in dom(\Delta)} : A}$$

subject to the following equations:

$$\frac{\Gamma, x_{\mathrm{c}}{:}\,\Delta(\mathrm{c}), \Gamma' \vdash t_{\mathrm{c}} : A \quad (\mathrm{c} \in dom(\Delta))}{\Gamma, x_{\mathrm{d}}{:}\,\Delta(\mathrm{d}), \Gamma' \vdash \mathrm{d}(x_{\mathrm{d}}) \rightsquigarrow y.\, \mathsf{match}\, y\, \mathsf{as}\, \big(\mathrm{c}(x_{\mathrm{c}}) \Rightarrow t_{\mathrm{c}}\big)_{\mathrm{c}} \equiv t_{\mathrm{d}} : A}$$

$$\frac{\Gamma, y{:}\,\oplus\Delta, \Gamma' \vdash t : A}{\Gamma, y{:}\,\oplus\Delta, \Gamma' \vdash \mathsf{match}\, y\, \mathsf{as}\, \big(\mathrm{c}(x_{\mathrm{c}}) \Rightarrow (\mathrm{c}(x_{\mathrm{c}}) \rightsquigarrow y.\, t)\big)_{\mathrm{c}} \equiv t : A}$$

4.2 Examples of sums

Sums in set-theoretic premulticategories. The multicategory of sets has sets as objects and multivariate functions as morphisms. Sums of constructor contexts are given by disjoint unions:

$$\oplus\Delta \stackrel{\text{def}}{=} \{(\mathrm{c}, a) \mid a \in \Delta(\mathrm{c})\}.$$

The universal maps take $a \in \Delta(\mathrm{c})$ to $(\mathrm{c}, a) \in \oplus\Delta$. A similar analysis works for stateful functions and for the premulticategory arising from a monad on a distributive category.

In the multicategory of pointed sets, the objects are sets A equipped with an element $\bot_A$, and the morphisms are multivariate functions that are strict in each argument. This multicategory also has sums of constructor contexts, given by a coalesced sum:

$$\oplus\Delta \stackrel{\text{def}}{=} (\{(\mathrm{c}, a) \mid a \in \Delta(\mathrm{c})\} \uplus \bot_{\oplus\Delta})/_{\sim} \qquad \text{where } \bot_{\oplus\Delta} \sim (\mathrm{c}, \bot_{\Delta(\mathrm{c})}).$$

Connections with programming languages. In ML-like languages, variant types must be declared. Given a constructor context $\Delta = (\mathrm{c}_1{:}\,A_1, \ldots, \mathrm{c}_n{:}\,A_n)$, we would expect an ML definition

$$\texttt{datatype}\ \oplus\Delta\ \texttt{=}\ \mathrm{c}_1 : A_1 \mid \ldots \mid \mathrm{c}_n : A_n.$$

The universal family of morphisms can be combined with composition to yield the more familiar term formation for constructors:

$$\frac{\Gamma \vdash t : \Delta(\mathrm{c})}{\Gamma \vdash \mathrm{c}(t) \stackrel{\text{def}}{=} t \rightsquigarrow x.\, \mathrm{c}(x) : A}\ (\mathrm{c} \in dom(\Delta))$$

The equations in the definition are straightforward observational equivalences in a programming language.

4.2.1 Basic first results

We state some basic properties that can be derived from the definition of sums. They are instances of general results in Section 7.2.

Proposition 9. *Matching associates and commutes with composition:*

$$\frac{\Gamma_1, x_{\mathrm{c}}{:}\,\Delta(\mathrm{c}), \Gamma_1' \vdash t_{\mathrm{c}} : A\ (\mathrm{c} \in dom(\Delta)) \quad \Gamma_2, z{:}\,A, \Gamma_2' \vdash u : B}{\begin{array}{l}\Gamma_2, \Gamma_1, y{:}\,\oplus\Delta, \Gamma_1', \Gamma_2' \vdash \mathsf{match}\, y\, \mathsf{as}\, (\mathrm{c}(x_{\mathrm{c}}) \Rightarrow (t_{\mathrm{c}} \rightsquigarrow z.\, u))_{\mathrm{c}} \\ \qquad\qquad \equiv\ (\mathsf{match}\, y\, \mathsf{as}\, (\mathrm{c}(x_{\mathrm{c}}) \Rightarrow t_{\mathrm{c}})_{\mathrm{c}}) \rightsquigarrow z.\, u : B\end{array}}$$

$$\frac{\Gamma_1 \vdash t : A \qquad \Gamma_2, x_{\mathrm{c}}{:}\,\Delta(\mathrm{c}), z{:}\,A, \Gamma_2' \vdash u_{\mathrm{c}} : B \quad (\mathrm{c} \in dom(\Delta))}{\begin{array}{l}\Gamma_2, y{:}\,\oplus\Delta, \Gamma_1, \Gamma_2' \vdash \mathsf{match}\, y\, \mathsf{as}\, (\mathrm{c}(x_{\mathrm{c}}) \Rightarrow (t \rightsquigarrow z.\, u_{\mathrm{c}}))_{\mathrm{c}} \\ \qquad\qquad \equiv\ t \rightsquigarrow z.\, (\mathsf{match}\, y\, \mathsf{as}\, (\mathrm{c}(x_{\mathrm{c}}) \Rightarrow u_{\mathrm{c}})_{\mathrm{c}}) : B\end{array}}$$

Proposition 10. *For any constructor context Δ, any object R and any family of morphisms $\{r_{\mathrm{c}}{:}\,(x{:}\,\Delta(\mathrm{c})) \to R\}_{\mathrm{c}\in dom(\Delta)}$, there is a unique morphism $t{:}\,(y{:}\,\oplus\Delta) \to R$ such that for all $\mathrm{c} \in dom(\Delta)$,*

$$x{:}\,\Delta(\mathrm{c}) \vdash \mathrm{c}(x) \rightsquigarrow y.\, t \equiv r_{\mathrm{c}} : R.$$

The mediating morphism t is central if each r_{c} is central.

In consequence, sums are unique up to unique isomorphism, and the unique mediating isomorphisms are central.

5. On values and Freyd multicategories

In impure programming languages it is usually necessary to identify a class of values among the class of all programs. For instance, in an impure functional programming language, the most natural evaluation strategy for function application is call-by-value: expressions are reduced to values before they are passed as arguments.

We have proposed to understand a program as a morphism in a premulticategory. To accommodate values, we must identify a class of morphisms which are the values. We call this structure a 'Freyd multicategory'. The idea of distinguishing between values and computations is long-established in the categorical study of semantics, stemming from Moggi's distinction between ordinary morphisms and Kleisli morphisms [30], running through the work on premonoidal categories (e.g. [18, 32, 33]) and Freyd categories [24, 34]. The distinction between values and computations also arises in 'arrows' in functional programming [2, 3, 17] and forms the basis of recent syntactic systems (e.g. [8, 23, 28]). The idea of identifying a class of values also plays a crucial technical role from the perspective of morphisms between premulticategories with tensor products.

5.1 Morphisms of premulticategories

If premulticategories represent programming languages, then a morphism between them is a compositional translation.

Definition 11. Let $\mathcal{C}$ and $\mathcal{D}$ be premulticategories. A morphism $F : \mathcal{C} \to \mathcal{D}$ is given by an assignment of an object FA of $\mathcal{D}$ to each object A of $\mathcal{C}$, and an assignment of a morphism $Ff : F\Gamma \to FB$ in $\mathcal{D}$ to each morphism $f{:}\,\Gamma \to B$ in $\mathcal{C}$, such that identities and composition are preserved.

Our running example concerns the two-element monoid $S_2 = (\{\texttt{flip}, e\}, \cdot, e)$ where $\texttt{flip} \cdot \texttt{flip} = e$. As explained in Section 2.2, the monad $S_2 \times (-)$ on the category of sets induces a Kleisli premulticategory $\mathcal{C}$ whose objects are sets and where a morphism $\Gamma \to B$ is a function $\prod \Gamma \to S_2 \times B$. It can be thought of as a semantics for a simple programming language with a command `flip` that flips the contents of a bit of memory.

We also consider the premulticategory $\mathcal{D}$ of stateful functions (§2.2) with two states $Bool = \{\mathrm{True}, \mathrm{False}\}$. The objects of $\mathcal{D}$ are sets and the morphisms $\Gamma \to B$ are functions $\prod \Gamma \times Bool \to B \times Bool$.

There is a morphism of multicategories $F{:}\,\mathcal{C} \to \mathcal{D}$ that is identity on objects and takes a morphism $f : \Gamma \to B$ in $\mathcal{C}$ to the stateful function given by $(Ff)(\vec{a}, s) = (b, s \text{ xor } s')$ where $f(\vec{a}) = (b, s')$,

i.e., interpreting the command `flip` as an instruction that flips the state. This can be understood as arising from a monad morphism $(S_2 \times -) \to (- \times Bool)^{Bool}$.

5.2 Discussion: Preservation of tensors

We now investigate what it means for a morphism of premulticategories to preserve tensor products. Since tensor products have a universal property we would expect preservation of tensors to be a property rather than extra structure.

Given a list $\Delta = (A_1, \ldots, A_n)$ of objects of $\mathcal{V}$ and a tensor product $r\colon \Delta \to R$, we say that a morphism $F\colon \mathcal{C} \to \mathcal{D}$ preserves the tensor if Fr is also a tensor.

Proposition 12. *Let $F : \mathcal{C} \to \mathcal{D}$ be a morphism between premulticategories, where $\mathcal{C}$ has tensor products. The following are equivalent:*

1. *F preserves every tensor product.*
2. *F preserves central isomorphisms, and every list of objects in $\mathcal{C}$ has a tensor product that is preserved by F.*

The issue here is that there may be many different tensors for a given list of objects, all related by canonical central isomorphisms. If F preserves central isomorphisms, e.g. if $\mathcal{D}$ is is a multicategory, then the equivalence of items 1 and 2 allows us to check that F preserves all tensor products by checking that one chosen tensor product is preserved.

To illustrate this, let us return to our example of the Kleisli premulticategory $\mathcal{C}$ and the premulticategory $\mathcal{D}$ of stateful functions. The premulticategory $\mathcal{C}$ is actually a multicategory, since e and `flip` commute. The morphism $F : \mathcal{C} \to \mathcal{D}$ does not preserve centrality, because $F(\texttt{flip}, \mathrm{id}) : () \to 1$ is not central in $\mathcal{D}$.

This example also illustrates the complications involved with preservation of tensor products in the non-commutative setting. In $\mathcal{C}$, every list Δ has a tensor product given by $(e, \mathrm{id})\colon \Delta \to \prod \Delta$. This is a reasonable implementation of the tensor product from the programming language perspective. This tensor product is preserved by the morphism $F : \mathcal{C} \to \mathcal{D}$ into the premulticategory of stateful functions. However, the structure $(\texttt{flip}, \mathrm{id})\colon \Delta \to \prod \Delta$ also happens to be a tensor product of the list Δ in $\mathcal{C}$. This tensor product is not preserved by the morphism $F\colon \mathcal{C} \to \mathcal{D}$, because $F(\texttt{flip}, \mathrm{id})$ is not central in $\mathcal{D}$.

In the example, informally, $(\texttt{flip}, \mathrm{id})$ is central in $\mathcal{C}$ by accident: it would have been better to explicitly designate (e, id) as a value and $(\texttt{flip}, \mathrm{id})$ as a non-value. We could then require this special class of central 'value' morphisms to be used for the representations for tensors, and we could require this class of value morphisms to be preserved by morphisms of premulticategories.

Before we make this formal, we briefly consider another way to resolve the situation: we could remove the requirement that the representing morphism $\Delta \to \otimes\Delta$ be central. This is difficult to motivate from a pragmatic perspective. Moreover from the mathematical perspective it is badly behaved, because the induced family of associativity morphisms $\alpha_{A,B,C} : \otimes(\otimes(A, B), C) \to \otimes(A, \otimes(B, C))$ is not natural and moreover does not support a coherence result: the pentagon diagram does not commute (see §8).

5.3 Definition: Freyd multicategory

Informally, a Freyd multicategory is a premulticategory that is equipped with data about which morphisms are values.

Formally, a Freyd multicategory is a morphism $\mathit{return}\colon \mathcal{V} \to \mathcal{C}$ from a multicategory $\mathcal{V}$ to a premulticategory $\mathcal{C}$ such that $\mathcal{V}$ and $\mathcal{C}$ have the same objects and return is identity on objects and preserves centrality. The idea is that morphisms in $\mathcal{V}$ are values, morphisms in $\mathcal{C}$ are computations, and $\mathit{return}(v)$ is the computation that immediately returns v. We do not require return to be injective on morphisms, although it often will be in practice.

We notate Freyd multicategories by introducing a special judgement $\Gamma \vdash_{\mathcal{V}} v : A$ of values (morphisms in $\mathcal{V}$) and a rule

$$\frac{\Gamma \vdash_{\mathcal{V}} v : A}{\Gamma \vdash \mathit{return}(v) : A}$$

describing the morphism $\mathit{return}\colon \mathcal{V} \to \mathcal{C}$. The functoriality of return is expressed by the following equations:

$$\frac{}{x\colon A \vdash x \equiv \mathit{return}(x) : A}$$

$$\frac{\Delta \vdash_{\mathcal{V}} v : A \qquad \Gamma, x\colon A, \Gamma' \vdash_{\mathcal{V}} w : B}{\Gamma, \Delta, \Gamma' \vdash \mathit{return}(v \rightsquigarrow x.\, w) \equiv \mathit{return}(v) \rightsquigarrow x.\, \mathit{return}(w) : B}$$

We also have the two equations stating that values are central among all morphisms (§2.1).

Because $\mathcal{V}$ is a multicategory and $\mathcal{V} \to \mathcal{C}$ preserves centrality then we can understand $\mathit{return}(v) \rightsquigarrow x.\, t$ as an explicit substitution of the value v for x in t. In the syntax, we can textually substitute $t\{v/x\}$ as shorthand for $\mathit{return}(v) \rightsquigarrow x.\, t$. (Note that this implicit substitution is only an informal convention, whereas it is genuine syntax in the explicit substitutions of Abadi et al. [1].)

A *morphism of Freyd multicategories*

$$(F_1, F_2) : (\mathcal{V} \xrightarrow{\mathit{return}} \mathcal{C}) \longrightarrow (\mathcal{V}' \xrightarrow{\mathit{return}'} \mathcal{C}')$$

is a pair of morphisms between premulticategories $(F_1\colon \mathcal{V} \to \mathcal{V}'$, $F_2\colon \mathcal{C} \to \mathcal{C}')$ such that $\mathit{return}' \circ F_1 = F_2 \circ \mathit{return}$.

5.3.1 Examples of Freyd multicategories

Recall that the premulticategory of stateful functions has sets as objects and stateful functions as morphisms. A stateful function $\Gamma \to A$ is an ordinary function $(\prod \Gamma) \times S \to A \times S$, where S is a fixed set of states. In this context we let our multicategory of values be the multicategory of sets and pure (stateless) functions, and $\mathit{return}\colon \mathcal{V} \to \mathcal{C}$ is the evident inclusion morphism, which is faithful provided S is not empty.

Given a monad T, recall that the associated Kleisli premulticategory has sets as objects and morphisms $\Gamma \to A$ are ordinary functions $(\prod \Gamma) \to T(A)$. In this context we let our multicategory of values be the category of sets and pure functions. The morphism $\mathit{return}\colon \mathcal{V} \to \mathcal{C}$ is defined by the unit of the monad.

We can now revisit our morphism $\mathcal{C} \to \mathcal{D}$ from the Kleisli premulticategory induced by the monad $(S_2 \times -)$ to the premulticategory of stateful functions (§5.2). In both cases, the value multicategory is the multicategory of sets and pure functions, and $(\mathrm{Id}, F) : \mathcal{C} \to \mathcal{D}$ is a morphism of Freyd multicategories.

5.3.2 Tensors and sums in Freyd multicategories

We define a *Freyd multicategory with tensors/sums* to be a Freyd multicategory $\mathit{return}\colon \mathcal{V} \to \mathcal{C}$ in which $\mathcal{V}$ has tensors/sums and return preserves them. This makes sense because return preserves centrality.

We say that a morphism of Freyd multicategories $(F_1, F_2) : (\mathcal{V} \to \mathcal{C}) \to (\mathcal{V}' \to \mathcal{C}')$ preserves tensors (resp. sums) if F_1 preserves tensors (resp. sums). To illustrate, we return to our motivating example: the morphism from the Freyd multicategory induced by the monad $(S_2 \times -)$ to the Freyd multicategory of stateful functions. This does preserves tensors.

As an aside we remark that, from the syntactic point of view, it is slightly unnatural to ask for $\mathcal{V}$ to have tensors/sums. For tensors, although it is usual to consider the representing maps $\langle v_1, \ldots, v_n \rangle$ as values, it is less common to consider the pattern matching syntax $x \mathrel{-\!\!\in} \langle \vec{y} \rangle.\, v$ as a value expression. Levy [24] coined the phrase 'complex value' for this situation. Our framework provides us with an option for describing tensors with*out* using complex values,

by asking $\mathcal{C}$ to have tensors and for the representing maps to come from $\mathcal{V}$, without asking for $\mathcal{V}$ to have tensors. Using implicit substitution, we would have a more familiar grammar for values:

$$v ::= \langle v_1, \ldots, v_n \rangle \mid X[v] \mid \ldots$$

Notice that these maps in $\mathcal{V}$ are not uniquely determined unless the functor $return\colon \mathcal{V} \to \mathcal{C}$ is injective on morphisms. Moreover, two such representations are not necessarily isomorphic in $\mathcal{V}$.

5.3.3 Structural laws

We can also speak of symmetric Freyd multicategories, which are Freyd multicategories $return\colon \mathcal{V} \to \mathcal{C}$ such that $\mathcal{V}$ and $\mathcal{C}$ are symmetric (Def. 3) and $return$ preserves the renaming structure:

$$\frac{\Gamma_1 \vdash_{\mathcal{V}} v : A}{\Gamma_2 \vdash return(v[f]) \equiv (return(v))[f] : A} \quad (f\colon \Gamma_1 \xrightarrow{\cong} \Gamma_2)$$

We say that a Freyd multicategory $return\colon \mathcal{V} \to \mathcal{C}$ is *cartesian* when $\mathcal{V}$ is cartesian as a multicategory (Def. 4), $\mathcal{C}$ is cartesian as a premulticategory (Def. 2), and $return$ preserves the renaming structure. All the examples in Section 5.3.1 are cartesian.

6. Function spaces

We have introduced an equational theory for impure programs (§2) and shown that product and sum types can be characterized by universal properties (§3, 4). We now characterize call-by-value function types by a universal property.

We characterize function types in the context of Freyd multicategories (§5.3) by retracing the steps taken by Power and Thielecke [34]. The key point is that first-class call-by-value functions can delay computations. For instance, in ML the expression

```
fn x => (print "testing" ; 3)
```

does not immediately print: it will only print when applied to an argument. We use Freyd multicategories to distinguish between computations, which can be delayed, and values, which cannot be delayed.

In what follows we will assume that our premulticategories are symmetric, in the sense of Section 2.3. This means that objects in contexts can be reordered. It is clumsy to work with function spaces without this assumption.

6.1 Definition: function spaces

We define the concept of function space in the setting of a symmetric Freyd multicategory (§5.3): a symmetric multicategory $\mathcal{V}$ (values), a symmetric premulticategory $\mathcal{C}$ (computations) and an identity-on-objects morphism of symmetric premulticategories $return\colon \mathcal{V} \to \mathcal{C}$ such that each $return(v)$ is central.

Let Δ be a list of objects and let A be an object. We will define what it means for the Freyd multicategory to have a function space $(\Delta \Rightarrow A)$. The idea is that the 'inhabitants' of the function space are functions that take a valuation of the context Δ and return a result of type A, perhaps with some side-effects along the way. When Δ is empty then the object $(\Delta \Rightarrow A)$ behaves like a space of delayed computations, like Moggi's monadic type constructor.

We do not have the currying isomorphism in a call-by-value language, and unless the Freyd multicategory has all tensor products we cannot accurately express n-ary functions in terms of unary functions.

Definition 13. A symmetric Freyd multicategory $return\colon \mathcal{V} \to \mathcal{C}$ has *function spaces* if for every list $\Delta = (x_1\colon A_1, \ldots, x_n\colon A_n)$ and every object B there is an object $(\Delta \Rightarrow B)$ and a morphism $(\Delta \Rightarrow B, \Delta) \to B$ (not necessarily central):

$$\frac{}{f : \Delta \Rightarrow B, x_1\colon A_1, \ldots, x_n\colon A_n \vdash f(\vec{x}) : B}$$

together with an operation on morphisms:

$$\frac{\Gamma, \Delta \vdash t : B}{\Gamma \vdash_{\mathcal{V}} \lambda(\Delta)t : (\Delta \Rightarrow B)} \tag{4}$$

subject to the following equations:

$$\frac{\Gamma, \Delta \vdash t : B}{\Gamma, \Delta \vdash return(\lambda(\vec{x})t) \rightsquigarrow f.\, f(\vec{x}) \equiv t : B}$$

$$\frac{\Gamma \vdash_{\mathcal{V}} v : \Delta \Rightarrow B}{\Gamma \vdash_{\mathcal{V}} \lambda(\vec{x}).\, (return(v) \rightsquigarrow f.\, f(\vec{x})) \equiv v : \Delta \Rightarrow B}$$

We work up to α-equivalence — we consider variables to be informal notation for indices in lists.

6.2 Examples of function spaces

Set theoretic examples. The Freyd multicategory of sets has $\mathcal{V} = \mathcal{C}$ as the multicategory with sets as objects and multivariate functions as morphisms. The function space $(\Delta \Rightarrow B)$ is the set of functions $(\prod \Delta) \to B$. The representing map is the evaluation function, which takes a function and a valuation and evaluates that function.

Recall that the Freyd multicategory of stateful functions has $\mathcal{V}$ as the multicategory of sets and functions and $\mathcal{C}$ as the premulticategory of sets and stateful functions. The function space $(\Delta \Rightarrow B)$ is the set of all stateful functions $\Delta \to B$. Notice that, in particular, the function space $(() \Rightarrow B)$ is the state monad $(S \to B \times S)$.

Given a monad T on the category of sets, recall that the corresponding Freyd multicategory has $\mathcal{V}$ as the multicategory of sets and functions and $\mathcal{C}$ as a premulticategory of sets and Kleisli functions. The function space $(\Delta \Rightarrow B)$ is the set of all functions $(\prod \Delta) \to T(B)$. In particular, the function space $(() \Rightarrow B)$ is isomorphic to the monad T.

Connection to programming languages. The connection with programming languages is hopefully clear. The two equations are variants of the β and η equations which are straightforward observational equivalences in all higher-typed languages. We suggest the following syntactic sugar: let

$$f(t_1, \ldots, t_n) \stackrel{\text{def}}{=} t_1 \rightsquigarrow x_1 \ldots t_n \rightsquigarrow x_n.\, f(\vec{x}). \tag{5}$$

In this 'call-by-value' semantics, expressions are reduced before being passed as arguments.

We certainly have not captured exactly observational equivalence. That is not our aim. Our aim is to identify an equational theory that holds in all good models. Our equations hold in syntactic models and also in more semantic models which are not fully abstract but which are nonetheless useful.

Connections with proof theory. Let us briefly investigate the extent to which the Curry-Howard correspondence is relevant in the call-by-value setting. The rule for λ-abstraction (4) is essentially the right implication rule of the sequent calculus. The syntactic sugar (5) leads to the left rule for implication:

$$\frac{\Gamma \vdash t_1 : A_1 \ \ldots \ \Gamma \vdash t_n : A_n \qquad \Gamma, y\colon B \vdash u : C}{\Gamma, f\colon (A_1, \ldots, A_n) \Rightarrow B \vdash f(t_1, \ldots, t_n) \rightsquigarrow y.\, u : C}$$

which is clearly not canonical, because it depends on the order of $t_1, \ldots, t_n$. In fact, the explicit let-binding helps us to distinguish different proofs. For example, consider the following two terms:

$$f\colon A \Rightarrow B, x\colon A \vdash return(\lambda(y\colon C).\, (f(x))) : C \Rightarrow B$$

$$f\colon A \Rightarrow B, x\colon A \vdash (f(x)) \rightsquigarrow w.\, return(\lambda(y\colon C).\, w) : C \Rightarrow B$$

If $(f(x))$ is a value, which is an assumption that the language is pure, then these terms are equal. Without this assumption, these

are two distinct terms which correspond to two different sequent calculus proofs, as first observed by Herbelin [13].

7. Representability in general

The last two technical sections of this article place our work on a secure abstract foundation.

Throughout this article we have spoken of representability and universal properties on the understanding that these concepts lend a canonical status to various constructions. We now provide a general notion of representability that accounts for the constructions that we have introduced: products, sums and function spaces.

For an ordinary category $\mathcal{C}$ there are two notions of representability that are dual to each other: we can speak of representability for a covariant functor $\mathcal{C} \to \mathbf{Set}$ and of representability for a contravariant functor $\mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$. The covariant notion of representability allows us to describe colimits, and the contravariant notion allows us to describe limits.

The situation is more sophisticated with multicategories, for there is no reasonable notion of 'dual' for multicategories. Thus we must treat the two variances differently. We do this by considering representability for left and right modules of (pre)multicategories.

Very informally, we may say that a left or right module specifies a concept – a concept like product, sum or function. A representation for the module is a first class type that represents the concept.

Throughout this section we work with symmetric premulticategories. This means that we can reorder the contexts, which makes the notation easier.

7.1 Left modules and left representability

We introduce a notion of left module and of representability for premulticategories. This general notion specializes to our characterization of function types.

7.1.1 Definitions and first results

Given a symmetric premulticategory $\mathcal{C}$ we define a *left module* M to be an assignment of a set $M(\Gamma)$ to each list Γ of objects, together with a family of functions

$$\mu_{\Gamma,A,\Delta,\Gamma'} : \mathcal{C}(\Delta; A) \times M(\Gamma, A, \Gamma') \to M(\Gamma, \Delta, \Gamma')$$

satisfying two conditions. We write $\Gamma \vdash_M t$ when $t \in M(\Gamma)$, and we write

$$\frac{\Delta \vdash t : A \qquad \Gamma, x{:}\, A, \Gamma' \vdash_M u}{\Gamma, \Delta, \Gamma' \vdash_M t \overset{M}{\rightsquigarrow} x.\, u}$$

for $\mu_{\Gamma,A,\Delta,\Gamma'}(t, u)$. The two conditions are

$$\frac{\Gamma, x{:}\, A, \Gamma' \vdash_M t}{\Gamma, x{:}\, A, \Gamma' \vdash_M x \overset{M}{\rightsquigarrow} x.\, t \equiv t} \tag{LM1}$$

$$\frac{\Gamma_3 \vdash t : A \quad \Gamma_2, x{:}\, A, \Gamma_2' \vdash u : B \quad \Gamma_1, y{:}\, B, \Gamma_1' \vdash_M v}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_2', \Gamma_1' \vdash_M t \overset{M}{\rightsquigarrow} x.\, (u \overset{M}{\rightsquigarrow} y.\, v) \equiv (t \rightsquigarrow x.\, u) \overset{M}{\rightsquigarrow} y.\, v} \tag{LM2}$$

Representations of left modules. For a fixed object R, we can understand $\mathcal{C}(-; R)$ as a left module.

Definition 14. Let M be a left module for a symmetric premulticategory $\mathcal{C}$. A *representation* of M is an object R of $\mathcal{C}$ together with an isomorphism of left modules:

$$\{\mathcal{C}(\Gamma; R) \cong M(\Gamma)\}_\Gamma$$

(An isomorphism of left modules is a family of bijections that respects the left module structure.)

A representation is always determined by an object R and an element of $M(R)$. To show this, we use the following variant of the Yoneda lemma. (Technically this can be seen an instance of the Yoneda lemma for ordinary categories.)

Proposition 15. *Let M be a left module for a symmetric premulticategory $\mathcal{C}$ and let R be an object. To give a left module morphism $\{\mathcal{C}(\Gamma; R) \to M(\Gamma)\}_\Gamma$ is to give an element of $M(R)$.*

In more detail, given an object R and an element $r \in M(R)$ we have a family of functions $(R, r)_\Gamma : \mathcal{C}(\Gamma; R) \to M(\Gamma)$ given by $(R, r)_\Gamma(t) \overset{\text{def}}{=} t \overset{M}{\rightsquigarrow} x.\, r$:

$$\frac{\Gamma \vdash t : R}{\Gamma \vdash_M t \overset{M}{\rightsquigarrow} x.\, r} \qquad (\text{using} \quad x{:}\, R \vdash_M r).$$

Conversely, given a left module morphism, i.e. a family of functions $\{\phi_\Gamma{:}\, \mathcal{C}(\Gamma; R) \to M(\Gamma)\}_\Gamma$ that respects the left module structure, we recover an element $\phi_R(\mathrm{id}_R)$ of $M(R)$. Proposition 15 says that these two constructions are mutually inverse.

Corollary 16. *Let M be a left module of a symmetric premulticategory. The following data are equivalent.*

- *A representation for M (Def. 14).*
- *An object R and an element $r \in M(x{:}\, R)$ together with a family of functions $\rho_\Gamma : M(\Gamma) \to \mathcal{C}(\Gamma; R)$ notated*

$$\frac{\Gamma \vdash_M t}{\Gamma \vdash \rho\, t : R}$$

satisfying the following conditions:

$$\frac{\Gamma \vdash_M t}{\Gamma \vdash_M (\rho\, t) \overset{M}{\rightsquigarrow} x.\, r \equiv t} \qquad \frac{\Gamma \vdash t : R}{\Gamma \vdash \rho(t \overset{M}{\rightsquigarrow} x.\, r) \equiv t : R} \tag{LR1–2}$$

Uniqueness of representations.

Proposition 17. *If $(R, r \in M(x{:}\, R))$ is a representation of M and $a \in M(y{:}\, A)$ then there is a unique morphism $t{:}\, (y{:}\, A) \to R$ such that $y{:}\, A \vdash_M a \equiv t \overset{M}{\rightsquigarrow} x.\, r$. In consequence, representations are unique up to unique isomorphism.*

7.1.2 Example: function spaces

In Section 6 we defined a notion of function space for Freyd multicategories $\mathit{return}{:}\, \mathcal{V} \to \mathcal{C}$. Given a context Δ and an object B, we define a module M for $\mathcal{V}$ as follows:

$$M(\Gamma) \overset{\text{def}}{=} \mathcal{C}(\Gamma, \Delta; B).$$

The module structure μ is defined in terms of the premulticategory structure of $\mathcal{C}$:

$$\frac{\Gamma_2 \vdash_{\mathcal{V}} v : A \qquad \Gamma_1, x{:}\, A, \Gamma_1' \vdash_M u}{\Gamma_1, \Gamma_2, \Gamma_1' \vdash_M (v \overset{M}{\rightsquigarrow} x.\, u) \overset{\text{def}}{=} (\mathit{return}(v) \rightsquigarrow x.\, u)}$$

The two conditions LM1–2 are immediately verified.

The data for a left representation is exactly the data for a function space $(\Delta \Rightarrow B)$. The axioms LR1–2 are exactly the axioms for function spaces in Section 6.1.

7.1.3 Centrality in terms of module morphisms

We briefly remark that left modules provide a more abstract account of centrality. Recall that a morphism t is central if it commutes with all morphisms: for all u and v, $t \rightsquigarrow x.\, u \rightsquigarrow y.\, v \equiv u \rightsquigarrow y.\, t \rightsquigarrow x.\, v$. This can be explained in a more abstract way as follows. Every morphism $t{:}\, \Delta \to A$ determines a family of functions between sets of morphisms:

$$(t \rightsquigarrow x.\, -) : \mathcal{C}(\Gamma, x{:}\, A; B) \to \mathcal{C}(\Gamma, \Delta; B)$$

The domain and codomain have an obvious left module structure, and the morphism t is central if and only if this family of functions is a left module morphism for all B.

7.2 Right modules and right representability

We now provide a notion of right module, which is an account of composing on the right. The induced notion of representability accounts for the product and sum types that we introduced in Sections 3 and 4.

7.2.1 Definitions and first results

Given a symmetric premulticategory $\mathcal{C}$ we define a *right module* M to be an assignment of a set $M(\Gamma; A)$ to each list Γ and each object A, together with for each B the structure of a left module (§7.1.1)

$$\mu_{\Gamma,A,\Delta,\Gamma';B} : \mathcal{C}(\Delta; A) \times M(\Gamma, A, \Gamma'; B) \to M(\Gamma, \Delta, \Gamma'; B)$$

and also a family of functions

$$\bar{\mu}_{\Gamma,\Delta,A,\Gamma';B} : M(\Delta; A) \times \mathcal{C}(\Gamma, A, \Gamma'; B) \to M(\Gamma, \Delta, \Gamma'; B)$$

satisfying three conditions. We use the following notation: we write $\Gamma \vdash_M t : A$ if $t \in M(\Gamma; A)$, and we write

$$\frac{\Delta \vdash t : A \quad \Gamma, x{:}\, A, \Gamma' \vdash_M u : B}{\Gamma, \Delta, \Gamma' \vdash_M t \overset{M}{\rightsquigarrow} x.\, u : B} \qquad \frac{\Delta \vdash_M t : A \quad \Gamma, x{:}\, A, \Gamma' \vdash u : B}{\Gamma, \Delta, \Gamma' \vdash_M t \overset{M}{\rightsquigarrow} x.\, u : B}$$

for $\mu_{\Gamma,A,\Delta,\Gamma';B}(t, u)$ and $\bar{\mu}_{\Gamma,A,\Delta,\Gamma';B}(t, u)$ respectively. The three conditions (in addition to LM1 and LM2) are

$$\frac{\Gamma \vdash_M t : A}{\Gamma \vdash_M t \overset{M}{\rightsquigarrow} x.\, x \equiv t : A} \tag{RM1}$$

$$\frac{\Gamma_3 \vdash_M t : A \quad \Gamma_2, x{:}\, A, \Gamma_2' \vdash u : B \quad \Gamma_1, y{:}\, B, \Gamma_1' \vdash v : C}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_2', \Gamma_1' \vdash_M (t \overset{M}{\rightsquigarrow} x.\, u) \overset{M}{\rightsquigarrow} y.\, v \equiv t \overset{M}{\rightsquigarrow} x.\, (u \rightsquigarrow y.\, v) : C} \tag{RM2}$$

$$\frac{\Gamma_3 \vdash t : A \quad \Gamma_2, x{:}\, A, \Gamma_2' \vdash_M u : B \quad \Gamma_1, y{:}\, B, \Gamma_1' \vdash v : C}{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_2', \Gamma_1' \vdash_M (t \overset{M}{\rightsquigarrow} x.\, u) \overset{M}{\rightsquigarrow} y.\, v \equiv t \overset{M}{\rightsquigarrow} x.\, (u \overset{M}{\rightsquigarrow} y.\, v) : C} \tag{RM3}$$

The last condition, RM3, specifies that the left (μ) and right ($\bar{\mu}$) structures associate with each other. Technically, our notion of right module can be seen as an instance of a general concept in locally indexed category theory and the self construction (see [23, §9.3.4, §9.3.6]).

Representations of right modules. For a fixed object R, the assignment $M(\Gamma; A) = \mathcal{C}(\Gamma, R; A)$ can be made into a right module in a straightforward way.

Definition 18. Let M be a right module for a premulticategory $\mathcal{C}$. A *representation of* M is an object R of $\mathcal{C}$ together with an isomorphism of right modules

$$\{\mathcal{C}(\Gamma, R; A) \cong M(\Gamma; A)\}_{\Gamma,A}$$

(An isomorphism of right modules is a family of bijections that respects the right module structure — that is, both μ and $\bar{\mu}$.)

A representation is always determined by an object R and an element of $M(R)$. This follows from a new variant of the Yoneda lemma, for which we need the following definition.

Definition 19. Let M be a right module for a premulticategory. An element $t \in M(\Gamma_1; A)$ is *central* if it satisfies the following equations:

$$\frac{\Gamma_2 \vdash u : B \quad \Delta_1, x{:}\, A, \Delta_2, y{:}\, B, \Delta_3 \vdash v : C}{\Delta_1, \Gamma_1, \Delta_2, \Gamma_2, \Delta_3 \vdash_M t \overset{M}{\rightsquigarrow} x.\, (u \rightsquigarrow y.\, v) \equiv u \overset{M}{\rightsquigarrow} y.\, (t \overset{M}{\rightsquigarrow} x.\, v) : C}$$

$$\frac{\Gamma_2 \vdash u : B \quad \Delta_1, y{:}\, B, \Delta_2, x{:}\, A, \Delta_3 \vdash v : C}{\Delta_1, \Gamma_2, \Delta_2, \Gamma_1, \Delta_3 \vdash_M t \overset{M}{\rightsquigarrow} x.\, (u \rightsquigarrow y.\, v) \equiv u \overset{M}{\rightsquigarrow} y.\, (t \overset{M}{\rightsquigarrow} x.\, v) : C}$$

Proposition 20. *Let M be a right module of a symmetric premulticategory $\mathcal{C}$ and let R be an object. To give a right module morphism $\{\mathcal{C}(\Gamma, R; A) \to M(\Gamma; A)\}_{\Gamma,A}$ is to give a central element in $M(();R)$.*

Corollary 21. *Let M be a right module of a symmetric premulticategory. The following data are equivalent.*

- *A representation for M (Def. 18).*
- *An object R and a central element $r \in M(();R)$ together with a family of functions $\rho_{\Gamma,A} : M(\Gamma; A) \to \mathcal{C}(\Gamma, R; A)$ notated*

$$\frac{\Gamma \vdash_M t : A}{\Gamma, x{:}\, R \vdash \rho(x, t) : A}$$

satisfying the following conditions:

$$\frac{\Gamma \vdash_M t : A}{\Gamma \vdash_M r \overset{M}{\rightsquigarrow} x.\, \rho(x, t) \equiv t : A} \tag{RR1}$$

$$\frac{\Gamma, x{:}\, R \vdash t : A}{\Gamma, x{:}\, R \vdash \rho(x, r \overset{M}{\rightsquigarrow} x.\, t) \equiv t : A} \tag{RR2}$$

Uniqueness of representations.

Proposition 22. *If $(R, r \in M(((), R))$ is a representation of M and $a \in M(();A)$ then there is a unique morphism $t{:}\,(x{:}\, R) \to A$ in $\mathcal{C}$ such that $- \vdash_M r \overset{M}{\rightsquigarrow} x.\, t \equiv a : A$. If a is central then so is t.*

In consequence, representations are unique up to unique isomorphism and the unique mediating isomorphism is central.

7.2.2 Examples: products and sums

- Given a list Δ, define a right module M_Δ by

$$M_\Delta(\Gamma; A) \overset{\text{def}}{=} \mathcal{C}(\Gamma, \Delta; A).$$

  A representation for M_Δ is a tensor product $\otimes\Delta$ in the sense of Section 3.1.

- Given a constructor context Δ (§4.1), define a right module M^Δ by $M^\Delta(\Gamma; A) \overset{\text{def}}{=} \prod_{c \in dom(\Delta)} \mathcal{C}(\Gamma, \Delta(c); A)$. A representation for M^Δ is a sum $\oplus\Delta$ in the sense of Section 4.1.

8. Premonoidal categories and monads

In Section 2 we defined premulticategories as a notion of model for an impure programming language with variables and let-binding. We argue that this is the primitive setting for studying call-by-value impure programming languages.

In Sections 3 and 4 we showed that products and sums can be characterized by a universal property, which means that they are unique up to unique isomorphism. In Section 6 we provided a similar universal property for function spaces.

We now justify our work by showing that it subsumes earlier axiomatizations of call-by-value programming languages: the premonoidal and Freyd categories of Power and Robinson [33] and the monadic models of Moggi [30].

- We show that a premonoidal category is essentially the same thing as a premulticategory with tensor products.
- We show that a Freyd category is essentially the same thing as a cartesian Freyd multicategory with tensor products.
- We show that a strong monad on a category with finite products is essentially the same thing as a cartesian Freyd multicategory with tensor products and function types with empty domain.
- We show that a λ_C-model is essentially the same thing as a cartesian Freyd multicategory with tensor products and all function types.

The last three facts can be deduced from the first one by building on earlier work by Levy, Power and Thielecke [24].

8.1 Premonoidal categories

We now recall the notion of premonoidal category. Before we begin, we give some concrete examples. The category of sets and functions is a premonoidal category, with the product of sets forming a premonoidal structure. Indeed, any monoidal category is a premonoidal category. But the motivating example is the Kleisli category for a monad. Let T be a monad on the category of sets. Recall that the Kleisli category of T has objects sets and that morphisms $A \to B$ are functions $A \to T(B)$. The product of sets induces a premonoidal structure on this Kleisli category which is typically not monoidal.

In Section 8.2 we will show how to convert a premulticategory to a premonoidal category. This will suggest a premonoidal category built from the syntax of a programming language.

Our definition of premonoidal category is streamlined by taking things up a level: we take advantage of a multicategory whose objects are themselves categories. We start by defining strict premonoidal categories, and move gradually to premonoidal categories.

8.1.1 A multicategory of categories

We organize the collection of all ordinary categories into a symmetric multicategory $\mathbf{Cat}_{\mathrm{m}}$. The objects of the multicategory $\mathbf{Cat}_{\mathrm{m}}$ are themselves ordinary categories, and so our contexts Γ are lists of categories. A morphism $F : \Gamma \to \mathcal{A}$ in $\mathbf{Cat}_{\mathrm{m}}$ is defined to be a mapping that is functorial in each argument. For instance, if $\Gamma = (\mathcal{B}, \mathcal{C})$ then for each pair of objects B and C, respectively from $\mathcal{B}$ and $\mathcal{C}$, an object $F(B, C)$ of $\mathcal{A}$ must be given, and this must extend to families of functors $F(-, C) : \mathcal{B} \to \mathcal{A}$ and $F(B, -) : \mathcal{C} \to \mathcal{A}$ (but not necessarily a functor $\mathcal{B} \times \mathcal{C} \to \mathcal{A}$).

This multicategory has tensor products [11] and this structure has been used in various areas, from rewriting theory [36] to bunched implications [31, Ex. 14].

8.1.2 Strict premonoidal categories

Before turning to premonoidal categories in general, we define strict premonoidal categories. A strict premonoidal category [33] is a monoid in the multicategory $\mathbf{Cat}_{\mathrm{m}}$. A monoid in a multicategory is an object $\mathcal{A}$ with morphisms $- \vdash i : \mathcal{A}$ and $x{:}\mathcal{A}, y{:}\mathcal{A} \vdash x \odot y : \mathcal{A}$ satisfying the monoid laws:

$$\begin{array}{c} y{:}\mathcal{A} \vdash i \rightsquigarrow x.\, x \odot y \equiv y : \mathcal{A} \quad x{:}\mathcal{A} \vdash i \rightsquigarrow y.\, x \odot y \equiv x : \mathcal{A} \\ x, y, z : \mathcal{A} \vdash x \odot y \rightsquigarrow w.\, w \odot z \equiv y \odot z \rightsquigarrow w.\, x \odot w : \mathcal{A}. \end{array} \tag{6}$$

In $\mathbf{Cat}_{\mathrm{m}}$, the morphism $i : () \to \mathcal{A}$ is the same thing as an object of the category $\mathcal{A}$. The construction $\odot : (\mathcal{A}, \mathcal{A}) \to \mathcal{A}$ is not a functor $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ from the product of categories, but rather a construction that is functorial in each argument.

8.1.3 Central morphisms

Just as central morphisms play a crucial role in premulticategories, they also play an important role in the theory of premonoidal categories.

We begin by recalling that a *binoidal category* [33] is a category $\mathcal{A}$ together with a morphism $\odot : (\mathcal{A}, \mathcal{A}) \to \mathcal{A}$ in the multicategory $\mathbf{Cat}_{\mathrm{m}}$. For each object C and each morphism $f{:}A \to B$ we have morphisms $(f \odot C) : A \odot C \to B \odot C$ and $(C \odot f) : C \odot A \to C \odot B$. Given morphisms $f{:}A \to B$, $g{:}C \to D$ in $\mathcal{A}$ we thus have two morphisms $A \odot C \to B \odot D$:

$$A \odot C \xrightarrow{f \odot C} B \odot C \xrightarrow{B \odot g} B \odot D \quad A \odot C \xrightarrow{A \odot g} A \odot D \xrightarrow{f \odot D} B \odot D$$

If these are equal, we say that f and g commute. A morphism in a binoidal category is *central* if it commutes with all morphisms.

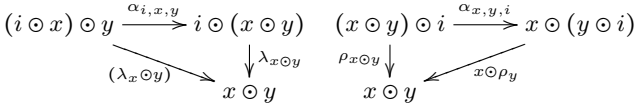
8.1.4 Premonoidal categories

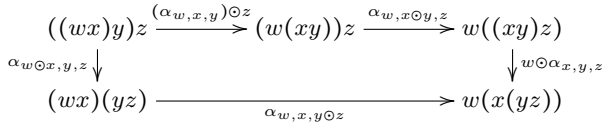
In the multicategory $\mathbf{Cat}_{\mathrm{m}}$ the sets of morphisms $(\mathcal{A}_1 \dots \mathcal{A}_n) \to \mathcal{B}$ can themselves be considered as categories. Given morphisms $F, G : (\mathcal{A}_1 \dots \mathcal{A}_n) \to \mathcal{B}$, a 2-cell $\alpha{:}F \to G$ comprises a family of morphisms $\alpha_{\vec{A}}{:}F(\vec{A}) \to G(\vec{A})$ in $\mathcal{B}$ indexed by lists of objects $(A_1 \dots A_n)$ from categories $\mathcal{A}_1 \dots \mathcal{A}_n$ respectively, that is natural in each argument. With this in mind, we can weaken (6) by replacing the equalities with central natural isomorphisms.

Definition 23 ([33])**.** A *premonoidal category* is a binoidal category $(\mathcal{A}, \odot : (\mathcal{A}, \mathcal{A}) \to \mathcal{A})$ with an object i and central natural isomorphisms

$$\lambda_y : i \odot y \cong y \quad \rho_x : x \odot i \cong x \quad \alpha_{x,y,z} : (x \odot y) \odot z \cong x \odot (y \odot z)$$

that make the following triangle and pentagon laws hold:





Note that while the centrality of λ, ρ, and α is reasonable from a pragmatic perspective, it is ad hoc in that it does not come from the analysis of $\mathbf{Cat}_{\mathrm{m}}$ as a 2-multicategory.

8.1.5 Coherence

The definition of premonoidal category is not based on universal properties and so, a priori, there is nothing canonical about λ, ρ and α. A canonical status is given by the coherence theorem (Proposition 25) which characterizes the morphisms that can be built from λ, ρ and α.

Definition 24. Let $\mathbb{A}$ be a set. An *object-string* is generated from the grammar

$$S, T ::= i \mid (S \odot T) \mid A \ \ (A \in \mathbb{A}).$$

such that each element of the set $\mathbb{A}$ appears exactly once. An *isomorphism-string* between object-strings, $f : S \to T$, is a string that is built using composition, $\odot$, λ, ρ, α and their inverses, subject to the obvious well-formedness condition.

Proposition 25 ([33])**.** *Let S and T be object-strings. All isomorphism strings $f, g : S \cong T$ are equal when interpreted in any premonoidal category.*

Coherence theorems like this form an important part of the category theory literature. Nonetheless we contend that it is more desirable that the structure arises a priori from universal properties, which is the content of our main theorem (Theorem 26). For one thing, the universal properties place the structure closer to programming language syntax.

8.2 Main theorem

Our main theorem provides a connection between premulticategories with tensor products and premonoidal categories. It is a variation of the established connection between multicategories with tensor products and monoidal categories [14, 22].

Theorem 26. *Let $\mathbb{A}$ be a set. The following data are equivalent.*

1. *A premulticategory with tensor products whose objects are $\mathbb{A}$.*
2. *A premonoidal category whose objects are $\mathbb{A}$.*

This section is devoted to the proof of this theorem. We begin with the following straightforward property of tensor products in premulticategories.

Proposition 27. *A premulticategory has all tensor products if and only if it has tensor products of the empty list and of every two element list.*

Of course, it is sometime useful to have explicit n-ary product types, but the universal property tells us that any two implementations are canonically isomorphic.

From a premulticategory to a premonoidal category. Given a premulticategory $\mathcal{C}$ with tensor products, we define a premonoidal category. A morphism $f\colon A \to B$ in the premonoidal category is a morphism $f\colon (A) \to B$ in $\mathcal{C}$. Composition and identities are immediately derived from the premulticategory too.

The unit of the premonoidal structure is $\otimes()$, and the premonoidal tensor $A \odot C$ is the tensor product $\otimes(A, C)$ in the premulticategory. This can be made into a binoidal structure: given $x\colon A \vdash f : B$ and $y\colon C \vdash g : D$, let

$$f \odot C \stackrel{\text{def}}{=} z\colon \otimes(A,C) \vdash z \multimap \langle x, y\rangle.\, f \rightsquigarrow x'.\, \langle x', y\rangle : \otimes(B,C)$$

$$A \odot g \stackrel{\text{def}}{=} z\colon \otimes(A,C) \vdash z \multimap \langle x, y\rangle.\, g \rightsquigarrow y'.\, \langle x, y'\rangle : \otimes(A,D)$$

One verifies that this structure is functorial (preserves composition and identities) by some algebraic manipulation using the axioms for premulticategories, which we omit for brevity.

Having the binoidal structure in place we can note that the two notions of centrality are related:

Proposition 28. *A morphism $f\colon (A) \to B$ in a premulticategory is central (§2.1) if and only if it is central in the induced binoidal category (§8.1.3).*

(This is deduced by algebraic manipulation.)

The coherence isomorphisms in the premonoidal category are defined by

$$\lambda_B \stackrel{\text{def}}{=} z\colon (\otimes(\otimes(), B)) \vdash z \multimap \langle x, y\rangle.\, x \multimap \langle\rangle.\, y : B$$

$$\rho_A \stackrel{\text{def}}{=} z\colon (\otimes(A, \otimes())) \vdash z \multimap \langle x, y\rangle.\, y \multimap \langle\rangle.\, x : A$$

$$\alpha_{A,B,C} \stackrel{\text{def}}{=} xyz\colon (\otimes(\otimes(A,B),C)) \vdash xyz \multimap \langle xy, z\rangle.\, xy \multimap \langle x, y\rangle.\, \langle x, \langle y, z\rangle\rangle : (\otimes(A, \otimes(B,C)))$$

Again, some algebraic manipulation is needed to show that these morphisms are isomorphisms, that they are central, that they are natural, and that they satisfy the coherence diagrams.

From a premonoidal category to a premulticategory. We begin with some remarks about contexts in a premonoidal category. A list of objects determines an object $|\Gamma|$ of the premonoidal category:

$$|A_1, A_2, A_3, \ldots, A_n| \stackrel{\text{def}}{=} ((((A_1 \odot A_2) \odot A_3) \cdots \odot A_n) \odot I.$$

In particular, $|\varnothing| = I$ and $|A| = A \odot I$. We often need to concatenate lists. The coherence result (Prop. 25) gives us a canonical central isomorphism between $|\Gamma, \Delta|$ and $|\Gamma| \odot |\Delta|$. (The objects are typically not identical.)

Given a premonoidal category, we define a premulticategory with the same objects. A morphism $\Gamma \to A$ in the premulticategory is a morphism $|\Gamma| \to A$ in the premonoidal category. The identity morphisms $(A) \to A$ in the premulticategory are the right identity isomorphisms $A \odot I \to A$ in the premonoidal category. For composition, given $t\colon |\Delta| \to A$ and $u\colon |\Gamma, x\colon A, \Gamma'| \to B$ we let $(t \rightsquigarrow x.\, u)$ be the following composite:

$$|\Gamma, \Delta, \Gamma'| \cong (|\Gamma| \odot |\Delta|) \odot |\Gamma'| \xrightarrow{(|\Gamma| \odot t) \odot |\Gamma'|} (|\Gamma| \odot A) \odot |\Gamma'| \cong |\Gamma, A, \Gamma'| \xrightarrow{u} B$$

where the unlabelled isomorphisms are the canonical coherence isomorphisms in the premonoidal category.

The identity and associativity laws for premulticategories follow from the centrality and canonicity of the coherence isomorphisms.

Proposition 29. *A morphism $t\colon |\Gamma| \to A$ in a premonoidal category is central in the premonoidal category if and only if it is central in the induced premulticategory.*

The tensor product of the premulticategory is straightforward: for any list Δ, let $\otimes\Delta \stackrel{\text{def}}{=} |\Delta|$. The universal morphism $\Delta \to |\Delta|$ is the identity morphism. Given $t\colon |\Gamma, \Delta, \Gamma'| \to B$ we let

$$y \multimap \langle \vec{x}\rangle.\, t \stackrel{\text{def}}{=} |\Gamma, y\colon \otimes\Delta, \Gamma'| \cong |\Gamma, \Delta, \Gamma'| \xrightarrow{t} B.$$

Again, the unlabelled isomorphism is the canonical coherence isomorphism. The two laws for tensor products follow from the centrality and canonicity of the coherence isomorphisms.

Equivalence. In the statement of Theorem 26, when we say that the two notions are equivalent, we do not mean that they are exactly the same. Rather, we mean that if we begin with a premulticategory $\mathcal{C}$, build a premonoidal category, and then recover a premulticategory from the premonoidal category, we recover a premulticategory that is canonically isomorphic to $\mathcal{C}$. Conversely, if we begin with a premonoidal category $\mathcal{C}$, and then build a premulticategory out of it, and then recover a premonoidal category from the premulticategory, we recover a premonoidal category that is isomorphic to $\mathcal{C}$. In this sense, the two notions are equivalent.

This concludes our proof of Theorem 26.

8.3 Corollaries of the main theorem

We conclude this paper by building on the main theorem (Theorem 26) to recover new universal characterizations of various models from the literature.

Structural laws. Power and Robinson [33] define a *symmetric premonoidal category* to be a premonoidal category with a central natural isomorphism $\{s_{A,B}\colon A \odot B \to B \odot A\}_{A,B}$ satisfying coherence conditions.

Proposition 30. *To give a symmetric premonoidal category is to give a symmetric premulticategory (Def. 3) with tensor products.*

Sums and predistributive categories. Power and Robinson [33] define a *predistributive category* to be a premonoidal category with finite coproducts in which the functors $A \odot (-)$ and $(-) \odot A$ preserve finite coproducts for all objects A.

Proposition 31. *To give a predistributive category is to give a premulticategory with all tensor products and sums (§4).*

Multicategories and monoidal categories. We turn to the well-known representation theorem for multicategories [14, 22]. Recall that a multicategory is a premulticategory in which all morphisms are central (§2.1). Similarly, a monoidal category is a premonoidal category (§8.1.4) in which all morphisms are central (§8.1.3).

Proposition 32 (c.f. [14])**.**

1. *A premulticategory with tensor products is a multicategory if and only if the corresponding premonoidal category is a monoidal category.*
2. *To give a symmetric multicategory (Def. 3) with tensor products is to give a symmetric monoidal category.*
3. *To give a cartesian multicategory (Def. 4) with tensor products is to give a category with finite products.*

Freyd multicategories and Freyd categories. We now connect our analysis of values (§5) with the notion of Freyd category [23, 24, 34]. A Freyd category is defined to be an identity-on-objects premonoidal functor $\mathcal{V} \to \mathcal{C}$ from a category $\mathcal{V}$ with cartesian products to a premonoidal category $\mathcal{C}$.

Proposition 33. *To give a Freyd category is to give a cartesian Freyd multicategory (§5.3.3) that has tensor products (§5.3.2).*

Strong monads and function spaces. We now use the known connections between closed Freyd categories and monads [24, 34] to provide a universal characterization for Moggi's monads in computation [30] .

Proposition 34. *Let $\mathcal{V}$ be a symmetric multicategory with all tensors. Let $Mon(\mathcal{V})$ be the corresponding symmetric monoidal category (Proposition 32). The following data are equivalent:*

1. *A strong monad on $Mon(\mathcal{V})$.*
2. *A symmetric Freyd multicategory $\mathcal{V} \to \mathcal{C}$ that has function spaces with empty domain $(() \Rightarrow A)$ (§6.1).*

Proof outline. From 1 to 2, let $\mathcal{C}$ be the Kleisli premulticategory for the monad (§2.2). From 2 to 1, a monad on $\mathcal{V}$ is given by the construction $T(A) \stackrel{\text{def}}{=} (() \Rightarrow A)$. □

Moggi [29] models a call-by-value functional programming language by a λ_C-model, which is a strong monad on a category with products and with certain 'Kleisli' function spaces. We now give this a universal status by exhibiting an equivalent definition using premulticategories.

Proposition 35. *Let $\mathcal{V}$ be a cartesian multicategory with tensor products. Let $Mon(\mathcal{V})$ be the corresponding category with finite products (Proposition 32). The following data are equivalent.*

1. *A λ_C-model structure for $Mon(\mathcal{V})$.*
2. *A cartesian Freyd multicategory $\mathcal{V} \to \mathcal{C}$ with tensor products and function spaces.*

9. Concluding remarks

We have given universal properties for type constructions in impure programming languages: products, sums, and function spaces. We have done this using the novel notion of premulticategory, which is a basic equational account of impure computation. We have shown that monads and premonoidal categories can be understood from this point of view and hence given a canonical status.

Acknowledgments

S Staton partly supported by the Isaac Newton Trust and ERC Grant ECSYM. PB Levy supported by EPSRC Advanced Research Fellowship EP/E056091/1.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

[2] T. Altenkirch, J. Chapman, and T. Uustalu. Monads need not be endofunctors. In *FOSSACS'10*, pages 297–311, 2010.

[3] R. Atkey. What is a categorical model of arrows? In *Proc. MSFP'08*, pages 19–37, 2011.

[4] P. N. Benton and P. Wadler. Linear logic, monads and the lambda calculus. In *Proc. LICS'96*, pages 420–431, 1996.

[5] A. Burroni. T-categories (catégories dans un triple). *Cahiers Topologie Géom. Différentielle*, 12(3):215–312, 1971.

[6] A. Burroni. Higher dimensional word problem. In *Proc. CTCS'91*, pages 94–105, 1991.

[7] P.-L. Curien. Operads, clones, and distributive laws. In *Operads and Universal Algebra*, 2010.

[8] J. Egger, R. E. Møgelberg, and A. Simpson. Enriching an effect calculus with linear types. In *Proc. CSL'09*, pages 240–254, 2009.

[9] M. P. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proc. PPDP'02*, pages 26–37, 2002.

[10] C. Flanagan, A. Sabry, B. F. Duba, and M. Felleisen. The essence of compiling with continuations. In *Proc. PLDI'93*, pages 237–247, 1993.

[11] F. Foltz, C. Lair, and G. M. Kelly. Algebraic categories with few monoidal biclosed structures or none. *J. Pure and Applied Algebra*, 17:171–177, 1980.

[12] C. Führmann. Direct models of the computational lambda-calculus. In *Proc. MFPS XV*, pages 147–172, 1999.

[13] H. Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Proc. CSL'94*, pages 61–75, 1994.

[14] C. Hermida. Representable multicategories. *Advances in Mathematics*, 151:164–225, 2000.

[15] M. Hyland. Multicategories in and around algebra and logic. Invited talk, TACL'09. Slides available from the author's home page, 2009.

[16] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69(1):73 – 106, 1994.

[17] B. Jacobs and I. Hasuo. Freyd is Kleisli, for arrows. In *Proc. MSFP'06*, 2006.

[18] A. Jeffrey. Premonoidal categories and a graphical view of programs. Unpublished, 1997.

[19] A. Kock. Monads on symmetric monoidal closed categories. *Archiv der Math.*, 21:1–10, 1970.

[20] Y. Lafont. Towards an algebraic theory of Boolean circuits. *J. Pure Appl. Algebra*, 184(2–3):257–310, 2003.

[21] J. Lambek. Deductive systems and categories II. In *Category theory, homology theory and their applications*, volume 86 of *LNM*, pages 76–122. Springer, 1969.

[22] T. Leinster. *Higher operads, higher categories*. CUP, 2004.

[23] P. B. Levy. *Call-by-push-value*. Springer, 2004.

[24] P. B. Levy, J. Power, and H. Thielecke. Modelling environments in call-by-value programming languages. *Inform. Comput.*, 2003.

[25] F. Linton. Autonomous equational categories. *Indiana Univ. Math. J.*, 15:637–642, 1966.

[26] G. McCusker. A fully abstract relational model of syntactic control of interference. In *Proc. CSL'02*, pages 247–261, 2002.

[27] P. Melliès and N. Tabareau. Linear continuations and duality. `hal.archives-ouvertes.fr/hal-00339156`, 2008.

[28] R. E. Møgelberg and S. Staton. Linearly-used state in models of call-by-value. In *CALCO'11*, pages 298–313, 2011.

[29] E. Moggi. Computational lambda-calculus and monads. In *LICS'89*, pages 14–23, 1989.

[30] E. Moggi. Notions of computation and monads. *Inform. Comput.*, 93 (1), 1991.

[31] P. W. O'Hearn. On bunched typing. *J. Funct. Program.*, 13(4):747–796, 2003.

[32] J. Power. Premonoidal categories as categories with algebraic structure. *Theor. Comput. Sci.*, 278(1–2):303–321, 2002.

[33] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Math. Struct. Comput. Sci.*, 7(5):453–468, 1997.

[34] J. Power and H. Thielecke. Closed Freyd- and kappa-categories. In *Proc. ICALP'99*, 1999.

[35] U. S. Reddy. Global state considered unnecessary: An introduction to object-based semantics. *Lisp and Symbolic Computation*, 9(1):7–76, 1996.

[36] J. G. Stell. Modelling term rewriting systems by sesqui-categories. Technical Report TR94-02, Keele University, 1994.

Addendum (December 2012)

We have some remarks about cartesian premulticategories (§2.3). It seems appropriate to make the following definitions and changes:

- Regarding tensor products (§3.1): a tensor product in a cartesian premulticategory is *cartesian* when the structure morphism $\Delta \to \otimes\Delta$ is discardable and copyable as well as central.
- Regarding sums (§4.1): a sum in a cartesian premulticategory is *cartesian* when the injections $\Delta(\mathrm{c}) \to \oplus\Delta$ are discardable and copyable as well as central.
- The definition of cartesian Freyd multicategory (§5.3.3) should be changed, to require the identity-on-objects morphism $return : \mathcal{V} \to \mathcal{C}$ to preserve discardability and copyabilty as well as centrality. That is, for every morphism v in $\mathcal{V}$, the morphism $return(v)$ in $\mathcal{C}$ should be discardable, copyable and central.

Our proposed change to the definition of cartesian Freyd multicategory is vacuous when it has tensor products, since we have the following lemma.

Lemma. *Consider an identity-on-objects morphism between cartesian premulticategories with tensors. If it preserves tensors then it also preserves discardability and copyability.*

Moreover, in a cartesian multicategory, all tensor products and sums are cartesian since all morphisms are discardable, copyable and central. As a result, the theorems in Section 8 are true with or without the modification.