Scientific Annals of Computer Science vol. 24 (2), 2014, pp. 325–368
doi: 10.7561/SACS.2014.2.325

Modeling Simply-Typed Lambda Calculi in the Category of Finite Vector Spaces

Benoît Valiron[1], Steve Zdancewic[2]

Abstract

In this paper we use finite vector spaces (finite dimension, over finite fields) as a non-standard computational model of linear logic. We first define a simple, finite PCF-like lambda-calculus with booleans, and then we discuss two finite models, one based on finite sets and the other on finite vector spaces. The first model is shown to be fully complete with respect to the operational semantics of the language, while the second model is not. We then develop an algebraic extension of the finite lambda calculus and study two operational semantics: a call-by-name and a call-by-value. These operational semantics are matched with their corresponding natural denotational semantics based on finite vector spaces. The relationship between the various semantics is analyzed, and several examples based on Church numerals are presented.

Keywords: finite vector spaces, finite sets, algebraic lambda-calculus, Kleisli category, Eilenberg-Moore category.

1 Introduction

A standard way to study properties of functional programming languages is via denotational semantics. A denotational semantics (or model) for a

[1]PPS, UMR 7126, Univ Paris Diderot, Sorbonne Paris Cité, F-75205 Paris, France and I2M, UMR 7373, Université Aix-Marseille, F-13288 Marseille, France; Partially supported by the ANR project ANR-2010-BLAN-021301 LOGOI; E-mail: benoit.valiron@monoidal.net

[2]University of Pennsylvania, Philadelphia, US; Funded in part by NFS grant CCF-1421193; E-mail: stevez@cis.upenn.edu

language is a mathematical representation of its programs [35], and the typical representation of a term is a function whose domain and codomain are the data-types of input and output. This paper is concerned with a non-standard class of models based on finite vector spaces.

The two languages we will consider are based on PCF [29] – the laboratory mouse of functional programming languages. PCF comes as an extension of simply-typed lambda-calculus with basic types, term constructs, and can be easily extended to handle specific effects. Here, we define $\mathbf{PCF}_f$ as a simple lambda-calculus with pairs and booleans, and $\mathbf{PCF}_f^{alg}$, its extension to linear combinations of terms. This additional algebraic structure will essentially be regarded as a particular kind of side effect: in this paper, we shall analyze a call-by-name and a call-by-value semantics for this side effect in the context of the extended language $\mathbf{PCF}_f^{alg}$.

There has been much work and progress on various denotational models of PCF, often with the emphasis on trying to achieve full abstraction. The seminal works are using term models [23], cpos [24] or game semantics [1], while more recent works use quantitative semantics of linear logic [13] and discuss probabilistic extensions [11] or non-determinism [7].

Any category that models a PCF language must be cartesian closed to account for internal morphisms and pairing. An expressive class of cartesian closed categories can be made of models of linear logic, by considering the (co)Kleisli category stemming from the exponential modality "!". Although the models that are usually considered are rich and expressive [10, 11, 7], "degenerate" models nevertheless exist [26, 17]. The consequences of the existence of such models of PCF have not been explored thoroughly.

In this paper, we consider two related finitary categories: the category of finite sets and functions **FinSet** and the category of finite vector spaces and linear functions **FinVec**, i.e. finite-dimensional vector spaces over a finite field. The adjunction between these two categories is known in the folklore to give a model of linear logic [25], but its computational behavior has not been studied until now.

The primary motivation for this work is simple curiosity: What do the vectors interpreting lambda calculus terms look like? Though not the focus of this paper, one could imagine that the ability to encode programming language constructs in the category of vector spaces might yield interesting applications. For instance, a Matlab-like programming language that natively supports rich datatypes and first-class functions, all with the same semantic

status as "vectors" and "matrices." A benefit of this design would be the possibility of "typed" matrix programming, or perhaps sparse matrix representations based on lambda terms and their semantics. The algebraic lambda calculus sketched in this paper is a (rudimentary) first step in this direction. Conversely, one could imagine applying techniques from linear algebra to lambda calculus terms. For instance, finite fields play a crucial role in cryptography, which, when combined with programming language semantics, might lead to new algorithms for homomorphic encryption.

The goal here is more modest, however. The objective of the paper is to study how the adjunction between the categories **FinSet** and **FinVec** fits with respect to the language $\mathbf{PCF}_f$ and its algebraic extension $\mathbf{PCF}_f^{alg}$.

In particular, we consider the usual three (gradually more constraining) properties: *adequacy*, *full abstraction* and *full completeness*. A semantics is *adequate* if whenever terms of some observable type (`Bool` for example) are operationally equivalent then their denotations match. An adequate semantics is "reasonable" in the sense that programs and their representations match at ground type. The semantics is *fully abstract* if operational equivalence and equality of denotation are the same thing for all types. In this situation, programs and their denotations are in correspondence at all types, but the model can contain non-representable elements. Finally, the semantics is *fully complete* if moreover, every element in the image of a type A is representable by a term in the language. With such a semantics, the set of terms and its mathematical representation are fully correlated. If a semantics is fully complete, then it is fully abstract and if it is fully abstract, then it is adequate.

1.1 Results

This paper presents the first account of the interpretation of two PCF-like languages in finite vector spaces. More specifically, we show in details that the category of finite sets **FinSet** forms a fully complete model for the language $\mathbf{PCF}_f$. and that the coKleisli category $\mathbf{FinVec}_!$ is adequate but not fully-abstract: this model has too many points compared to what one can express in the language. We present several examples based on the usual lambda-calculus encoding of Church numerals to illustrate the model. We then present an algebraic extension $\mathbf{PCF}_f^{alg}$ of $\mathbf{PCF}_f$ and show that $\mathbf{FinVec}_!$ forms a fully complete model for this extension with a call-by-name operational semantics. We then develop a corresponding call-by-value

operational semantics and present a fully-complete model for it using an extension $\mathbf{FreeCoalg}^{+}$ of the Eilenberg-Moore category of the comonad "!", provided that the field is $\mathbb{F}_2$.

We then explore the relationships between the language $\mathbf{PCF}_f$, the call-by-name variant $\mathbf{PCF}_f^{alg}$, and their categorical models **FinSet** and $\mathbf{FinVec}_!$. Finally, we discuss the relationship between the call-by-name and call-by-value operational semantics of $\mathbf{PCF}_f^{alg}$ using the "thunk" construction, and show how this construction transposes to a relation between the category $\mathbf{FinVec}_!$ and the category $\mathbf{FreeCoalg}^{+}$.

This paper is an extended version of a former paper [33] which appeared in the proceedings of ICTAC 2014. This paper extends the proceedings version with an extended discussion of **FinVec** as a model of linear logic, the call-by-value operational semantics for $\mathbf{PCF}_f^{alg}$, its relation to call-by-name, and the category $\mathbf{FreeCoalg}^{+}$.

1.2 Related Works

In the literature, finite models for lambda-calculi are common. For example, Hillebrand analyzes databases as finite models of the simply-typed lambda calculus [16]. Otherwise, Salvati presents a model based on finite sets [27], while Selinger presents models based on finite posets [30]. Finally, Solovev [31] relates the equational theory of cartesian closed categories with the category of finite sets.

More general than vector spaces, various categories of modules over semirings, as standard models of linear logic have been studied as computational models: sets and relations [7], finiteness spaces [10], probabilistic coherent spaces [11], *etc.*

As models of linear logic, finite vector spaces are folklore [25] and appear as side examples of more general constructions such as Chu spaces [26] or glueing [17]. Computationally, Chu spaces (and then to some extent finite vector spaces) have been used in connection with automata [26]. Finally, recently finite vector spaces have also been used as a toy model for quantum computation (see e.g. [28, 18]).

Algebraic lambda-calculi, that is, lambda-calculi with a vectorial structure have been first defined in connection with finiteness spaces [12, 34]. Another approach [4, 2] arrives at a similarly structured language by way of quantum computation. The former approach is call-by-name while the latter

is call-by-value. A general, categorical semantics has been developed [32] but no other concrete models have been considered.

1.3 Plan of the Paper

The paper is shaped as follows. Section 2 presents a finite PCF-style language $\mathbf{PCF}_f$ with pairs and booleans, together with its operational semantics. Section 3 presents the category **FinSet** of finite sets and functions, and discusses its properties as a model of the language $\mathbf{PCF}_f$. Section 4 describes finite vector spaces and Section 4.3 shows how to build a model of linear logic from the adjunction with finite sets. In particular, the section presents the Kleisli category $\mathbf{FinVec}_!$, the Eilenberg-Moore category **FreeCoalg**, and its extension $\mathbf{FreeCoalg}^+$ used later to model call-by-value. Section 4.4 discusses the Kleisli category $\mathbf{FinVec}_!$ as a model of $\mathbf{PCF}_f$ and presents some examples based on Church numerals. Section 5 explains how to extend the language with algebraic structures: this gives the language $\mathbf{PCF}_f^{alg}$. Section 5.2 exposes a call-by-name semantics for which $\mathbf{FinVec}_!$ is fully complete, while Section 5.3 presents a call-by-value semantics for which $\mathbf{FreeCoalg}^+$ is fully complete, Finally, Section 6 discusses various related aspects. Section 6.1 looks at the relationship between $\mathbf{PCF}_f$ and its algebraic extension $\mathbf{PCF}_f^{alg}$ with call-by-name semantics, Section 6.2 analyzes the thunk construction for $\mathbf{PCF}_f^{alg}$ in the light of the denotations $\mathbf{FinVec}_!$ and $\mathbf{FreeCoalg}^+$, and finally Section 6.3 discusses generalizations of fields.

2 A Finite PCF-style Lambda Calculus

We start with a minimal, finite PCF-style language with pairs and booleans called $\mathbf{PCF}_f$. It is intrinsically typed (i.e. Church-style—all subterms are defined with their type) and defined as follows.

$$\begin{array}{rcl} M, N, P & ::= & x \mid \lambda x.M \mid MN \mid \pi_l(M) \mid \pi_r(M) \mid \langle M, N \rangle \mid \star \mid \\ & & \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P \mid \mathtt{let}\, \star = M\, \mathtt{in}\, N, \\ A, B & ::= & \mathtt{Bool} \mid A \to B \mid A \times B \mid \mathbf{1}. \end{array}$$

Values, including "lazy" pairs (that is, pairs of arbitrary terms, as opposed to pairs of values), are inductively defined by

$$U, V \quad ::= \quad x \mid \lambda x.M \mid \langle M, N \rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff}.$$

$$\frac{}{\Delta, x : A \vdash x : A} \quad \frac{}{\Delta \vdash \star : \mathbf{1}} \quad \frac{}{\Delta \vdash \mathtt{tt}, \mathtt{ff} : \mathtt{Bool}}$$

$$\frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \to B} \quad \frac{\Delta \vdash M : A_l \times A_r}{\Delta \vdash \pi_i(M) : A_i}$$

$$\frac{\Delta \vdash M : A \to B \quad \Delta \vdash N : A}{\Delta \vdash MN : B} \qquad \frac{\Delta \vdash M : A \quad \Delta \vdash N : B}{\Delta \vdash \langle M, N \rangle : A \times B}$$

$$\frac{\Delta \vdash M : \mathtt{Bool} \quad \Delta \vdash N_1, N_2 : A}{\Delta \vdash \mathtt{if}\, M\, \mathtt{then}\, N_1\, \mathtt{else}\, N_2 : A} \qquad \frac{\Delta \vdash M : \mathbf{1} \quad \Delta \vdash N : A}{\Delta \vdash \mathtt{let} \star = M\, \mathtt{in}\, N : A}$$

Table 1: Typing rules for the language $\mathbf{PCF}_f$.

The terms consist of the regular lambda-terms, plus specific term constructs. The terms `tt` and `ff` respectively stand for the booleans True and False, while `if - then - else` is the boolean test operator. The type `Bool` is the type of the booleans. The term $\star$ is the unique value of type $\mathbf{1}$, and $\mathtt{let} \star = - \mathtt{in} -$ is the evaluation of a "command", that is, of a term evaluating to $\star$. The term $\langle -, - \rangle$ is the pairing operation, and π_l and π_r stand for the left and right projections. The type operator $(\times)$ is used to type pairs, while $(\to)$ is used to type lambda-abstractions and terms representing functions.

A typing judgment is a sequent of the form $\Delta \vdash M : A$, where Δ is a typing context: a collection of typed variables $x : A$. A typing judgment is said to be *valid* when there exists a valid typing derivation built out of the rules in Table 1.

Note that since terms are intrinsically typed, for any valid typing judgment there is only one typing derivation. Again because the terms are intrinsically typed, by abuse of notation when the context is clear we use $M : A$ instead of $\Delta \vdash M : A$.

Notation 1. When considering typing judgments such as $x : A \vdash M : B$ and $y : B \vdash N : C$, we use categorical notation to denote the composition: $M; N$ stands for the (typed) term $x : A \vdash (\lambda y.N)M : C$, also written as $A \xrightarrow{M} B \xrightarrow{N} C$. We also extend pairs to finite products as follows: $\langle M_1, M_2, \dots \rangle$ is the term $\langle M_1, \langle M_2, \langle \dots \rangle \rangle \rangle$. Projections are generalized

$$\begin{array}{rcl@{\qquad}rcl} (\lambda x.M)N & \to & M[N/x] & \pi_l\langle\, M, N\,\rangle & \to & M \\ \mathtt{let}\star = \star\,\mathtt{in}\,M & \to & M & \pi_r\langle\, M, N\,\rangle & \to & N \end{array}$$

$$\mathtt{if}\,\mathtt{tt}\,\mathtt{then}\,M\,\mathtt{else}\,N \to M \qquad \mathtt{if}\,\mathtt{ff}\,\mathtt{then}\,M\,\mathtt{else}\,N \to N$$

$$\frac{M \to M'}{MN \to M'N} \qquad \frac{M \to M'}{\pi_l(M) \to \pi_l(M')} \qquad \frac{M \to M'}{\pi_r(M) \to \pi_r(M')}$$

$$\frac{M \to M'}{\mathtt{if}\,M\,\mathtt{then}\,N_1\,\mathtt{else}\,N_2 \to \mathtt{if}\,M'\,\mathtt{then}\,N_1\,\mathtt{else}\,N_2}$$

$$\frac{M \to M'}{\mathtt{let}\star = M\,\mathtt{in}\,N \to \mathtt{let}\star = M'\,\mathtt{in}\,N}$$

Table 2: Small-step semantics for the language $\mathbf{PCF}_f$.

to finite products with the notation π_i projecting the i-th coordinate of the product. Types are extended similarly: $A \times \cdots \times A$, also written as $A^{\times n}$, is defined as $A \times (A \times (\cdots))$.

2.1 Small Step Semantics

The language is equipped with a call-by-name reduction strategy: a term M reduces to a term M', denoted with $M \to M'$, when the reduction can be derived from the rules of Table 2. We use the notation $\to^*$ to refer to the reflexive transitive closure of $\to$. The language then verify some standard properties, as stated in Lemma 3

Lemma 2 (Substitution)**.** *If we have* $\Delta, x : A \vdash M : B$ *and* $\Delta \vdash N : B$*, then the judgement* $\Delta \vdash M[N/x] : B$ *is valid.*

Proof. The proof is standard: it is done by structural induction on the derivation of the judgement $\Delta, x : A \vdash M : B$. □

Lemma 3. *The following statements are correct.*

1. *The only closed value of type* $\mathbf{1}$ *is* $\star$ *and the only closed values of type* `Bool` *are* `tt` *and* `ff`*.*

2. *For any well-typed closed term $M : A$, either M is a value or M reduces to some term N.*

3. *If $\Delta \vdash M : A$ and if M reduces to N, then $\Delta \vdash N : A$ is also valid.*

4. *The language $\mathbf{PCF}_f$ is strongly normalizing.*

Proof. Each item is proven separately: (1) By case inspection. (2) By structural induction on M: either it is already a value, or there is a redex inside M. (3) By structural induction on the derivation of $\Delta, x : A \vdash M : B$, using Lemma 2 to take care of the redex $(\lambda x.M)N$. (4) The language $\mathbf{PCF}_f$ is strongly normalizing since it can be easily encoded in the strongly normalizing language system F [15]. □

2.2 Operational Equivalence

We define the operational equivalence on terms in a standard way. A *context* $C[-]$ is a "term with a hole", that is, a term consisting of the following grammar:

$$\begin{aligned} C[-] ::= {} & x \mid [-] \mid \lambda x.C[-] \mid C[-]N \mid MC[-] \mid \pi_l(C[-]) \mid \pi_r(C[-]) \mid \\ & \langle\, C[-], N \,\rangle \mid \langle\, M, C[-] \,\rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}\, C[-]\, \mathtt{then}\, N\, \mathtt{else}\, P \mid \\ & \mathtt{if} M \mathtt{then}\, C[-]\, \mathtt{else}\, P \mid \mathtt{if}\, M\, \mathtt{then}\, N \mathtt{else}\, C[-] \mid \\ & \mathtt{let} \star = C[-]\, \mathtt{in}\, M \mid \mathtt{let} \star = M\, \mathtt{in}\, C[-]. \end{aligned}$$

The hole can bind term variables, and a well-typed context is defined as for terms. A closed context is a context with no free variables.

We say that $\Delta \vdash M : A$ and $\Delta \vdash N : A$ are operationally equivalent, written $M \simeq_{\mathrm{op}} N$, if for all closed contexts $C[-]$ of type `Bool` where the hole binds Δ, for all b ranging over `tt` and `ff`, $C[M] \to^* b$ if and only if we have $C[N] \to^* b$.

2.3 Axiomatic Equivalence

We also define an equational theory for the language, called *axiomatic equivalence* and denoted with $\simeq_{\mathrm{ax}}$, and mainly used as a technical apparatus. The relation $\simeq_{\mathrm{ax}}$ is defined as the smallest reflexive, symmetric, transitive and fully-congruent relation verifying the rules of Table 2, together with the rules $\lambda x.Mx \simeq_{\mathrm{ax}} M$ and $\langle\, \pi_l(M), \pi_r(M) \,\rangle \simeq_{\mathrm{ax}} M$. A relation $\sim$ is said to be *fully-congruent* on $\mathbf{PCF}_f$ if whenever $M \sim M'$, for all contexts $C[-]$ we also have $C[M] \sim C[N]$. The two additional rules are standard equational rules for a lambda-calculus [19].

$$
\begin{aligned}
[\![\Delta, x : A \vdash x : A]\!]^{\mathrm{set}} &= (d, a) \longmapsto a \\
[\![\Delta \vdash \mathtt{tt} : \mathtt{Bool}]\!]^{\mathrm{set}} &= d \longmapsto \mathtt{tt} \\
[\![\Delta \vdash \mathtt{ff} : \mathtt{Bool}]\!]^{\mathrm{set}} &= d \longmapsto \mathtt{ff} \\
[\![\Delta \vdash \star : \mathbf{1}]\!]^{\mathrm{set}} &= d \longmapsto \star \\
[\![\Delta \vdash \langle\, M, N \,\rangle : A \times B]\!]^{\mathrm{set}} &= d \longmapsto \langle\, [\![M]\!]^{\mathrm{set}}(d), [\![N]\!]^{\mathrm{set}}(d) \,\rangle \\
[\![\Delta \vdash MN : B]\!]^{\mathrm{set}} &= d \longmapsto [\![M]\!]^{\mathrm{set}}(d)([\![N]\!]^{\mathrm{set}}(d)) \\
[\![\Delta \vdash \pi_l(M) : A]\!]^{\mathrm{set}} &= [\![M]\!]^{\mathrm{set}}; \pi_l \\
[\![\Delta \vdash \pi_r(M) : B]\!]^{\mathrm{set}} &= [\![M]\!]^{\mathrm{set}}; \pi_r \\
[\![\Delta \vdash \lambda x.M : A \to B]\!]^{\mathrm{set}} &= d \longmapsto (a \mapsto [\![M]\!]^{\mathrm{set}}(d, a)) \\
[\![\Delta \vdash \mathtt{let}\, \star = M\, \mathtt{in}\, N : A]\!]^{\mathrm{set}} &= [\![N]\!]^{\mathrm{set}}
\end{aligned}
$$

$$
[\![\Delta \vdash \mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P : A]\!]^{\mathrm{set}} = d \longmapsto \begin{cases} [\![N]\!]^{\mathrm{set}}(d) & \text{if} \quad [\![M]\!]^{\mathrm{set}}(d) = \mathtt{tt}, \\ [\![P]\!]^{\mathrm{set}}(d) & \text{if} \quad [\![M]\!]^{\mathrm{set}}(d) = \mathtt{ff}. \end{cases}
$$

Table 3: Denotational semantics for the language $\mathbf{PCF}_f$.

Lemma 4. *If $M : A$ and $M \to N$ then $M \simeq_{\mathrm{ax}} N$.* □

3 Finite Sets as a Concrete Model

Finite sets generate the full sub-category **FinSet** of the category **Set**: objects are finite sets and morphisms are set-functions between finite sets. The category is cartesian closed [31]: the product is the set-product and the internal hom between two sets X and Y is the set of all set-functions from X to Y. Both sets are finite: so is the hom-set.

We can use the category **FinSet** as a model for the language $\mathbf{PCF}_f$. The denotation of types corresponds to the implicit meaning of the types:

$$
[\![\mathbf{1}]\!]^{\mathrm{set}} := \{\, \star \,\}, \qquad [\![\mathtt{Bool}]\!]^{\mathrm{set}} := \{\, \mathtt{tt}, \mathtt{ff} \,\},
$$

the product is the product of sets while the arrow is encoded as the set of morphisms

$$[\![A \times B]\!]^{\mathrm{set}} := [\![A]\!]^{\mathrm{set}} \times [\![B]\!]^{\mathrm{set}}, \qquad [\![A \to B]\!]^{\mathrm{set}} := \mathbf{FinSet}([\![A]\!]^{\mathrm{set}}, [\![B]\!]^{\mathrm{set}}).$$

The set $\{\mathtt{tt}, \mathtt{ff}\}$ is also written $\mathtt{Bool}$. Similarly, the set $\{\star\}$ is written $\mathbf{1}$. The denotation of a typing judgment $x_1 : A_1, \dots x_n : A_n \vdash M : B$ is a morphism

$$[\![A_1]\!]^{\mathrm{set}} \times \cdots \times [\![A_n]\!]^{\mathrm{set}} \to [\![B]\!]^{\mathrm{set}},$$

inductively defined as in Table 3. The variable d is assumed to be an element of $[\![\Delta]\!]^{\mathrm{set}}$, while a and b are elements of $[\![A]\!]^{\mathrm{set}}$ and $[\![B]\!]^{\mathrm{set}}$ respectively.

This denotation is sound with respect to the operational equivalence.

Lemma 5. *If $M \simeq_{\mathrm{ax}} N : A$ then $[\![M]\!]^{\mathrm{set}} = [\![N]\!]^{\mathrm{set}}$.*

Proof. Every axiom defining $(\simeq_{\mathrm{ax}})$ corresponds to an equality coming from either the cartesian closedness or the fact that **FinSet** has coproduct. The congruence rules are true by functoriality of the constructions in the model. □

Theorem 6. *The model is sound with respect to the operational equivalence: Suppose that $\Delta \vdash M, N : A$. If $[\![M]\!]^{\mathrm{set}} = [\![N]\!]^{\mathrm{set}}$ then $M \simeq_{\mathrm{op}} N$.*

Proof. Suppose that $M \not\simeq_{\mathrm{op}} N$ and let Δ be $\{x_i : A_i\}_i$. Then, because of Lemma 3, there exists a context $C[-]$ such that $C[M] \to^* \mathtt{tt}$ and $C[N] \to^* \mathtt{ff}$. It follows that $(\lambda z.C[z\, x_1 \dots x_n])(\lambda x_1 \dots x_n.M) \simeq_{\mathrm{ax}} \mathtt{tt}$ and $(\lambda z.C[z\, x_1 \dots x_n])(\lambda x_1 \dots x_n.N) \simeq_{\mathrm{ax}} \mathtt{ff}$. If the denotations of M and N were equal, so would be the denotations of the terms $(\lambda x_1 \dots x_n.M)$ and $(\lambda x_1 \dots x_n.N)$. Finally, Lemmas 4 and 5 yield a contradiction. □

FinSet and the language $\mathbf{PCF}_f$ are somehow two sides of the same coin. Theorems 7 and 8 formalize this correspondence.

Theorem 7 (Full completeness)**.** *For every morphism $f : [\![A]\!]^{\mathrm{set}} \to [\![B]\!]^{\mathrm{set}}$ there exists a valid judgment $x : A \vdash M : B$ such that $f = [\![M]\!]^{\mathrm{set}}$.*

Proof. We start by defining inductively on A two families of terms $M_a : A$ and $\delta_a : A \to \mathtt{Bool}$ indexed by $a \in [\![A]\!]^{\mathrm{set}}$, such that $[\![M_a]\!]^{\mathrm{set}} = a$ and $[\![\delta_a]\!]^{\mathrm{set}}$ sends a to $\mathtt{tt}$ and all other elements to $\mathtt{ff}$. For the types $\mathbf{1}$ and $\mathtt{Bool}$, the terms $M_\star$, $M_{\mathtt{tt}}$ and $M_{\mathtt{ff}}$ are the corresponding constants. The term $\delta_\star$ is $\lambda x.\star$, $\delta_{\mathtt{tt}}$ is $\lambda x.x$ while $\delta_{\mathtt{ff}}$ is the negation. For the type $A \times B$, one trivially

calls the induction step. The type $A \to B$ is handled by remembering that the set $[\![A]\!]^{\mathrm{set}}$ is finite: if $g \in [\![A \to B]\!]^{\mathrm{set}}$, the term M_g is the lambda-term with argument x containing a list of `if-then-else` testing with δ_a whether x is equal to a, and returning $M_{g(a)}$ if it is. The term δ_g is built similarly. The judgement $x : A \vdash M : B$ asked for in the theorem is obtained by setting M to $(M_f)x$. □

Theorem 8 (Equivalence)**.** *Suppose that $\Delta \vdash M, N : A$. Then we have the equality $[\![M]\!]^{\mathrm{set}} = [\![N]\!]^{\mathrm{set}}$ if and only if $M \simeq_{\mathrm{op}} N$.*

Proof. The left-to-right implication is Theorem 6. We prove the right-to-left implication by contrapositive. Assume that $[\![M]\!]^{\mathrm{set}} \neq [\![N]\!]^{\mathrm{set}}$. Then there exists a function $f : \mathbf{1} \to [\![A]\!]^{\mathrm{set}}$ and a function $g : [\![A]\!]^{\mathrm{set}} \to [\![\mathtt{Bool}]\!]^{\mathrm{set}}$ such that the boolean $f; [\![M]\!]^{\mathrm{set}}; g$ is different from $f; [\![N]\!]^{\mathrm{set}}; g$. By Theorem 7, the functions f and g are representable by two terms N_f and N_g. They generate a context that distinguishes M and N: this proves that $M \not\simeq_{\mathrm{op}} N$. □

Corollary 9. *Since it is fully complete, the semantics* **FinSet** *is also adequate and fully abstract with respect to* $\mathbf{PCF}_f$. □

Example 10. Consider the Church numerals based over $\mathbf{1}$: they are of type $(\mathbf{1} \to \mathbf{1}) \to (\mathbf{1} \to \mathbf{1})$. In **FinSet**, there is only one element since there is only one map from $\mathbf{1}$ to $\mathbf{1}$. As a consequence of Theorem 8, one can conclude that all Church numerals $\lambda f x. f(f(\cdots(fx)\cdots))$ of type $(\mathbf{1} \to \mathbf{1}) \to (\mathbf{1} \to \mathbf{1})$ are operationally equivalent. Note that this is not true in general as soon as the type is inhabited by more elements.

Example 11. How many operationally distinct Church numerals based over `Bool` are there? From Theorem 8, it is enough to count how many distinct denotations of Church numerals lie in $[\![(\mathtt{Bool} \to \mathtt{Bool}) \to (\mathtt{Bool} \to \mathtt{Bool})]\!]^{\mathrm{set}}$. There are exactly 4 distinct maps $\mathtt{Bool} \to \mathtt{Bool}$. Written as pairs (x, y) when $f(\mathtt{tt}) = x$ and $f(\mathtt{ff}) = y$, the maps tt, tf, ft and ff are respectively $(\mathtt{tt}, \mathtt{tt})$, $(\mathtt{tt}, \mathtt{ff})$, $(\mathtt{ff}, \mathtt{tt})$ and $(\mathtt{ff}, \mathtt{ff})$.

Then, if the Church numeral $\bar{n}$ is written as $(\bar{n}(tt), \bar{n}(tf), \bar{n}(ft), \bar{n}(ff))$, we have the following equalities: $\bar{0} = (tf, tf, tf, tf)$, $\bar{1} = (tt, tf, ft, ff)$, $\bar{2} = (tt, tf, tf, ff)$, $\bar{3} = (tt, tf, ft, ff)$, and one can show that for all $n \geq 1$, $[\![\bar{n}]\!]^{\mathrm{set}} = [\![n \dot{+} 2]\!]^{\mathrm{set}}$. There are therefore only 3 operationally distinct Church numerals based on the type `Bool`: the number $\bar{0}$, then all even non-null numbers, and finally all odd numbers.

4 Finite Vector Spaces

We now turn to the second finitary model that we want to use for the language $\mathbf{PCF}_f$: finite vector spaces. We first start by reminding the reader about this algebraic structure.

4.1 Background Definitions

A *field* [21] K is a commutative ring such that the unit 0 of the addition is distinct from the unit 1 of the multiplication and such all non-zero elements of K admit an inverse with respect to the multiplication. A *finite field* is a field of finite size. The *characteristic* q of a field K is the minimum (non-zero) number such that $1 + \cdots + 1 = 0$ (q instances of 1). If there is none, we say that the characteristic is 0. For example, the field of real numbers has characteristic 0, while the field $\mathbb{F}_2$ consisting of 0 and 1 has characteristic 2. The *order* of a finite field is the order of its multiplicative group.

A *vector space* [20] V over a field K is an algebraic structure consisting of a set $|V|$, a binary addition $+$ and a scalar multiplication $(\cdot) : K \times V \to V$, satisfying the equations obtained by treating the rewrite rules of Table 6 as equivalences. The *dimension* of a vector space is the size of the largest set of independent vectors. A particular vector space is the vector space *freely generated from a space* X, denoted with $\langle X \rangle$: it consists of all the formal finite linear combinations $\sum_i \alpha_i \cdot x_i$, where x_i belongs to X and α_i belongs to K. To define a linear map f on $\langle X \rangle$, it is enough to give its behavior on each of the vector $x \in X$: the image of $\sum_i \alpha_i \cdot x_i$ is then by linearity imposed to be $\sum_i \alpha_i \cdot f(x_i)$.

In this paper, the vector spaces we shall concentrate on are *finite vector spaces*, that is, vector spaces of finite dimensions over a finite field. For example, the 2-dimensional space $\mathbb{F}_2 \times \mathbb{F}_2$ consists of the four vectors $\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}$ and is a finite vector space. It is also the vector space freely generated from the 2-elements set $\{\mathtt{tt}, \mathtt{ff}\}$: each vectors respectively corresponds to $0, \mathtt{tt}, \mathtt{ff}$, and $\mathtt{tt} + \mathtt{ff}$.

Once a given finite field K has been fixed, the category **FinVec** has for objects finite vector spaces over K and for morphisms linear maps between these spaces. The category is symmetric monoidal closed: the tensor product is the algebraic tensor product, the unit of the tensor is $I = K = \langle \star \rangle$ and the internal hom between two spaces U and V is the vector space of all linear functions $U \multimap V$ between U and V. The addition and the scalar multiplication over functions are pointwise.

4.2 A Linear-non-linear Model

It is well-known [22] that the category of finite sets and functions and the category of finite vector spaces and linear maps form an adjunction

$$\mathbf{FinSet} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{FinVec}. \tag{1}$$

The functor F sends the set X to the vector space $\langle X \rangle$ freely generated from X and the set-map $f : X \to Y$ to the linear map sending a basis element $x \in X$ to the base element $f(x)$. The functor G sends a vector space U to the same space seen as a set, and consider any linear function as a set-map from the corresponding sets.

This adjunction makes **FinVec** into a model of linear logic [25]. Indeed, the adjunction is symmetric monoidal with the following two natural transformations:

$$\begin{array}{rcl} m_{X,Y} : \langle X \times Y \rangle & \to & \langle X \rangle \otimes \langle Y \rangle \\ (x, y) & \mapsto & x \otimes y, \end{array} \qquad \begin{array}{rcl} m_1 : \langle 1 \rangle & \to & I \\ \star & \mapsto & 1. \end{array}$$

This makes a *linear-non-linear category* [5], equivalent to a linear category, and is a model of intuitionistic linear logic [6].

4.3 Model of Linear Logic

The adjunction in Eq. (1) is monoidal and generates a linear comonad on **FinVec**. If A is a finite vector space, we define the finite vector space $!A$ as the vector space freely generated from the set $\{b_v\}_{v \in A}$: it consists of the space $\langle b_v \mid v \in A \rangle$. If $f : A \to B$ is a linear map, the map $!f : !A \to !B$ is defined as $b_v \mapsto b_{f(v)}$. The comultiplication and the counit of the comonad are respectively $\delta_A : !A \to !!A$ and $\epsilon_A : !A \to A$ where $\delta_A(b_v) = b_{b_v}$ and $\epsilon_A(b_v) = v$. Every element $!A$ is a commutative comonoid when equipped with the natural transformations $\Delta_A : !A \to !A \otimes !A$ and $\diamond_A : !A \to I$ where $\Delta_A(b_v) = b_v \otimes b_v$ and $\diamond(b_v) = 1$. This makes the category **FinVec** into a linear category.

Moreover, the category **FinVec** has biproducts: $A \oplus B$ is the regular product of vector spaces. Since all spaces are finite dimensional, the product and the coproduct of vector spaces coincides. In particular, we have an isomorphism between $!A \otimes !B$ and $!(A \times B)$, as it is customary in models of linear logics with additives.

Remark 12. The construction exposed in a side example by Hyland and Schalk [17] considers finite vector spaces over a field of characteristic 2. There, the modality is built using the exterior product algebra, which turns out in that case to be identical to the functor we use in the present paper. Note though, that their construction does not work with fields of other characteristics.

¿From a model of linear logics, there are two ways of building a cartesian closed category. In the rest of this section, we develop these two categories.

4.3.1 The Kleisli Category

The first cartesian closed category one can build from **FinVec** is the Kleisli category $\mathbf{FinVec}_!$. In $\mathbf{FinVec}_!$, the objects are the objects of **FinVec**, and the morphisms $\mathbf{FinVec}_!(A, B)$ are the morphisms $\mathbf{FinVec}(!A, B)$. The identity $!A \to A$ is the counit and the composition of $f : !A \to B$ and $!B \to C$ is $f; g := !A \xrightarrow{\delta_A} !!A \xrightarrow{!f} !B \xrightarrow{g} C$. It is a cartesian closed category: the product of A and B is $A \times B$, the usual product of vector spaces, and the terminal object is the vector space $\langle 0 \rangle$.

In $\mathbf{FinVec}_!$ the product of vector spaces $A \times B$ also features left and right injections from A and B defined as

$$\begin{array}{lll} !A & \to A \times B & \qquad !B \quad \to A \times B \\ b_u & \mapsto \langle u, 0 \rangle, & \qquad b_v \quad \mapsto \langle 0, v \rangle. \end{array}$$

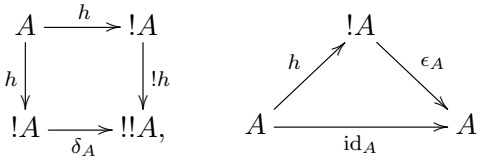
As in finite vector spaces the product is a *bi*product, one might hope to endow $A \times B$ with a coproduct structure too. However, $A \times B$ is only a *weak* coproduct: given two maps $f : !A \to C$ and $g : !B \to C$, there is no canonical function $[f, g] : !(A \times B) \to C$. Indeed, the comonad equations enforce the values of $[f, g](b_{\langle u,0 \rangle})$ and $[f, g](b_{\langle 0,v \rangle})$, but not of $[f, g](b_{\langle u,v \rangle})$ when both u and v are non-zero vectors.

As we shall see in Section 4.4, this weak coproduct is nonetheless enough to define a satisfactory denotation for `Bool` as $I \oplus I$.

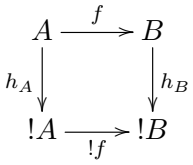
4.3.2 The Eilenberg-Moore Category

Given the comonad !, the Eilenberg-Moore category [5] $\mathbf{FinVec}^!$ consists in !-coalgebras, that is, pairs (A, h) where A is an object of **FinVec** (a vector

space) and h is a linear map $A \to !A$ such that the diagrams



commutes. A morphism $(A, h_A) \to (B, h_B)$ is a morphism of the base category $A \to B$ such that



commutes. The identity on (A, h_A) is the identity on A, and the composition of two arrows in the Eilenberg-Moore category is the composition in the base category.

The equations arising from the comonadic structures ensure that for every element A of **FinVec** there is a canonical coalgebra $(!A, \delta_A)$: it is called the *free* coalgebra on A.

Lemma 13 (from Benton [5])**.** *For all coalgebras (A, h_A) and all objects B, there is an isomorphism between the homset* $\mathbf{FinVec}^!((A, h_A), (!B, \delta_B))$ *and the homset* $\mathbf{FinVec}(A, B)$*.* □

The category $\mathbf{FinVec}^!$ is cartesian. The product of (A, h_A) and (B, h_B) is constructed as $(A \otimes B, h_{A\otimes B})$ where $h_{A\otimes B}$ is defined as

$$A \otimes B \xrightarrow{h_A \otimes h_B} !A \otimes !B \xrightarrow{q_{A,B}} !(A \otimes B),$$

where $q_{A,B}$ is the strength of the comonad: a map sending $b_u \otimes b_v$ to $b_{u\otimes v}$. The projections are

$$\pi_1 : A \otimes B \xrightarrow{h_A \otimes h_B} !A \otimes !B \xrightarrow{\epsilon_A \otimes \Diamond_B} A \otimes I \simeq A,$$
$$\pi_2 : A \otimes B \xrightarrow{h_A \otimes h_B} !A \otimes !B \xrightarrow{\Diamond_A \otimes \epsilon_B} I \otimes B \simeq B,$$

and the terminal object is (I, q_I), where $q_I = \Diamond_I$ sends $\star$ to $b_\star$.

4.3.3 The Category of Free Coalgebras

Every free coalgebra is *exponentiable* [5]: that is, for all coalgebras (A, h_A) and all objects B, the free coalgebra $(A, h_A) \Rightarrow (!B, \delta_B)$ defined as $(!(A \multimap B), \delta_{A \multimap B})$ forms an internal hom as shown by the following sequence of isomorphisms:

$$\begin{aligned}
&\mathbf{FinVec}^!((C, h_C), (!(A \multimap B), \delta_{A\multimap B})) \\
&\quad \simeq \mathbf{FinVec}(C, A \multimap B) && \text{by Lemma 13,} \\
&\quad \simeq \mathbf{FinVec}(C \otimes A, B) && \text{since } A \multimap B \text{ is an internal hom,} \\
&\quad \simeq \mathbf{FinVec}^!((C \otimes A, h_{C\otimes A}), (!B, \delta_B)).
\end{aligned}$$

Let **FreeCoalg** be the full subcategory of $\mathbf{FinVec}^!$ having for objects the free coalgebras. It is cartesian closed, and the cartesian structure is inherited from the category $\mathbf{FinVec}^!$:

- the terminal object (I, q_I) can written as a free coalgebra, since $I = !\langle 0\rangle$, and since $q_I = \delta_{\langle 0\rangle}$.
- Since $!A \otimes !B = !(A \times B)$, the product $(!A, \delta_A) \times (!B, \delta_B)$ can be written as $(!(A \times B), \delta_{A\times B})$.
- An internal hom is by definition a free coalgebra.

What is a morphism $f : !A \to !B$ of free coalgebras in **FinVec**? It is a linear map sending each b_v to $f(b_v) = \sum_i \alpha_i \cdot b_{u_i}$ and such that

$$\sum_i \alpha_i \cdot b_{b_{u_i}} = \delta_B(f(b_v)) = (!f)(\delta_A(b_v)) = (!f)(b_{b_v}) = b_{f(b_v)}.$$

Thus there is only one element in the sum, and we can state the following property:

Property 14. *A free-coalgebra morphism $f : !A \to !B$ is equivalent to a set-function $g : A \to B$ such that, to each $b_v \in !A$ we have $f(b_v) = b_{g(v)}$.*

This property clarifies the bijection between $\mathbf{FinVec}^!((!A, \delta_A), (!B, \delta_B))$ and $\mathbf{FinVec}(!A, B)$ presented in Lemma 13, and shows how the category **FreeCoalg** is equivalent to the Kleisli category $\mathbf{FinVec}_!$.

In particular, since we can equip $\mathbf{FinVec}_!(A, B)$ with the structure of a vector space by using pointwise addition and pointwise scalar multiplication,

the homset $\mathbf{FreeCoalg}((!A, \delta_A), (!B, \delta_B))$ can be made into a vector space in the following way. Given f and g free-coalgebra morphisms $!A \to !B$ such that $f(b_v) = b_u$ and $g(b_v) = b_w$, define

$$(f \mathbin{\tilde{+}} g)(b_v) = b_{u+w}, \qquad (\alpha \mathbin{\tilde{\cdot}} f)(b_v) = b_{\alpha \cdot u}.$$

Similarly, the category **FreeCoalg** enjoys weak coproducts. The weak coproduct of $(!A, \delta_A)$ and $(!B, \delta_B)$ is the free co-algebra $(!(!A \oplus !B), \delta_{!A \oplus !B})$, where $\oplus$ is the usual product of vector space. The left and right injections are respectively

$$b_u \longmapsto b_{\langle b_u, 0\rangle}, \qquad b_v \longmapsto b_{\langle 0, b_v\rangle}.$$

4.3.4 Linear Combinations of Free Coalgebra Morphisms

The operations $(\tilde{+})$ and $(\tilde{\cdot})$ are not the naïve operations on functions we could have defined: as for the homset $\mathbf{FinVec}_!$, one could have asked the addition and the scalar multiplication to be pointwise. The problem with this definition is that addition and scalar multiplication of free co-algebra morphisms does not in general gives free-coalgebra morphisms: they would break Property 14.

However, one can elude the difficulty be defining a new category on top of **FreeCoalg** whose morphisms are indeed closed under pointwise addition and pointwise scalar multiplication. Thus, let us define $\mathbf{FreeCoalg}^+$ whose objects are the objects of **FreeCoalg** and whose homsets $\mathbf{FreeCoalg}^+(!A, !B)$ are the free vector spaces $\langle \mathbf{FreeCoalg}(!A, !B)\rangle$ generated from the set of free-coalgebra morphisms $!A \to !B$. The identity on $(!A, \delta_A)$ is the identity on $!A$ in **FinVec** and the composition of two morphisms $\sum_i \alpha_i \cdot f_i : (!A, \delta_A) \to (!B, \delta_B)$ and $\sum_j \beta_j \cdot g_j : (!B, \delta_B) \to (C, \delta_C)$ is the morphism obtained by distributivity:

$$\sum_{i,j} \alpha_i \beta_j \cdot (f_i; g_j).$$

The cartesian closed structure of **FreeCoalg** is turned into a symmetric monoidal closed structure in $\mathbf{FreeCoalg}^+$.

However, as we discuss in Section 5.3.3 the weak coproducts of the category **FreeCoalg** are not enough to make a satisfactory interpretation of the type `Bool` in $\mathbf{FreeCoalg}^+$.

4.3.5 Computational Interpretation

The main computational difference between the category $\mathbf{FinVec}_!$ and the category $\mathbf{FreeCoalg}^+$ is the way they deal with the "algebraic side-effect".

Let B be the two-dimensional vector space $\{\alpha\cdot\mathtt{ff}+\beta\cdot\mathtt{tt}\}$, and remember $\Delta_B : !B \to !(B \times B)$ sending b_v to the element $b_{\langle v,v\rangle}$. The map Δ_B is a free-coalgebra morphism, and as such belongs both in $\mathbf{FinVec}_!$ (seen as $\mathbf{FreeCoalg}$) and in $\mathbf{FreeCoalg}^+$. Let $\mathtt{xor} : B \times B \to B$ be the exclusive-or operation. Let $f, g : I \to !B$ be the two free-coalgebra morphisms $f(\star) = b_{\mathtt{ff}}$ and $g(\star) = b_{\mathtt{tt}}$. Finally, assume that the field is $\mathbb{F}_5$. Then in $\mathbf{FreeCoalg}$ (thus in $\mathbf{FinVec}_!$), the composition $(f \tilde{+} g); \Delta_B; !\mathtt{xor}$ computes

$$\star \xrightarrow{f\tilde{+}g} b_{\mathtt{ff}+\mathtt{tt}} \xrightarrow{\Delta_B} b_{\langle \mathtt{ff}+\mathtt{tt},\mathtt{ff}+\mathtt{tt}\rangle} \xrightarrow{!\mathtt{xor}} b_{(\mathtt{xor}(\mathtt{ff}+\mathtt{tt}))(\mathtt{ff}+\mathtt{tt})} = b_{2\cdot\mathtt{tt}+2\cdot\mathtt{ff}}$$

since $(\mathtt{xor}(\mathtt{ff}+\mathtt{tt}))(\mathtt{ff}+\mathtt{tt})$ is really simply $\mathtt{xor}\,\mathtt{ff}\,\mathtt{ff}+\mathtt{xor}\,\mathtt{ff}\,\mathtt{tt}+\mathtt{xor}\,\mathtt{tt}\,\mathtt{ff}+\mathtt{xor}\,\mathtt{tt}\,\mathtt{tt}$. On the contrary, in $\mathbf{FreeCoalg}^+$ the composition $(f+g); \Delta_B; !\mathtt{xor}$ computes

$$\star \xrightarrow{f\tilde{+}g} b_{\mathtt{ff}+\mathtt{tt}} \xrightarrow{\Delta_B} b_{\langle \mathtt{ff},\mathtt{ff}\rangle} + b_{\langle \mathtt{tt},\mathtt{tt}\rangle} \xrightarrow{!\mathtt{xor}} b_{\mathtt{xor}\,\mathtt{ff}\,\mathtt{ff}} + b_{\mathtt{xor}\,\mathtt{tt}\,\mathtt{tt}} = 2 \cdot b_{\mathtt{ff}}.$$

Remark 15. One can therefore say that $\mathbf{FinVec}_!$ is call-by-name in spirit: the "computation" $\mathtt{ff}+\mathtt{tt}$ gets duplicated, whereas the category $\mathbf{FreeCoalg}^+$ is more call-by-value: the computation $\mathtt{ff}+\mathtt{tt}$ is evaluated before being fed as argument. We will analyse this intuition in more detail in Section 5.

4.3.6 Relation Between the Categories

Cartesian closed structures. ¿From **FinVec** we built $\mathbf{FinVec}_!$ which is cartesian closed. It does relate with the cartesian closed **FinSet**: the forgetful functor

$$\mathbf{FinVec}_! \xrightarrow{U} \mathbf{FinSet}$$

sends an object V to the set of vectors of V and sending the linear map $f : !V \to W$ to the map $v \mapsto f(b_v)$. It is a full embedding: Given two vector spaces V and W, there is an exact correspondance between the set of maps $\mathbf{FinVec}_!(V, W)$ and the set of set-functions $\mathbf{FinSet}(U(V), U(W))$.

The functor U preserves the cartesian closed structure: the terminal object $\langle 0\rangle$ of $\mathbf{FinVec}_!$ is sent to the set containing only 0, that is, the singleton-set $\mathbf{1}$. The product space $V \times W$ is sent to the set of vectors $\{\langle u, v\rangle \mid u \in U, v \in V\}$, which is exactly the set-product of V and W. Finally, the function space $!V \to W$ is in exact correspondence with the set of set-functions $V \to W$.

Relating $\mathbf{FinVec}_!$ and $\mathbf{FreeCoalg}^+$. Since the category **FreeCoalg** and $\mathbf{FinVec}_!$ are equivalent, there is an embedding

$$\mathbf{FinVec}_! \xrightarrow{E} \mathbf{FreeCoalg}^+$$

sending an object V onto $(!V, \delta_V)$ and sending a morphism in the Kleili category on its corresponding free-coalgebra map, as specified in Property 14. This embedding maps the cartesian structure of $\mathbf{FinVec}_!$ onto the monoidal structure of $\mathbf{FreeCoalg}^+$.

4.4 Finite Vector Spaces as a Model

Since $\mathbf{FinVec}_!$ is a cartesian closed category, one can model terms of $\mathbf{PCF}_f$ as linear maps. Types are interpreted as follows. The unit type is $[\![\mathbf{1}]\!]^{\mathrm{vec}} := \{\, \alpha \cdot \star \mid \alpha \in K \,\}$. The boolean type is $[\![\mathtt{Bool}]\!]^{\mathrm{vec}} := \{\, \alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff} \mid \alpha, \beta \in K \,\}$. The product is the usual product space: $[\![A \times B]\!]^{\mathrm{vec}} := [\![A]\!]^{\mathrm{vec}} \times [\![B]\!]^{\mathrm{vec}}$, whereas the arrow type is $[\![A \to B]\!]^{\mathrm{vec}} := \mathbf{FinVec}(![\![A]\!]^{\mathrm{vec}}, [\![B]\!]^{\mathrm{vec}})$. A typing judgment $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ is represented by a morphism of **FinVec** of type

$$![\![A_1]\!]^{\mathrm{vec}} \otimes \cdots \otimes ![\![A_n]\!]^{\mathrm{vec}} \longrightarrow [\![B]\!]^{\mathrm{vec}}, \tag{2}$$

inductively defined as in Table 4. The variable d stands for a base element $b_{u_1} \otimes \ldots \otimes b_{u_n}$ of $[\![\Delta]\!]^{\mathrm{vec}}$, and b_a is a base element of $[\![A]\!]^{\mathrm{vec}}$. The functions π_l and π_r are the left and right projections of the product.

Note that because of the equivalence between $!(A \times B)$ and $!A \otimes !B$, the map in Eq. (2) is a morphism of $\mathbf{FinVec}_!$, as desired.

Example 16. In **FinSet**, there was only one Church numeral based on type $\mathbf{1}$. In $\mathbf{FinVec}_!$, there are more elements in the corresponding space $!(!\mathbf{1} \multimap \mathbf{1}) \multimap (!\mathbf{1} \multimap \mathbf{1})$ and we get more distinct Church numerals.

Assume that the finite field under consideration is the 2-elements field $\mathbb{F}_2 = \{0, 1\}$. Then $[\![\mathbf{1}]\!]^{\mathrm{vec}} = \mathbf{1} = \{0 \cdot \star, 1 \cdot \star\} = \{0, \star\}$. The space $!\mathbf{1}$ is freely generated from the vectors of $\mathbf{1}$: it therefore consists of just the four vectors $\{0, b_0, b_\star, b_0 + b_\star\}$. The space of morphisms $[\![\mathbf{1} \to \mathbf{1}]\!]^{\mathrm{vec}}$ is the space $!\mathbf{1} \multimap \mathbf{1}$. It is generated by two functions: f_0 sending b_0 to $\star$ and $b_\star$ to 0, and $f_\star$ sending b_v to v. The space therefore also contains 4 vectors: 0, f_0, $f_\star$ and $f_0 + f_\star$. Finally, the vector space $!(!\mathbf{1} \multimap \mathbf{1})$ is freely generated from the 4 base elements b_0, b_{f_0}, $b_{f_\star}$ and $b_{f_0+f_\star}$, therefore containing 16 vectors. Morphisms

$$
\begin{array}{rcl}
[\![\Delta, x : A \vdash x : A]\!]^{\mathrm{vec}} & = & d \otimes b_a \longmapsto a \\
[\![\Delta \vdash \mathtt{tt} : \mathtt{Bool}]\!]^{\mathrm{vec}} & = & d \longmapsto \mathtt{tt} \\
[\![\Delta \vdash \mathtt{ff} : \mathtt{Bool}]\!]^{\mathrm{vec}} & = & d \longmapsto \mathtt{ff} \\
[\![\Delta \vdash \star : \mathbf{1}]\!]^{\mathrm{vec}} & = & d \longmapsto \star \\
[\![\Delta \vdash \langle\, M, N \,\rangle : A \times B]\!]^{\mathrm{vec}} & = & d \longmapsto [\![M]\!]^{\mathrm{vec}}(d) \otimes [\![N]\!]^{\mathrm{vec}}(d) \\
[\![\Delta \vdash MN : B]\!]^{\mathrm{vec}} & = & d \longmapsto [\![M]\!]^{\mathrm{vec}}(d)([\![N]\!]^{\mathrm{vec}}(d)) \\
[\![\Delta \vdash \pi_l(M) : A]\!]^{\mathrm{vec}} & = & [\![M]\!]^{\mathrm{vec}}; \pi_l \\
[\![\Delta \vdash \pi_r(M) : B]\!]^{\mathrm{vec}} & = & [\![M]\!]^{\mathrm{vec}}; \pi_r \\
[\![\Delta \vdash \lambda x.M : A \to B]\!]^{\mathrm{vec}} & = & d \longmapsto (b_a \mapsto [\![M]\!]^{\mathrm{vec}}(d \otimes b_a)) \\
[\![\Delta \vdash \mathtt{let}\, \star = M\, \mathtt{in}\, N : A]\!]^{\mathrm{vec}} & = & d \longmapsto \alpha \cdot [\![N]\!]^{\mathrm{vec}}(d) \\
 & & \text{where } [\![M]\!]^{\mathrm{vec}}(d) = \alpha \cdot \star. \\
[\![\Delta \vdash \mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P : A]\!]^{\mathrm{vec}} & = & d \longmapsto \alpha \cdot [\![N]\!]^{\mathrm{vec}}(d) + \beta \cdot [\![P]\!]^{\mathrm{vec}}(d) \\
 & & \text{where } [\![M]\!]^{\mathrm{vec}}(d) = \alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff}.
\end{array}
$$

Table 4: Modeling the language $\mathbf{PCF}_f$ in **FinVec**.

$!(!\mathbf{1} \multimap \mathbf{1}) \multimap (!\mathbf{1} \multimap \mathbf{1})$ can be represented by 2×4 matrices with coefficients in $\mathbb{F}_2$. The basis elements b_v are ordered as follows:

$$\begin{matrix} \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} & \begin{matrix} f_0 \\ f_\star \end{matrix} \\ \begin{matrix} b_0 & b_{f_0} & b_{f_\star} & b_{f_0+f_\star} \end{matrix} & \end{matrix}$$

as are the basis elements f_w, as shown on the right. The Church numeral $\bar{0}$ sends all of its arguments to the identity function, that is, $f_\star$. The Church numeral $\bar{1}$ is the identity. So their respective matrices are $\left(\begin{smallmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{smallmatrix}\right)$. The next two Church numerals are $\bar{2} = \left(\begin{smallmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{smallmatrix}\right)$ and $\bar{3} = \left(\begin{smallmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{smallmatrix}\right)$, which is also $\bar{1}$. So $\mathbf{FinVec}_!$ with the field of characteristic 2 distinguishes null, even and odds numerals over the type $!\mathbf{1}$.

Note that this characterization is similar to the **FinSet** Example 11, except that there, the type over which the Church numerals were built was `Bool`. Over $\mathbf{1}$, Example 10 stated that all Church numerals collapse.

Example 17. The fact that $\mathbf{FinVec}_!$ with the field of characteristic 2 can be put in parallel with **FinSet** when considering Church numerals is an artifact of the fact that the field has only two elements. If instead one chooses another field $K = \mathbb{F}_p = \{0, 1, \ldots, p-1\}$ of characteristic p, with p prime, then this is in general not true anymore. In this case, $[\![\mathbf{1}]\!]^{\mathrm{vec}} = \{0, \star, 2 \cdot \star, \ldots, (p-1) \cdot \star\}$, and $!\mathbf{1} \multimap \mathbf{1}$ has dimension p with basis elements f_i sending $b_{i \cdot \star} \mapsto \star$ and $b_{j \cdot \star} \mapsto 0$ when $i \neq j$. It therefore consists of p^p vectors. Let us represent a function $f : !\mathbf{1} \multimap \mathbf{1}$ with $\mathtt{x_0} \ldots \mathtt{x_{p-1}}$ where $f(b_{i \cdot \star}) = \mathtt{x}_i \cdot \star$. A morphism $!(!\mathbf{1} \multimap \mathbf{1}) \multimap (!\mathbf{1} \multimap \mathbf{1})$ can be represented with a $p^p \times p$ matrix. The basis elements $b_{\mathtt{x_0} \ldots \mathtt{x_{p-1}}}$ of $!(!\mathbf{1} \multimap \mathbf{1})$ are ordered lexicographically: $b_{0\ldots00}, b_{0\ldots01}, b_{0\ldots02}, \ldots, b_{0\ldots0(p-1)}, \ldots, b_{(p-1)\ldots(p-1)}$, as are the basis elements $f_0, f_1, \ldots, f_{p-1}$.

The Church numeral $\bar{0}$ is again the constant function returning the identity, that is, $\sum_i i \cdot f_i$. The numeral $\bar{1}$ sends $\mathtt{x_0} \cdots \mathtt{x_{p-1}}$ onto the function sending $b_{i \cdot \star}$ onto $\mathtt{x}_i \cdot \star$. The numeral $\bar{2}$ sends $\mathtt{x_0} \cdots \mathtt{x_{p-1}}$ onto the function sending $b_{i \cdot \star}$ onto $\mathtt{x}_{\mathtt{x}_i} \cdot \star$. The numeral $\bar{3}$ sends $\mathtt{x_0} \cdots \mathtt{x_{p-1}}$ onto the function sending $b_{i \cdot \star}$ onto $\mathtt{x}_{\mathtt{x}_{\mathtt{x}_i}} \cdot \star$. And so on.

In particular, each combination $\mathtt{x_0} \cdots \mathtt{x}_{p-1}$ can be considered as a function $\mathtt{x} : \{0, \ldots p-1\} \to \{0, \ldots p-1\}$. The sequence $(\mathtt{x}^0, \mathtt{x}^1, \mathtt{x}^2, \ldots)$ eventually loops. The order of the loop is $lcm(p)$, the least common multiple of all integers $1, \ldots, p$, and for all $n \geq p-1$ we have $\mathtt{x}^n = \mathtt{x}^{n+lcm(p)}$: there are $lcm(p) + p - 1$ distinct Church numerals in the model $\mathbf{FinVec}_!$ with a field of characteristic p prime.

$$\overline{0} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2 \end{pmatrix}$$

$$\overline{1} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2 \\ 0&0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2&0&0&0&1&1&2&2&2&2&2&2&2&2&2 \\ 0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2 \end{pmatrix}$$

$$\overline{2+6n} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&2&2&2&0&1&2&0&1&2&0&1&2&0&1&2&0 \\ 0&0&0&1&1&1&0&1&2&1&1&1&1&1&1&0&1&2&2&2&2&2&1&1&1&0&1&2&2&2&2&2 \\ 0&0&2&0&1&2&0&2&2&1&0&2&1&1&2&1&2&2&2&0&2&2&1&2&2&2&2&2&2&2&2&2 \end{pmatrix}$$

$$\overline{3+6n} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&0&1&2&2&0&2&2&1&2&2&2&2&2&2&2&2 \\ 0&0&0&1&1&1&0&2&2&0&0&0&1&1&1&1&2&2&0&1&2&1&1&1&2&2&2&2&2&2&2&2 \\ 0&0&2&0&1&2&0&1&2&0&1&2&1&1&2&2&1&2&0&2&2&0&1&2&0&1&2&2&2&2&2&2 \end{pmatrix}$$

$$\overline{4+6n} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&2&2&0&2&2&0&1&2&0&1&2&2&2&2&2 \\ 0&0&0&1&1&1&0&1&2&1&1&1&1&1&1&2&1&2&2&0&2&1&1&1&0&1&2&2&2&2&2&2 \\ 0&0&2&0&1&2&0&2&2&1&0&2&1&1&2&0&2&2&2&1&2&2&1&2&2&2&2&2&2&2&2&2 \end{pmatrix}$$

$$\overline{5+6n} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&2&1&2&2&1&2&2&1&2&2&2&2&2&2&2&2&2 \\ 0&0&0&1&1&1&0&2&2&0&0&0&1&1&1&0&2&2&0&2&2&1&1&1&2&2&2&2&2&2&2&2 \\ 0&0&2&0&1&2&0&1&2&0&1&2&1&1&2&1&1&2&0&0&2&0&1&2&0&1&2&2&2&2&2&2 \end{pmatrix}$$

$$\overline{6+6n} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&2&2&2&0&0&2&0&1&2&0&1&2&0&1&2 \\ 0&0&0&1&1&1&0&1&2&1&1&1&1&1&1&1&1&2&2&2&1&2&1&1&1&0&1&2&2&2&2&2 \\ 0&0&2&0&1&2&0&2&2&1&0&2&1&1&2&2&2&2&2&2&2&2&2&2&1&2&2&2&2&2&2&2 \end{pmatrix}$$

$$\overline{7+6n} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&2&2&2&2&2&2&1&2&2&2&2&2&2&2&2 \\ 0&0&0&1&1&1&0&2&2&0&0&0&1&1&1&2&2&2&2&0&0&2&1&1&1&2&2&2&2&2&2&2 \\ 0&0&2&0&1&2&0&1&2&0&1&2&1&1&2&0&1&2&0&1&2&0&1&2&0&1&2&2&2&2&2&2 \end{pmatrix}$$

Table 5: The 8 Church numerals over type $\mathbf{1}$ in $\mathbf{FinVec}_!$ with $K = \mathbb{F}_3$.

For $p = 2$ we recover the 3 distinct Church numerals. But for $p = 3$, we deduce that there are 8 distinct Church numerals (the 8 corresponding matrices are reproduced in Table 5). As there is almost a factorial function, the number of distinct Church numerals grows fast as p grows: With $\mathbb{F}_5$, there are 64 distinct numerals, and with $\mathbb{F}_7$ there are 426 distinct numerals.

Example 18. Let us briefly reprise Example 11 in the context of $\mathbf{FinVec}_!$. Even with a field of size 2, the space $[\![(\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})]\!]^{\mathrm{vec}}$ is relatively large: `Bool` has dimension 2 and consists of 4 vectors, `!Bool` then has dimension 4 and consists of 16 vectors. The dimension of the homset $\texttt{!Bool} \multimap \texttt{Bool}$ is 8, and it contains $2^8 = 256$ vectors. Using the representation of the two previous examples, a Church numeral is a matrix of size 256×8.

Let us represent a function $\texttt{!Bool} \multimap \texttt{Bool}$ as a tuple $(\mathtt{x}^k_{ij})_{i,j,k}$ lexicographically ordered $\mathtt{x}^0_{00}, \mathtt{x}^1_{00}, \mathtt{x}^0_{01}, \mathtt{x}^1_{01}, \mathtt{x}^0_{10}, \mathtt{x}^1_{10}, \mathtt{x}^0_{11}, \mathtt{x}^1_{11}$, representing the map sending $b_{i\cdot\mathtt{tt}+j\cdot\mathtt{ff}}$ to $\mathtt{x}^0_{ij}\cdot\mathtt{tt}+\mathtt{x}^1_{ij}\cdot\mathtt{ff}$. These form the basis elements of the range of the matrix. The domain of the matrix consists of all the 256 combinations of 0/1 values that these can take. Ordered lexicographically, they form the basis of the domain of the matrix.

As before, the numeral $\bar{0}$ is constant while $\bar{1}$ is the identity. The numeral

$\bar{2}$ sends each of the 8-tuples $(\mathtt{x}_{ij}^k)_{i,j,k}$ to the 8-tuple

$$(x^0_{x^0_{a,b},\mathtt{x}^1_{a,b}}, \mathtt{x}^1_{x^0_{a,b},\mathtt{x}^1_{a,b}})_{a,b\in\{0,1\}},$$

and so forth. So for example, the negation sending $b_{a\cdot\mathtt{tt}+b\cdot\mathtt{ff}}$ to $a\cdot\mathtt{ff}+b\cdot\mathtt{tt}$ is the 8-tuple $(0,0,1,0,0,1,1,1)$ and is sent by $\bar{2}$ to the tuple $(0,0,0,1,1,0,1,1)$ which is indeed the identity.

If one performs the calculation, one finds out that in $\mathbf{FinVec}_!$, over the type `Bool`, there are exactly 15 distinct Church numerals. The numerals $\bar{0}$, $\bar{1}$ and $\bar{2}$ are uniquely determined, and then the semantics distinguishes the equivalence classes $\{i+12n \mid n\in\mathbb{N}\}$, for $i=3,4,\dots 14$.

4.5 Properties of the FinVec Model

As shown in the next results, this semantics is both sound and adequate with respect to the operational equivalence. Usually adequacy uses non-terminating terms. Because the language is strongly normalizing, we adapt the notion. However, because there are usually more maps between $[\![A]\!]^{\mathrm{vec}}$ and $[\![B]\!]^{\mathrm{vec}}$ than between $[\![A]\!]^{\mathrm{set}}$ and $[\![B]\!]^{\mathrm{set}}$ (as shown in Examples 16, 17 and 18), the model fails to be fully abstract.

Lemma 19. *If $M \simeq_{\mathrm{ax}} N : A$ then $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$.*

Proof. The proof follows the same pattern as the proof of Lemma 5: every axiom defining $(\simeq_{\mathrm{ax}})$ corresponds to an equality coming from either the cartesian closedness or the fact that $\mathbf{FinVec}_!$ has coproduct. The congruence rules are then proven correct by functoriality of the constructions in the model. □

Theorem 20. *If $\Delta \vdash M, N : A$ and $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$ then $M \simeq_{\mathrm{op}} N$.*

Proof. The proof is similar to the proof of Theorem 6 and proceeds by contrapositive, using Lemmas 3, 3, 4 and 19. □

Theorem 21 (Adequacy)**.** *Given two closed terms M and N of type* `Bool`*, we have the equality $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$ if and only if $M \simeq_{\mathrm{op}} N$.*

Proof. The left-to-right direction is Theorem 20. For the right-to-left direction, since the terms M and N are closed of type `Bool`, one can choose the context $C[-]$ to be $[-]$, and we have $M \to^* b$ if and only if $N \to^* b$. From Lemma 3, there exists such a boolean b: we deduce from Lemma 4 that $M \simeq_{\mathrm{ax}} N$. We conclude with Lemma 19. □

Remark 22. The model $\mathbf{FinVec}_!$ is not fully abstract. Indeed, consider the two valid typing judgments $x : \mathtt{Bool} \vdash \mathtt{tt} : \mathtt{Bool}$ and $x : \mathtt{Bool} \vdash \mathtt{if}\, x\, \mathtt{then}\, \mathtt{tt}\, \mathtt{else}\, \mathtt{tt} : \mathtt{Bool}$. The denotations of both of these judgments are linear maps $!\llbracket \mathtt{Bool} \rrbracket^{\mathrm{vec}} \to \llbracket \mathtt{Bool} \rrbracket^{\mathrm{vec}}$. According to the rules of Table 4, the denotation of the first term is the constant function sending all non-zero vectors b_- to $\mathtt{tt}$.

For the second term, suppose that $v \in !\llbracket \mathtt{Bool} \rrbracket^{\mathrm{vec}}$ is equal to $\sum_i \gamma_i \cdot b_{\alpha_i \cdot \mathtt{tt} + \beta_i \cdot \mathtt{ff}}$. Let $\nu = \sum_i \gamma_i(\alpha_i + \beta_i)$. Then since $\llbracket x : \mathtt{Bool} \vdash x : \mathtt{Bool} \rrbracket^{\mathrm{vec}}(v) = \nu$, the denotation of the second term is the function sending v to $\nu \cdot \llbracket x : \mathtt{Bool} \vdash \mathtt{tt} : \mathtt{Bool} \rrbracket^{\mathrm{vec}}(v)$, equal to $\nu \cdot \mathtt{tt}$ from what we just discussed. We conclude that if $v = b_0$, then $\nu = 0$: the denotation of $x : \mathtt{Bool} \vdash \mathtt{if}\, x\, \mathtt{then}\, \mathtt{tt}\, \mathtt{else}\, \mathtt{tt} : \mathtt{Bool}$ sends b_0 to 0.

Nonetheless, they are clearly operationally equivalent in $\mathbf{PCF}_f$ since their denotation in **FinSet** is the same. The language is not expressive enough to distinguish between these two functions. Note that there exists operational settings where these would actually be different, for example if we were to allow divergence.

Remark 23. Given a term A, another question one could ask is whether the set of terms $M : A$ in $\mathbf{PCF}_f$ generates a free family of vectors in the vector space $\llbracket A \rrbracket^{\mathrm{vec}}$. It turns out not: The field structure brought into the model introduces interferences, and algebraic sums coming from operationally distinct terms may collapse to a representable element. For example, supposing for simplicity that the characteristic of the field is $q = 2$, consider the terms $T_{\mathtt{tt},\mathtt{tt}}$, $T_{\mathtt{ff},\mathtt{ff}}$, $T_{\mathtt{tt},\mathtt{ff}}$ and $T_{\mathtt{ff},\mathtt{tt}}$ defined as $T_{\mathtt{y},\mathtt{z}} = \lambda x.\mathtt{if}\, x\, \mathtt{then}\, \mathtt{y}\, \mathtt{else}\, \mathtt{z}$, all of types $\mathtt{Bool} \to \mathtt{Bool}$. They are clearly operationally distinct, and their denotations live in $!\mathtt{Bool} \multimap \mathtt{Bool}$. They can be written as a 2×4 matrices along the bases $(b_0, b_{\mathtt{tt}}, b_{\mathtt{ff}}, b_{\mathtt{tt}+\mathtt{ff}})$ for the domain and $(\mathtt{tt}, \mathtt{ff})$ for the range. The respective images of the 4 terms are $\left(\begin{smallmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{smallmatrix}\right)$ and clearly, $\llbracket T_{\mathtt{tt},\mathtt{tt}} \rrbracket^{\mathrm{vec}} = \llbracket T_{\mathtt{ff},\mathtt{ff}} \rrbracket^{\mathrm{vec}} + \llbracket T_{\mathtt{tt},\mathtt{ff}} \rrbracket^{\mathrm{vec}} + \llbracket T_{\mathtt{ff},\mathtt{tt}} \rrbracket^{\mathrm{vec}}$.

So if the model we are interested in is $\mathbf{FinVec}_!$, the language is missing some structure to correctly handle the algebraicity.

5 Two Algebraic Lambda Calculi

As discussed in Section 4.3.5, there are two possible ways for incorporing a "vectorial side-effect" into the language: either a call-by-name interpretation (with **FreeCoalg**, or equivalently $\mathbf{FinVec}_!$ as model), or a call-by-value

$$\alpha \cdot M + \beta \cdot M \to (\alpha + \beta) \cdot M \qquad \alpha \cdot (M + N) \to \alpha \cdot M + \alpha \cdot N$$
$$\alpha \cdot M + M \to (\alpha + 1) \cdot M \qquad 0 \cdot M \to 0 \qquad 1 \cdot M \to M$$
$$M + M \to (1 + 1) \cdot M \qquad \alpha \cdot 0 \to 0 \qquad 0 + M \to M$$
$$(M + N) + P \to M + (N + P) \qquad M + N \to N + M$$
$$\alpha \cdot (\beta \cdot M) \to (\alpha\beta) \cdot M$$

Table 6: Rewrite system for the algebraic fragment of $\mathbf{PCF}_f^{alg}$.

interpretation (with $\mathbf{FreeCoalg}^+$ as canonical model). In this section, we shall explore both extensions of the calculus already introduced.

5.1 A Common Term Grammar

We first augment the language $\mathbf{PCF}_f$ with an algebraic structure to mimic the notion of linear distribution. The extended language, called $\mathbf{PCF}_f^{alg}$, is a call-by-name variation of the linear-algebraic lambda calculus [4, 2], and it reads as follows:

$$\begin{array}{rl} M, N, P ::= & x \mid \lambda x.M \mid MN \mid \pi_l(M) \mid \pi_r(M) \mid \langle M, N \rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff} \mid \\ & \mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P \mid \mathtt{let}\, \star = M\, \mathtt{in}\, N \mid \\ & 0 \mid M + N \mid \alpha \cdot M, \\ A, B ::= & \mathbf{1} \mid \mathtt{Bool} \mid A \to B \mid A \times B. \end{array}$$

The scalar α ranges over the field.

The typing rules are as usual for the ordinary constructs. The new constructs are typed as follows: for all A we have $\Delta \vdash 0 : A$, and, given that $\Delta \vdash M, N : A$, then $\Delta \vdash M + N : A$ and $\Delta \vdash \alpha \cdot M : A$.

5.2 A Call-by-Name Rewrite System

In this call-by-name language, the values are

$$U, V ::= x \mid \lambda x.M \mid \langle M, N \rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff} \mid 0 \mid U + V \mid \alpha \cdot U.$$

To account for the new structure, the operational semantics of the language are modified as follows.

1) We add the set of algebraic rewrite rules shown in Table 6. We shall explicitly talk about *algebraic rewrite rules* when referring to these additional rules. The top row consists of the associativity and commutativity (AC) rules. We shall use the term *modulo AC* when referring to a rule or property that is true when not regarding AC rules. For example, modulo AC the term $\star$ is in normal form and $\alpha \cdot M + (N + \alpha \cdot P)$ reduces to $\alpha \cdot (M + P) + N$.

2) We define the relationship between the algebraic structure and the other constructs. One says that a construct $c(-)$ is *distributive* when for all M, N, $c(M + M) \to c(M) + c(N)$, $c(\alpha \cdot M) \to \alpha \cdot c(M)$ and $c(0) \to 0$. The following constructs are distributive: $(-)P$, $\mathtt{if}\,(-)\,\mathtt{then}\,P_1\,\mathtt{else}\,P_2$, $\pi_i(-)$, $\mathtt{let}\,\star = (-)\,\mathtt{in}\,N$, and the pairing construct factors: $\langle\, M, N \,\rangle + \langle\, M', N' \,\rangle \to \langle\, M + M', N + N' \,\rangle$, $\alpha \cdot \langle\, M, N \,\rangle \to \langle\, \alpha \cdot M, \alpha \cdot N \,\rangle$ and $0^{A \times B} \to \langle\, 0^A, 0^B \,\rangle$.

3) We add two congruence rules. If $M \to M'$, then $M + N \to M' + N$ and $\alpha \cdot M \to \alpha \cdot M'$.

Remark 24. Note that if $(M_1 + M_2)(N_1 + N_2)$ reduces to $M_1(N_1 + N_2) + M_2(N_1 + N_2)$, it does *not* reduce to $(M_1 + M_2)N_1 + (M_1 + M_2)N_2$. If it did, one would get an inconsistent calculus [4]. For example, the term $(\lambda x.\langle\, x, x \,\rangle)(\mathtt{tt} + \mathtt{ff})$ would reduce both to $\langle\, \mathtt{tt}, \mathtt{tt} \,\rangle + \langle\, \mathtt{ff}, \mathtt{ff} \,\rangle$ and to $\langle\, \mathtt{tt}, \mathtt{tt} \,\rangle + \langle\, \mathtt{ff}, \mathtt{ff} \,\rangle + \langle\, \mathtt{tt}, \mathtt{ff} \,\rangle + \langle\, \mathtt{ff}, \mathtt{tt} \,\rangle$. Observe that this property reflects the discussion in Section 4.3.5. Later (in Section 5.3), we shall analyze a reduction system were the other reduction is permitted.

5.2.1 Properties

The algebraic extension preserves the safety properties, the characterization of values, and strong normalization. Associativity and commutativity, however, must be taken into account.

Lemma 25 (Safety properties modulo AC)**.** *Progress: A closed, well-typed term $M : A$ is a value or, if not, reduces to some N via a sequence of steps among which one is* not *algebraic. Type Preservation: If $\Delta \vdash M : A$ and M reduces to N, then $\Delta \vdash N : A$.*

Proof. The proof of progress is done by induction on the type derivation of $M : A$. As it is customary, the proof of type preservation uses a substitution sub-lemma stating that if $\Delta \vdash N : A$ and $\Delta, x : A \vdash M : B$ then we also have that $\Delta \vdash M[N/x] : B$. □

Lemma 26. *Any value of type* **1** *has AC-normal form* 0, $\star$ *or* $\alpha \cdot \star$, *with* $\alpha \neq 0, 1$.

Proof. Proof by case analysis. □

Strong normalization is only valid modulo AC. As in [4], we use an auxiliary lemma discussing the algebraic fragment of the reduction-rules.

Lemma 27. *Modulo AC, the algebraic fragment of* $\mathbf{PCF}_f^{alg}$ *is strongly normalizing.*

Proof. The proof can be done as in [4], using the same measure on terms that decreases with algebraic rewrites which are not associativity or commutativity. The measure, written a, is defined by $a(x) = 1$, $a(M+N) = 2+a(M)+a(N)$, $a(\alpha \cdot M) = 1 + 2a(M)$, $a(0) = 0$. □

Lemma 28. *Modulo AC,* $\mathbf{PCF}_f^{alg}$ *is strongly normalizing.*

Proof. The proof is done by defining an intermediate language $\mathbf{PCF}_{f\,int}$ where scalars are omitted. Modulo AC, this language is essentially the language $\lambda_{-\mathbf{wLK}^{\to}}$ of [8], and is therefore SN. Any term of $\mathbf{PCF}_f^{alg}$ can be re-written as a term of $\mathbf{PCF}_{f\,int}$. With Lemma 25, by eliminating some algebraic steps a sequence of reductions in $\mathbf{PCF}_f^{alg}$ can be rewritten as a sequence of reductions in $\mathbf{PCF}_{f\,int}$. We conclude with Lemma 27, saying there is always a finite number of these eliminated algebraic rewrites. □

5.2.2 Operational Equivalence

As for $\mathbf{PCF}_f$, we define an operational equivalence on terms of the language $\mathbf{PCF}_f^{alg}$. A *context* $C[-]$ for this language has the same grammar as for $\mathbf{PCF}_f$, augmented with algebraic structure: $C[-] ::= \alpha \cdot C[-] \mid C[-] + N \mid M + C[-] \mid 0$.

For $\mathbf{PCF}_f^{alg}$, instead of using closed contexts of type `Bool`, we shall use contexts of type $\mathbf{1}$: thanks to Lemma 26, there are distinct normal forms for values of type $\mathbf{1}$, making this type a good (and slightly simpler) candidate.

We therefore say that $\Delta \vdash M : A$ and $\Delta \vdash N : A$ are operationally equivalent, written $M \simeq_{\text{op}} N$, if for all closed contexts $C[-]$ of type $\mathbf{1}$ where the hole binds Δ, for all α element of the field, $C[M] \to^* \alpha \cdot \star$ if and only if $C[N] \to^* \alpha \cdot \star$.

5.2.3 Axiomatic Equivalence

The axiomatic equivalence on $\mathbf{PCF}_f^{alg}$ consists of the one of $\mathbf{PCF}_f$, augmented with the added reduction rules.

Lemma 29. *If $M : A$ and $M \to N$ then $M \simeq_{\mathrm{ax}} N$.* □

5.2.4 Finite Vector Spaces as a Model

The category $\mathbf{FinVec}_!$ is a denotational model of the language $\mathbf{PCF}_f^{alg}$. Types are interpreted as for the language $\mathbf{PCF}_f$ in Section 4.4. Typing judgments are also interpreted in the same way, with the following additional rules. First, $[\![\Delta \vdash 0 : A]\!]^{\mathrm{vec}} = 0$. Then $[\![\Delta \vdash \alpha \cdot M : A]\!]^{\mathrm{vec}} = \alpha \cdot [\![\Delta \vdash M : A]\!]^{\mathrm{vec}}$. Finally, we have $[\![\Delta \vdash M + N : A]\!]^{\mathrm{vec}} = [\![\Delta \vdash M : A]\!]^{\mathrm{vec}} + [\![\Delta \vdash N : A]\!]^{\mathrm{vec}}$.

Remark 30. With the extended term constructs, the language $\mathbf{PCF}_f^{alg}$ does not share the drawbacks of $\mathbf{PCF}_f$ emphasized in Remark 22. In particular, the two valid typing judgments $x : \mathtt{Bool} \vdash \mathtt{tt} : \mathtt{Bool}$ and $x : \mathtt{Bool} \vdash \mathtt{if}\, x\, \mathtt{then}\, \mathtt{tt}\, \mathtt{else}\, \mathtt{tt} : \mathtt{Bool}$ are now operationally distinct. For example, if one chooses the context $C[-]$ to be the application $(\lambda x.[-])0$, the term $C[\mathtt{tt}]$ reduces to $\mathtt{tt}$ whereas the term $C[\mathtt{if}\, x\, \mathtt{then}\, \mathtt{tt}\, \mathtt{else}\, \mathtt{tt}]$ reduces to 0.

Lemma 31. *If $M \simeq_{\mathrm{ax}} N : A$ in $\mathbf{PCF}_f^{alg}$ then $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$.* □

Theorem 32. *Let $\Delta \vdash M, N : A$ be two valid typing judgments in $\mathbf{PCF}_f^{alg}$. If $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$ then we also have $M \simeq_{\mathrm{op}} N$.*

Proof. The proof is similar to the proof of Theorem 6: Assume $M \not\simeq_{\mathrm{op}} N$. Then there exists a context $C[-]$ that distinguishes them. The call-by-name reduction preserves the type from Lemma 25, and $C[M]$ and $C[N]$ can be rewritten as the terms $(\lambda y.C[y\, x_1 \dots x_n])\lambda x_1 \dots x_n.M$ and $(\lambda y.C[y\, x_1 \dots x_n])\lambda x_1 \dots x_n.N$, and these are axiomatically equivalent to distinct normal forms, from Lemmas 28 and 29. We conclude from Lemmas 29 and 31 that the denotations of M and N are distinct. □

5.2.5 Two Auxiliary Constructs

Full completeness requires some machinery. It is obtained by showing that for every type A, for every vector v in $[\![A]\!]^{\mathrm{vec}}$, there are two terms $M_v^A : A$ and $\delta_v^A : A \to \mathbf{1}$ such that $[\![M_v^A]\!]^{\mathrm{vec}} = v$ and $[\![\delta_v^A]\!]^{\mathrm{vec}}$ sends b_v to $\star$ and all other b_-'s to 0.

We first define a family of terms $\mathtt{exp}^i : \mathbf{1} \to \mathbf{1}$ inductively on i: $\mathtt{exp}^0 = \lambda x.\star$ and $\mathtt{exp}^{i+1} = \lambda x.\mathtt{let} \star = x\, \mathtt{in}\, \mathtt{exp}^i(x)$. One can show that $[\![\mathtt{exp}^i(\alpha \cdot \star)]\!]^{\mathrm{vec}} = \alpha^i \cdot \star$. Then assume that o is the order of the field. Let $\mathtt{iszero} : \mathbf{1} \to \mathbf{1}$ be the term $\mathtt{exp}^o$. The denotation of $\mathtt{iszero}$ is such that $[\![\mathtt{izero}(\alpha \cdot \star)]\!]^{\mathrm{vec}} = 0$ if $\alpha = 0$ and $\star$ otherwise.

The mutually recursive definitions of δ_v^A and M_v^A read as follows.

At type $A = \mathbf{1}$. The term $M^{\mathbf{1}}_{\alpha\cdot\star}$ is simply $\alpha \cdot \star$. The term $\delta^{\mathbf{1}}_{\alpha\cdot\star}$ is defined as the lambda-abstraction $\lambda x.\mathtt{iszero}(x - \alpha \cdot \star)$.

At type $A = \mathtt{Bool}$. As for the type $\mathbf{1}$, the term $M^{\mathtt{Bool}}_{\alpha\cdot\mathtt{tt}+\beta\cdot\mathtt{ff}}$ is simply $\alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff}$. The term $\delta^{\mathtt{Bool}}_{\alpha\cdot\mathtt{tt}+\beta\cdot\mathtt{ff}}$ is reusing the definition of $\delta^{\mathbf{1}}$: it is the term $\lambda x.\mathtt{let}\,\star = \delta^{\mathbf{1}}_{\alpha\cdot\star}(\mathtt{if}\,x\,\mathtt{then}\,\star\,\mathtt{else}\,0)\,\mathtt{in}\;\delta^{\mathbf{1}}_{\beta\cdot\star}(\mathtt{if}\,x\,\mathtt{then}\,0\,\mathtt{else}\,\star)$.

At type $A = B \times C$. If $v \in [\![A]\!]^{\mathrm{vec}} = [\![B]\!]^{\mathrm{vec}} \times [\![C]\!]^{\mathrm{vec}}$, then $v = \langle\, u, w \,\rangle$, with $u \in [\![B]\!]^{\mathrm{vec}}$ and $w \in [\![C]\!]^{\mathrm{vec}}$. By induction, one can construct M_u^B and M_w^C: the term $M_v^{B\times C}$ is $\langle\, M_u^B, M_w^C \,\rangle$. Similarly, one can construct the terms δ_u^B and δ_w^C: the term $\delta_v^{B\to C}$ is $\lambda x.\mathtt{let}\,\star = \delta_u^B\,\pi_l(x)\,\mathtt{in}\,\delta_w^C\,\pi_r(x)$.

At type $A = B \to C$. Consider $f \in [\![A]\!]^{\mathrm{vec}} = !{[\![B]\!]}^{\mathrm{vec}} \multimap [\![C]\!]^{\mathrm{vec}}$. The domain of f is finite-dimensional: let $\{b_{u_i}\}_{i=1\ldots n}$ be its basis, and let w_i be the value $f(b_{u_i})$. Then, using the terms $\delta^B_{u_i}$ and $M^C_{w_i}$, one can define $M_v^{B\to C}$ as the term $\sum_i \lambda x.\mathtt{let}\,\star = \delta^B_{u_i}\,x\,\mathtt{in}\,M^C_{w_i}$. Similarly, one can construct $\delta^C_{w_i}$ and $M^B_{u_i}$, and from the construction in the previous paragraph we can also generate $\delta^{C^{\times n}}_{\langle\, w_1,\ldots w_n\,\rangle} : C^{\times n} \to \mathtt{Bool}$. The term $\delta_v^{B\to C}$ is then $\lambda f.\delta^{C^{\times n}}_{\langle\, w_1,\ldots w_n\,\rangle}\,\langle\, f\,M^B_{u_1}, \ldots, f\,M^B_{u_1}\,\rangle$.

5.2.6 Full Completeness

We are now ready to state completeness, whose proof is simply by observing that any $v \in [\![A]\!]^{\mathrm{vec}}$ can be realized by the term $M_v^A : A$.

Theorem 33 (Full completeness)**.** *For any type A, any vector v of $[\![A]\!]^{\mathrm{vec}}$ in* $\mathbf{FinVec}_!$ *is representable in the language* $\mathbf{PCF}_f^{alg}$. □

A corollary of the full completeness is that the category $\mathbf{FinVec}_!$ is also fully abstract (thus adequate) with respect to $\mathbf{PCF}_f^{alg}$ with the call-by-name semantics.

Corollary 34 (Full abstraction)**.** *For all M and N, with $\Delta \vdash M, N : A$, we have $M \simeq_{\mathrm{op}} N$ if and only if $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$.* □

5.3 A Call-by-Value Rewrite System

Remark 24 mentioned that in call-by-name evaluation the term $(M_1 + M_2)(N_1 + N_2)$ would not reduce to $M_1N_1 + M_1N_2 + M_2N_1 + M_2N_2$. In this section, we present a rewrite system for the term grammar $\mathbf{PCF}_f^{alg}$ where

it is in fact allowed. To compensate, the term $(\lambda x.M)N$ will not reduce to $M[N/x]$ for arbitrary N, but only when N is a *base term*.

In this call-by-value language, the values and the base terms are respectively defined as follows:

$$\begin{array}{lll} U, V & ::= & x \mid \lambda x.M \mid \langle\, U, V \,\rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff} \mid 0 \mid U + V \mid \alpha \cdot U. \\ B & ::= & x \mid \lambda x.M \mid \langle\, B_1, B_2 \,\rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff}. \end{array}$$

Intuitively, base terms are values which are *not* linear combinations.

The rewrite rules are modified from the rules of $\mathbf{PCF}_f$ as follows. If $M \to_{\mathrm{cbv}} N$ in this system, we say that M cbv-reduces to N.

1) The set of algebraic rewrite rules shown in Table 6 still holds in this call-by-value language.

2) We again define the relationship between the algebraic structure and the other language features, reusing the definition of *distributive* construct. The following constructs are distributive: $(-)P$, $M(-)$, $\mathtt{if}\,(-)\,\mathtt{then}\,P_1\,\mathtt{else}\,P_2$, $\pi_i(-)$, $\mathtt{let}\,\star = (-)\,\mathtt{in}\,N$, $\langle\,(-), N\,\rangle$, $\langle\, M, ()\,\rangle$

3) We add wo congruence rules. If $M \to_{\mathrm{cbv}} M'$, then $M + N \to_{\mathrm{cbv}} M' + N$ and $\alpha \cdot M \to_{\mathrm{cbv}} \alpha \cdot M'$.

4) Finally, the reduction $(\lambda x.M)N \to_{\mathrm{cbv}} M[N/x]$ is permitted only when N is a *base term*.

Remark 35. Note that now, the pairing construct is distributives, as is the application, on both sides. Also note that we ask for the argument of the beta-redex to be a base term.

5.3.1 Properties

As in the call-by-name case, the algebraic extension preserves the safety properties, the characterization of values and the strong normalization modulo AC. The proofs of the two first lemmas are exactly similar to the case of the call-by-name strategy.

Lemma 36 (Safety properties modulo AC)**.** *A closed, well-typed term $M : A$ is a value or, if not, reduces to some N via a sequence of steps among which one is* not *algebraic. If $\Delta \vdash M : A$ and M cbv-reduces to N, then $\Delta \vdash N : A$.* □

Lemma 37. *In the call-by-value reduction strategy, any value of type* **1** *has AC-normal form 0, $\star$ or $\alpha \cdot \star$, with $\alpha \neq 0, 1$.* □

For strong normalization, we reuse a result known in the literature [3].

Lemma 38. *Modulo AC,* $\mathbf{PCF}_f^{alg}$ *is strongly normalizing with respect to the call-by-value reduction strategy.* □

Proof. The call-by-value lambda-calculus $\mathbf{PCF}_f^{alg}$ can be encoded in the language $\lambda 2^{la}$ presented in [3]: as its type system is system-F-like, one can encode booleans and pairs in the regular way. Then strong normalization of $\lambda 2^{la}$ without the constraints on the Application Rules implies strong normalization modulo AC of $\mathbf{PCF}_f^{alg}$ with call-by-value reduction rules. □

5.3.2 Operational and Axiomatic Equivalence

We define an operational equivalence on terms of the language $\mathbf{PCF}_f^{alg}$, using the same notion of context as in Section 5.2.2.

We therefore say that $\Delta \vdash M : A$ and $\Delta \vdash N : A$ are operationally equivalent, written $M \simeq_{\mathrm{op}}^{\mathrm{cbv}} N$, if for all closed contexts $C[-]$ of type $\mathbf{1}$ where the hole binds Δ, for all b normal forms of type $\mathbf{1}$, $C[M] \to_{\mathrm{cbv}}{}^* b$ if and only if $C[N] \to_{\mathrm{cbv}}{}^* b$.

The call-by-value axiomatic equivalence on $\mathbf{PCF}_f^{alg}$ consists in the reflexive, symmetric and transitive closure of the call-by-value reduction rules.

Lemma 39. *If* $M : A$ *and* $M \to_{\mathrm{cbv}} N$ *then* $M \simeq_{\mathrm{ax}}^{\mathrm{cbv}} N$. □

5.3.3 FreeCoalg$^+$ as a Model

For all types but `Bool`, there exist natural interpretations in the model **FreeCoalg**$^+$. We first describe the denotation of the language without `Bool`: booleans and tests are discussed later in the next paragraph.

Interpreting the core lambda-term constructs. The unit type $[\![\mathbf{1}]\!]^{\mathrm{cbv}}$ is mapped to the zero-vector space $\langle 0 \rangle$. The element b_0 is denoted with $\star$. The product is mapped to the product of vector spaces: remembering that $![\![A]\!]^{\mathrm{cbv}} \otimes ![\![B]\!]^{\mathrm{cbv}}$ is isomorphic to $!([\![A]\!]^{\mathrm{cbv}} \times [\![B]\!]^{\mathrm{cbv}})$, define $[\![A \times B]\!]^{\mathrm{cbv}} := [\![A]\!]^{\mathrm{cbv}} \times [\![B]\!]^{\mathrm{cbv}}$. Finally, the arrow type $[\![A \to B]\!]^{\mathrm{cbv}}$ is the internal hom $![\![A]\!]^{\mathrm{cbv}} \multimap [\![B]\!]^{\mathrm{cbv}}$.

A typing judgment $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ is represented by a morphism of **FinVec** of type

$$![\![A_1]\!]^{\mathrm{cbv}} \otimes \cdots \otimes ![\![A_n]\!]^{\mathrm{cbv}} \longrightarrow ![\![B]\!]^{\mathrm{cbv}}, \tag{3}$$

$$
\begin{array}{rcl}
[\![\Delta, x : A \vdash x : A]\!]^{\mathrm{cbv}} & = & d \otimes b_a \longmapsto b_a \\
[\![\Delta \vdash \star : \mathbf{1}]\!]^{\mathrm{cbv}} & = & d \longmapsto \star \\
[\![\Delta \vdash \langle\, M, N \,\rangle : A \times B]\!]^{\mathrm{cbv}} & = & d \longmapsto [\![M]\!]^{\mathrm{cbv}}(d) \otimes [\![N]\!]^{\mathrm{cbv}}(d) \\
[\![\Delta \vdash MN : B]\!]^{\mathrm{cbv}} & = & d \longmapsto (\epsilon([\![M]\!]^{\mathrm{cbv}}(d)))([\![N]\!]^{\mathrm{cbv}}(d)) \\
[\![\Delta \vdash \pi_l(M) : A]\!]^{\mathrm{cbv}} & = & [\![M]\!]^{\mathrm{cbv}}; \pi_l \\
[\![\Delta \vdash \pi_r(M) : B]\!]^{\mathrm{cbv}} & = & [\![M]\!]^{\mathrm{cbv}}; \pi_r \\
[\![\Delta \vdash \lambda x.M : A \to B]\!]^{\mathrm{cbv}} & = & d \longmapsto b_{\sum_i \alpha_i \cdot g_i(d)} \\
 & & \text{where } [\![M]\!]^{\mathrm{cbv}} = \sum_i \alpha_i f_i \\
 & & \text{and } f_i(d \otimes b_a) = b_{g_i(d)(b_a)} \\
[\![\Delta \vdash \mathtt{let}\, \star = M\, \mathtt{in}\, N : A]\!]^{\mathrm{cbv}} & = & d \longmapsto \alpha \cdot [\![N]\!]^{\mathrm{cbv}}(d) \\
 & & \text{where } [\![M]\!]^{\mathrm{cbv}}(d) = \alpha \cdot \star. \\
[\![\Delta \vdash M + N : A]\!]^{\mathrm{cbv}} & = & [\![M]\!]^{\mathrm{cbv}} + [\![N]\!]^{\mathrm{cbv}} \\
[\![\Delta \vdash \alpha \cdot M : A]\!]^{\mathrm{cbv}} & = & \alpha \cdot [\![M]\!]^{\mathrm{cbv}} \\
[\![\Delta \vdash 0 : A]\!]^{\mathrm{cbv}} & = & 0
\end{array}
$$

Table 7: Modeling the language $\mathbf{PCF}_f^{alg}$ in $\mathbf{FreeCoalg}^+$: the main constructs.

made of linear combinations of free-coalgebra maps of type

$$!([\![A_1]\!]^{\mathrm{cbv}} \times \cdots \times [\![A_n]\!]^{\mathrm{cbv}}) \longrightarrow ![\![B]\!]^{\mathrm{cbv}}.$$

The morphisms of Eq. (3) are inductively defined as in Table 7 (constants of type `Bool` and the `if-then-else` construct are discussed later). The variable d stands for a base element $b_{u_1} \otimes \ldots \otimes b_{u_n}$ of $[\![\Delta]\!]^{\mathrm{cbv}}$, and b_a is a base element of $![\![A]\!]^{\mathrm{cbv}}$. The functions π_l and π_r are the left and right projections of the product.

Interpretation of the boolean type. The remaining type to be interpreted is the boolean type `Bool`. If we follow the call-by-name interpretation, in $\mathbf{FreeCoalg}^+$ it should correspond to the (weak) coproduct of I with itself.

$$\begin{aligned}
[\![\Delta \vdash \mathtt{tt} : \mathtt{Bool}]\!]^{\mathrm{cbv}} &= d \longmapsto \mathtt{tt} \\
[\![\Delta \vdash \mathtt{ff} : \mathtt{Bool}]\!]^{\mathrm{cbv}} &= d \longmapsto \mathtt{ff} \\
[\![\Delta \vdash \mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P : A]\!]^{\mathrm{cbv}} &= d \longmapsto \alpha \cdot [\![N]\!]^{\mathrm{cbv}}(d) + \beta \cdot [\![P]\!]^{\mathrm{cbv}}(d) \\
&\quad \text{where } [\![M]\!]^{\mathrm{cbv}}(d) = \alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff}.
\end{aligned}$$

Table 8: Modeling the language $\mathbf{PCF}_f^{alg}$ in $\mathbf{FreeCoalg}^+$: the type `Bool`.

Weeding out the exterior "!", we would get $[\![\mathtt{Bool}]\!]^{\mathrm{cbv}} := !I \oplus !I$. But as this vector space contains all $\langle \alpha, \beta \rangle$ when $\alpha, \beta \in K$, it has $|K|^2$ elements. We would like the space to contain only 2 elements, so vectors can be identified with `ff` and `tt`, as it happens for the call-by-name interpretation. We thus cannot use the weak coproduct for this task.

Since any vector space contains at least the zero vector, the only possibility to get a vector space V consisting of only 2 elements is to choose the one-dimensional vector space $\mathbb{F}_2$.

Note that this particular vector space belongs to **FinVec** only when the field K is equal to $\mathbb{F}_2$: for any other field, there is no such V. For that reason, let us consider from now on in this section that the field K is equal to $\mathbb{F}_2$, and define $[\![\mathtt{Bool}]\!]^{\mathrm{cbv}}$ as the vector space $\mathbb{F}_2$. We use some shortcut notations: `ff` stands for b_0 and `tt` stands for b_1. Then the denotation of the lambda-terms `tt`, `ff`, and of the `if-then-else` constructs are defined in Table 8.

5.3.4 Properties of the Model

Despite the somewhat arbitrary interpretation of `Bool`, the denotation of the call-by-value system in $\mathbf{FreeCoalg}^+$ is very well-behaved.

Lemma 40. *If B is a base term and if $\Delta \vdash B : A$, then $[\![B]\!]^{\mathrm{cbv}} : [\![\Delta]\!]^{\mathrm{cbv}} \to ![\![A]\!]^{\mathrm{cbv}}$ is an element of* **FreeCoalg**. *That is, it is a free-coalgebra morphisms, and sends each b_v to $b_{g(v)}$, for some set-function g.*

Proof. The proof is done by structural induction on B. It is clear for constant terms. It is also trivially correct for lambda-abstractions by definition. Finally, it is true for products, since $b_v \otimes b_u$ is assimilated with $b_{\langle v,u \rangle}$. □

Lemma 41. *Let B be a base term such that $\Delta \vdash B : A$. If $\Delta, x : A \vdash M : B$, then $[\![\Delta \vdash M[B/x] : B]\!]^{\mathrm{cbv}}(d) = [\![M]\!]^{\mathrm{cbv}}(d \otimes ([\![B]\!]^{\mathrm{cbv}}(d)))$.*

Proof. The proof is done by structural induction on the derivation of $\Delta, x : A \vdash M : B$. We use the fact that the denotation of B is not a linear combination (deduced from Lemma 40) for the cases of the application and of the pair. □

Lemma 42. *If $\Delta \vdash M, N : A$ and $M \simeq^{\mathrm{cbv}}_{\mathrm{ax}} N$, then $[\![M]\!]^{\mathrm{cbv}} = [\![N]\!]^{\mathrm{cbv}}$.*

Proof. The proof is done by induction on the derivation of $M \simeq^{\mathrm{cbv}}_{\mathrm{ax}} N$. For the beta-redex, one uses Lemma 41. □

Theorem 43. *Let $\Delta \vdash M, N : A$ be two valid typing judgments in $\mathbf{PCF}^{alg}_f$. If $[\![M]\!]^{\mathrm{cbv}} = [\![N]\!]^{\mathrm{cbv}}$ then we also have $M \simeq^{\mathrm{cbv}}_{\mathrm{op}} N$.*

Proof. The proof is exactly similar to the proof of Theorem 32, using Lemma 36, 38, 39 and 42. □

As in the case of the call-by-name language, the other direction is implied by a full completeness theorem: for all types A, any vector v in $[\![A]\!]^{\mathrm{cbv}}$ is representable by a closed term in $\mathbf{PCF}^{alg}_f$.

5.3.5 Full Completeness Theorem

We follow the same strategy as in Section 5.2.5: for every type A, for every vector v in $[\![A]\!]^{\mathrm{cbv}}$, we describe two closed terms $\bar{M}^A_v : A$ and $\bar{\delta}^A_v : A \to \mathbf{1}$ such that $[\![\bar{M}^A_v]\!]^{\mathrm{cbv}}(\star) = b_v$ and $[\![\bar{\delta}^A_v]\!]^{\mathrm{cbv}}$ sends b_v to $\star$ and all other b_-'s to 0.

The terms $\bar{M}^A_v$ and $\bar{\delta}^A_v$ are defined by mutual recursion on A as follows. The base cases are specific to call-by-value, whereas the two inductive cases for product and arrow-type are similar to what was done in call-by-name.

At type $A = \mathbf{1}$. v is a vector in the zero vector space $\langle 0 \rangle$: the term $\bar{M}^A_v$ is simply $\alpha \cdot \star$, and $\bar{\delta}^A_v$ is the identity $\lambda x.x$.

At type $A = \mathtt{Bool}$. ¿From the definition of $[\![\mathtt{Bool}]\!]^{\mathrm{cbv}}$, the vector v is either $0 \in \mathbb{F}_2$ or $1 \in \mathbb{F}_2$. Then because of the definitions in Table 8, we can define $\bar{M}^{\mathtt{Bool}}_0 = \mathtt{ff}$ and $\bar{M}^{\mathtt{Bool}}_1 = \mathtt{tt}$. We define $\bar{\delta}^{\mathtt{Bool}}_0$ and $\bar{\delta}^{\mathtt{Bool}}_1$ using the test construct:

$$\bar{\delta}^{\mathtt{Bool}}_0 = \lambda x.\mathtt{if}\, x\, \mathtt{then}\, 0\, \mathtt{else}\, \star, \qquad \bar{\delta}^{\mathtt{Bool}}_1 = \lambda x.\mathtt{if}\, x\, \mathtt{then}\, \star\, \mathtt{else}\, 0.$$

At type $A = B \times C$. The vector v is $\langle u, w \rangle$, where $u \in [\![B]\!]^{\mathrm{cbv}}$ and $w \in [\![C]\!]^{\mathrm{cbv}}$. The term $\bar{M}_v^A$ is therefore defined as $\langle\, \bar{M}_u^B, \bar{M}_w^C \,\rangle$. The term $\bar{\delta}_v^A$ is testing its left projection and its right projection:

$$\bar{\delta}_v^A = \lambda x.\mathtt{let} \star = \bar{\delta}_u^B(\pi_l x) \,\mathtt{in}\, \bar{\delta}_w^C(\pi_r x).$$

These two terms are reminiscent of the call-by-name situation.

At type $A = B \to C$. Consider $f \in [\![A]\!]^{\mathrm{cbv}} = !{[\![B]\!]}^{\mathrm{cbv}} \multimap [\![C]\!]^{\mathrm{cbv}}$. The domain of f is finite-dimensional: let $\{b_{u_i}\}_{i=1\ldots n}$ be its basis, and let w_i be the value $f(b_{u_i})$. Then, using the terms $\bar{\delta}_{u_i}^B$ and $\bar{M}_{w_i}^C$, one can define $\bar{M}_v^{B\to C}$ as the term $\lambda x. \sum_i \mathtt{let} \star = \bar{\delta}_{u_i}^B\, x \,\mathtt{in}\, \bar{M}_{w_i}^C$. Note that in the call-by-name, we chose to place the sum outside of the lambda-abstraction. In this call-by-value case, we are forced to place it inside. By invoking the induction principle one can build $\bar{\delta}_{w_i}^C$ and $\bar{M}_{u_i}^B$, and from the construction in the previous paragraph we can also generate $\bar{\delta}^{C^{\times n}}_{\langle w_1,\ldots w_n\rangle} : C^{\times n} \to \mathtt{Bool}$. The term $\bar{\delta}_v^{B\to C}$ is then $\lambda f.\bar{\delta}^{C^{\times n}}_{\langle w_1,\ldots w_n\rangle} \langle\, f\, M_{u_1}^B, \ldots, f\, \bar{M}_{u_1}^B \,\rangle$.

With these two definitions, we can now state and prove the result of full completeness, as follows.

Theorem 44 (Full Completeness)**.** *For any type, any vector* v *of* $[\![A]\!]^{\mathrm{cbv}}$ *is representable in the call-by-value* $\mathbf{PCF}_f^{alg}$*: that is, there exists a typing judgement* $\vdash M : A$ *in* $\mathbf{PCF}_f^{alg}$ *such that* $[\![M]\!]^{\mathrm{cbv}}(\star) = v$.

Proof. The requested term M is simply $\bar{M}_v^A$. □

A corollary of the full completeness is that the category $\mathbf{FreeCoalg}^+$ is also fully abstract (therefore adequate) with respect to $\mathbf{PCF}_f^{alg}$ with the call-by-value semantics.

Corollary 45 (Full abstraction)**.** *For all terms* M *and* N *such that the judgement* $\Delta \vdash M, N : A$ *holds, we have* $M \simeq_{\mathrm{op}}^{\mathrm{cbv}} N$ *if and only if in the category* $\mathbf{FreeCoalg}^+$ *we have* $[\![M]\!]^{\mathrm{cbv}} = [\![N]\!]^{\mathrm{cbv}}$. □

6 Discussion

6.1 Simulating the Vectorial Structure

As we already saw, there is a forgetful functor $U : \mathbf{FinVec}_! \hookrightarrow \mathbf{FinSet}$ preserving the cartesian closed structure. This functor can be understood as "mostly" saying that the vectorial structure "does not count" in $\mathbf{FinVec}_!$, as

$$\begin{aligned}
\phi^{\mathrm{vec}}_{\mathbf{1}} &= \mathtt{let}\star = \delta_0 x\,\mathtt{in}\,\mathtt{tt} + \mathtt{let}\star = \delta_\star x\,\mathtt{in}\,\mathtt{ff}\\
\bar{\phi}^{\mathrm{vec}}_{\mathbf{1}} &= \mathtt{if}\,x\,\mathtt{then}\,0\,\mathtt{else}\star
\end{aligned}$$

$$\begin{aligned}
\phi^{\mathrm{vec}}_{\mathtt{Bool}} = \quad & \mathtt{let}\star = \delta_0 x\,\mathtt{in}\,\langle\,\mathtt{tt},\mathtt{tt}\,\rangle + \mathtt{let}\star = \delta_{\mathtt{tt}} x\,\mathtt{in}\,\langle\,\mathtt{tt},\mathtt{ff}\,\rangle\\
& + \mathtt{let}\star = \delta_{\mathtt{ff}} x\,\mathtt{in}\,\langle\,\mathtt{ff},\mathtt{tt}\,\rangle + \mathtt{let}\star = \delta_{\mathtt{tt}+\mathtt{ff}} x\,\mathtt{in}\,\langle\,\mathtt{ff},\mathtt{ff}\,\rangle
\end{aligned}$$

$$\begin{aligned}
\bar{\phi}^{\mathrm{vec}}_{\mathtt{Bool}} = \mathtt{if}\ (\pi_l x)\,\mathtt{then}\,(\mathtt{if}\,(\pi_r x)\,\mathtt{then}\,0\,\mathtt{else}\,\mathtt{tt})\\
\mathtt{else}\ (\mathtt{if}\,(\pi_r x)\,\mathtt{then}\,\mathtt{ff}\,\mathtt{else}\,\mathtt{tt} + \mathtt{ff})
\end{aligned}$$

$$\begin{aligned}
\phi^{\mathrm{vec}}_{B\times C} &= \langle\, x;\pi_l;\phi^{\mathrm{vec}}_B, x;\pi_r;\phi^{\mathrm{vec}}_C\,\rangle, \qquad & \phi^{\mathrm{vec}}_{B\to C} &= \lambda y.x(y;\bar{\phi}^{\mathrm{vec}}_B);\phi^{\mathrm{vec}}_C,\\
\bar{\phi}^{\mathrm{vec}}_{B\times C} &= \langle\, x;\pi_l;\bar{\phi}^{\mathrm{vec}}_B, x;\pi_r;\bar{\phi}^{\mathrm{vec}}_C\,\rangle, \qquad & \bar{\phi}^{\mathrm{vec}}_{B\to C} &= \lambda y.x(y;\phi^{\mathrm{vec}}_B);\bar{\phi}^{\mathrm{vec}}_C.
\end{aligned}$$

Table 9: Relation between $\mathbf{PCF}_f$ and $\mathbf{PCF}_f^{alg}$

one can simulate it with finite sets. Because of Theorems 8 and 34, on the syntactic side, algebraic terms can also be simulated by the regular $\mathbf{PCF}_f$.

In this section, for simplicity, we assume that the field is $\mathbb{F}_2$. In general, the field can be of any finite size, provided that the regular lambda-calculus $\mathbf{PCF}_f$ is augmented with q-bits, i.e. base types with q elements (where q is the characteristic of the field).

Definition 46. The *vec-to-set* encoding of a type A, written VtoSA, is defined inductively as follows: VtoS($\mathbf{1}$) = $\mathtt{Bool}$, VtoS($\mathtt{Bool}$) = $\mathtt{Bool}\times\mathtt{Bool}$, VtoS$(A\times B)$ = VtoS$(A)\times$VtoS(B), and VtoS$(A\to B)$ = VtoS$(A)\to$ VtoS(B).

Theorem 47. *There are two typing judgments $x : A \vdash \phi^{\mathrm{vec}}_A : \mathrm{VtoS}(A)$ and $x : \mathrm{VtoS}(A) \vdash \bar{\phi}^{\mathrm{vec}}_A : A$, inverse of each other in $\mathbf{PCF}_f^{alg}$ with call-by-name semantics such that any typing judgment $x : A \vdash M : B$ can be factored into*

$$A \xrightarrow{\phi^{\mathrm{vec}}_A} \mathrm{VtoS}(A) \xrightarrow{\tilde{M}} \mathrm{VtoS}(B) \xrightarrow{\bar{\phi}^{\mathrm{vec}}_B} B,$$

where $\tilde{M}$ is a regular lambda-term of $\mathbf{PCF}_f$.

Proof. The two terms ϕ^{vec}_A and $\bar{\phi}^{\mathrm{vec}}_A$ are defined inductively on A. For the definition of $\phi^{\mathrm{vec}}_{\mathtt{Bool}}$ we are reusing the term δ_v of Section 5.2.5. The definition is in Table 9 □

6.1.1 Structures of the Syntactic Categories

Out of the language $\mathbf{PCF}_f$ one can define a syntactic category: objects are types and morphisms $A \to B$ are valid typing judgments $x : A \vdash M : B$

modulo operational equivalence. Thanks to Theorem 8, this category is cartesian closed, and one can easily see that the product of $x : A \vdash M : B$ and $x : A \vdash N : C$ is $\langle M, N \rangle : B \times C$, that the terminal object is $\star : \mathbf{1}$, that projections are defined with π_l and π_r, and that the lambda-abstraction plays the role of the internal morphism.

The language $\mathbf{PCF}_f^{alg}$ almost defines a cartesian closed category: by Theorem 34, it is clear that pairing and lambda-abstraction form a product and an internal hom. However, it is missing a terminal object (the type $\mathbf{1}$ doesn't make one as $x : A \vdash 0 : \mathbf{1}$ and $x : A \vdash \star : \mathbf{1}$ are operationally distinct). There is no type corresponding to the vector space $\langle 0 \rangle$. It is not difficult, though, to extend the language to support it: it is enough to only add a type $\mathbf{0}$. Its only inhabitant will then be the term 0, making it a terminal object for the syntactic category.

Finally, Theorem 47 is essentially giving us a functor $\mathbf{PCF}_f^{alg} \to \mathbf{PCF}_f$ corresponding to the forgetful functor U. This makes a full correspondence between the two models **FinSet** and $\mathbf{FinVec}_!$, and $\mathbf{PCF}_f$ and $\mathbf{PCF}_f^{alg}$, showing that computationally the algebraic structure is virtually irrelevant.

6.2 Call-by-Name and Call-by-Value

There exists a standard trick to model call-by-name by mean of call-by-value: the *thunk* construction. This construction takes a call-by-name term and send it to a call-by-value term of similar behavior. Denotationally, it is using Property 14 to send a map $!A \to B$ in the category **FinVec** to a free-coalgebra morphism $!A \to !B$. For a reason similar to what happened in Section 5.3.5, we choose in this section the field K to be $\mathbb{F}_2$.

Formally, we define an operation $(-)^t$ on types and typed terms of $\mathbf{PCF}_f^{alg}$ as follows. The arrow and the product are encoded with

$$(A \to B)^t := (\mathbf{1} \to A^t) \to B^t, \quad (A \times B)^t := (\mathbf{1} \to A^t) \times (\mathbf{1} \to B^t).$$

We encode the base types using the following intuition. A term of type $\mathbf{1}$ in call-by-name is either the zero-vector, or $\star$ (because the base field is of size 2). Through Property 14, these two distinct maps should be sent to two distinct free-coalgebra maps: we need to send $\mathbf{1}$ to a type whose denotation in call-by-value contains 2 elements. There is one, and it is `Bool`. Similarly, the type `Bool` in call-by-name hosts four distinct linear combinations. Therefore we define

$$\mathbf{1}^t := \texttt{Bool}, \qquad \texttt{Bool}^t := \texttt{Bool} \times \texttt{Bool}. \tag{4}$$

A term $\alpha \cdot \mathtt{ff} + \beta \cdot \mathtt{tt}$ is sent to $\langle\, \alpha, \beta \,\rangle$. With respect to terms, when z is a fresh variable, we define

$$x^t := x\star, \qquad (\lambda x.M)^t := \lambda x.M^t, \qquad (MN)^t := M^t(\lambda z.N^t),$$
$$(\pi_l M)^t := (\pi_l M^t)\star, \qquad (\pi_r M)^t := (\pi_r M^t)\star.$$

For the vectorial term constructs, the operation $(-)^t$ depends on the type of the term, and is defined according to the encoding in Eq. (4). The constructors $\mathcal{Z}_A$ $\mathcal{A}_A$ and $\mathcal{M}_A$ are inductively defined on the type A.

$$\mathcal{Z}_{\mathbf{1}} = \mathtt{ff}, \quad \mathcal{Z}_{\mathtt{Bool}} = \langle\, 0, 0 \,\rangle, \quad \mathcal{Z}_{A\times B} = \langle\, \lambda z.\mathcal{Z}_A, \lambda z.\mathcal{Z}_B \,\rangle, \quad \mathcal{Z}_{A\to B} = \lambda x.\mathcal{Z}_B.$$

The addition is defined using the auxiliary term

$$M \oplus N := \mathtt{if}\, M\, \mathtt{then}\, (\mathtt{if}\, N\, \mathtt{then}\, \mathtt{ff}\, \mathtt{else}\, \mathtt{tt})\, \mathtt{else}\, N$$

explicitly computing the addition in $\mathbb{F}_2$: 0 corresponds to $\mathtt{ff}$ and 1 corresponds to $\mathtt{tt}$. Then

$$\begin{aligned}
\mathcal{A}_{\mathbf{1}}(M,N) &:= M \oplus N,\\
\mathcal{A}_{\mathtt{Bool}}(M,N) &:= \langle \pi_l M \oplus \pi_l N, \pi_r M \oplus \pi_r N\rangle,\\
\mathcal{A}_{A\times B}(M,N) &:= \langle\, \lambda z.\mathcal{A}_A(\pi_l M^t, \pi_l N^t), \lambda z.\mathcal{A}_B(\pi_r M^t, \pi_r N^t)\,\rangle,\\
\mathcal{A}_{A\to B}(M,N) &:= \lambda x.\mathcal{A}_B(M^t x, N^t x).
\end{aligned}$$

Finally,

$$\begin{aligned}
\mathcal{M}^0_{\mathbf{1}}(M) &:= M, & \mathcal{M}^1_{\mathbf{1}}(M) &:= \mathtt{tt} \oplus M,\\
\mathcal{M}^0_{\mathtt{Bool}}(M) &:= M, & \mathcal{M}^1_{\mathtt{Bool}}(M) &:= \langle\, \mathtt{tt} \oplus \pi_l M, \mathtt{tt} \oplus \pi_r M\,\rangle,\\
\mathcal{M}^\alpha_{A\to B}(M) &:= \lambda x.\mathcal{M}^\alpha_B(Mx), & \mathcal{M}^\alpha_{A\times B}(M) &:= \langle\, \mathcal{M}^\alpha_A(\pi_l M), \mathcal{M}^\alpha_B(\pi_r M)\,\rangle.
\end{aligned}$$

Using these definitions, for terms of type A:

$$(M+N)^t := \mathcal{A}_A(M,N), \quad (\alpha \cdot N)^t := \mathcal{M}^\alpha_A(N), \quad (0)^t := \mathcal{Z}_A.$$

Finally, the transformation $(-)^t$ for the last term constructs are now "engineered" according to the definition of $\mathbf{1}^t$ and $\mathtt{Bool}^t$: if we assume N to be of type A,

$$(\mathtt{let}\star = M\, \mathtt{in}\, N)^t := \mathtt{if}\, M^t\, \mathtt{then}\, N^t\, \mathtt{else}\, \mathcal{Z}_A,$$
$$(\mathtt{if}\, P\, \mathtt{then}\, M\, \mathtt{else}\, N)^t := \mathcal{A}_A \begin{pmatrix} \mathtt{if}\, \pi_l P\, \mathtt{then}\, M\, \mathtt{else}\, \mathcal{Z}_A, \\ \mathtt{if}\, \pi_r P\, \mathtt{then}\, \mathcal{Z}_A\, \mathtt{else}\, N \end{pmatrix}.$$

If Δ is the typing context $\{x_i : A_i\}_{i\in I}$, let us write $\mathbf{1} \to \Delta^t$ to stand for the context $\{x_i : \mathbf{1} \to A_i^t\}_{i\in I}$.

Lemma 48. *If* $\Delta \vdash M : A$*, then* $(\mathbf{1} \to \Delta^t) \vdash M : A^t$*.*

Proof. By structural induction over the typing derivation of $\Delta \vdash M : A$. □

Lemma 49. *For all type* A*,* $[\![A]\!]^{\mathrm{vec}} \simeq [\![A^t]\!]^{\mathrm{cbv}}$ *as vector spaces.*

Proof. By induction on A. The proof uses the fact that $[\![1 \to A]\!]^{\mathrm{cbv}}$ is equal to the space $!\langle 0\rangle \multimap [\![A]\!]^{\mathrm{cbv}}$. Since $!\langle 0\rangle$ is really the one-dimensional vector space, the space $[\![1 \to A]\!]^{\mathrm{cbv}}$ is isomorphic to $[\![A]\!]^{\mathrm{cbv}}$. For the base types, we explicitly chose representations of the right size. □

Theorem 50. *If* $\Delta \vdash M : A$*, then the linear map* $[\![\Delta \vdash M : A]\!]^{\mathrm{vec}}$ *in* $\mathbf{FinVec}_!$ *and the map* $[\![(\mathbf{1} \to \Delta^t) \vdash M^t : A^t]\!]^{\mathrm{cbv}}$ *in* **FreeCoalg** *are related through the isomorphisms of Property 14 and Lemma 49:*

$$[\![(\mathbf{1} \to \Delta^t) \vdash M^t : A^t]\!]^{\mathrm{cbv}}(b_x) \quad = \quad ![\![\Delta \vdash M : A]\!]^{\mathrm{vec}}(b_{b_x}).$$

Proof. The proof goes by structural induction on the typing derivation of the judgement $\Delta \vdash M : A$. We give here the application, illustrating how the "thunks" ensure that addition does not distribute over application as it normally does in the call-by-value case.

Let M be the term NP, where $\Delta \vdash N : B \to A$ and $\Delta \vdash P : B$. By induction hypothesis, we know that

$$[\![(\mathbf{1} \to \Delta^t) \vdash N^t : (\mathbf{1} \to A^t) \to B^t]\!]^{\mathrm{cbv}}(b_x) = ![\![\Delta \vdash N : A \to B]\!]^{\mathrm{vec}}(b_{b_x}),$$
$$[\![(\mathbf{1} \to \Delta^t) \vdash P^t : A^t]\!]^{\mathrm{cbv}}(b_x) = ![\![\Delta \vdash P : A]\!]^{\mathrm{vec}}(b_{b_x}).$$

Let $[\![\Delta \vdash N : A \to B]\!]^{\mathrm{vec}}$ and $[\![\Delta \vdash P : A]\!]^{\mathrm{vec}}$ be written respectively f and g. By definition the morphism $[\![(\mathbf{1} \to \Delta^t) \vdash (\lambda z.P^t) : \mathbf{1} \to B^t]\!]^{\mathrm{cbv}}$ sends d to $b_{g(d)}$, when d is a base element. If $f(d)$ is the linear combination $\sum_i \alpha_i \cdot b_{f_i(d)}$, then the morphism $[\![(\mathbf{1} \to \Delta^t) \vdash N^t(\lambda z.P^t) : B^t]\!]^{\mathrm{cbv}}$ is by definition sending d to the sum $\sum_i \alpha_i \cdot f_i(d)(b_{g(d)})$, which is precisely $![\![\Delta \vdash NP : A]\!]^{\mathrm{vec}}(b_{b_x})$.

Note how the potential linear combination $g(d)$ is encapsulated inside $b_{(-)}$, so that it is passed without modification to the function $f_i(d)$. □

6.3 Generalizing to Modules

To conclude this discussion, let us consider a generalization of finite vector spaces to finite modules over finite semi-rings.

Indeed, the model of linear logic this paper uses would work in the context of a finite semi-ring instead of a finite field, as long as addition and

multiplication have distinct units. For example, by using the semiring $\{0, 1\}$ where $1 + 1 = 1$ one recovers sets and relations.

In this case, we could not complete the construction of Section 5.2.5 as it heavily relies on the fact that we have a finite field K. We would therefore not get the completeness result in Theorem 33. This particular construction works because one can construct any function between any two finite vector spaces as polynomial, for the same reason as any function $K \to K$ can be realized as a polynomial.

On another hand, the construction in Section 5.3.5 is only using the fact that the field has two elements: the same construction would therefore proceed for any semi-ring of size 2, and full completeness in the call-by-value setting should therefore hold for sets and relations.

6.4 Conclusion

In this paper, we presented a degenerate model of linear logic based on finite vector spaces and presented three categories related to it: **FinSet**, $\mathbf{FinVec}_!$ and $\mathbf{FreeCoalg}^+$. We explored their computational properties using three lambda-calculi. First, we presented the finitary PCF language $\mathbf{PCF}_f$, and showed how **FinSet** forms a fully complete model for it (Theorem 7). Then we discussed the fitness of $\mathbf{FinVec}_!$ as a model of $\mathbf{PCF}_f$ and proved that if it is adequate (Theorem 21), it is not fully-abstract (Remark 22).

In a second step, we extended the simple language $\mathbf{PCF}_f$ to $\mathbf{PCF}_f^{alg}$, featuring linear combinations of terms, and presented two canonical rewrite systems: one call-by-name and one call-by-value. We demonstrated that $\mathbf{FinVec}_!$ and $\mathbf{FreeCoalg}^+$ are full and complete models for the call-by-name and call-by-value variants of $\mathbf{PCF}_f^{alg}$ (Theorems 33 and 44), respectively.

Finally, in Section 6 we discussed the correspondence between $\mathbf{PCF}_f$ and the call-by-name language $\mathbf{PCF}_f^{alg}$, and its relationship with the correspondence between the two categories **FinSet** and $\mathbf{FinVec}_!$. We also analyzed how the "thunk" construction can be carried over $\mathbf{PCF}_f^{alg}$ to emulate call-by-name in call-by-value.

Summary of insights. There are two lessons that can be obtained from these results and discussions. First, there is a general understanding that, on one side, Kleisli categories and the "call-by-name" encoding of intuitionistic logic into linear logic really correspond to computational call-by-name, and, on the other side, that Eilenberg-Moore categories and the "call-by-value" encoding of intuitionistic logic into linear logic really correspond to

computational call-by-value. This intuition turns out to be fully verified in the case of finite vector spaces. There was, nonetheless, a subtlety in the sense that the Eilenberg-Moore category is not quite the right one: the correct category for call-by-value turned out to be the construction $\mathbf{FreeCoalg}^{+}$.

The second lesson one can take from this analysis is the conclusion of Section 6.1.1: the fact that the vectorial structure of **FinVec** "does not really count" from a computational perspective. Any set-morphism between vector spaces corresponds to an actual linear map, due to the fact that to any such map there exists a corresponding polynomial. Because of the correspondence between the models and the languages, this fact is also reflected at the language level (Section 6.1).

Further work. There are further connections and questions to explore in future research. One possibility is to develop a comonad as it is done in *quantitative* models of linear logic such as finiteness spaces [10]. Finiteness spaces are also based on vector spaces, and can handle both infinite and finite fields. However, their definition of comultiplication assumes that the space $!A$ has infinite dimension. By switching to yet a different variant of linear logic, namely *bounded* linear logic [14], one might obtain a good fit with finite dimensional spaces. This suggests that an interesting path to follow would be to explore finite vector spaces as a model of bounded linear logic and to compare it with the present paper's comonad.

References

[1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, Vol. 163, pp. 409–470, 2000. doi:10.1006/inco.2000.2930.

[2] P. Arrighi, A. Díaz-Caro, and B. Valiron. A type system for the vectorial aspects of the linear-algebraic lambda-calculus. In *Proceedings of the 7th International Workshop on Developments of Computational Methods (DCM 2011)*, Electronic Proceedings in Theoretical Computer Science, Vol. 88, pp. 1–15, 2012. doi:10.4204/EPTCS.88.1.

[3] P. Arrighi and A. Díaz Caro. A System F accounting for scalars. *Logic Methods in Computer Science*, Vol. 8, Paper 11, 2012. doi:10.2168/LMCS-8(1:11)2012.

[4] P. Arrighi and G. Dowek. Linear-algebraic λ-calculus: higher-order, encodings, and confluence. In *Proceedings of the 19th international conference on Rewriting Techniques and Applications (RTA '08)*, Lecture Notes in Computer Science, Vol. 5117, pp. 17–31, 2008. doi:10.1007/978-3-540-70590-1_2.

[5] N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Proceedings of the 8th Workshop on Computer Science Logic*, Lecture Notes in Computer Science, Vol. 933, pp 121–135, 1995. doi:10.1007/BFb0022251.

[6] G. Bierman. *On Intuitionistic Linear Logic*. PhD thesis, Cambridge Univ., 1994. Also Tech. Report UCAM-CL-TR-346.

[7] A. Bucciarelli, T. Ehrhard, and G. Manzonetto. A relational semantics for parallelism and non-determinism in a functional setting. *Ann. Annals of Pure and Applied Logic*, Vol. 163, pp. 918–934, 2012. doi:10.1016/j.apal.2011.09.008.

[8] P. de Groote. Strong normalization in a non-deterministic typed lambda-calculus. In *Proceedings of the Third International Symposium on Logical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 813, pp. 142–152, 1994. doi:10.1007/3-540-58140-5_15.

[9] A. Díaz-Caro. *Du Typage Vectoriel*. PhD thesis, Univ. de Grenoble, 2011. Archived online at tel.archives-ouvertes.fr:tel-00631514.

[10] T. Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, Vol. 15, pp. 615–646, 2005. doi:10.1017/S0960129504004645.

[11] T. Ehrhard, M. Pagani, and C. Tasson. The computational meaning of probabilistic coherence spaces. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS 2011)*, pp. 87–96, 2011. doi:10.1109/LICS.2011.29.

[12] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science*, Vol. 309, pp. 1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.

[13] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, Vol. 50, pp. 1–101, 1987. doi:10.1016/0304-3975(87)90045-4.

[14] J.-Y. Girard, A. Scedrov and P. J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theoretical Computer Science*, Vol. 97, pp. 1–65, 1992. doi:10.1016/0304-3975(92)90386-T.

[15] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proof and Types*. Cambridge Univ. Press, 1990. Available online at http://www.paultaylor.eu/stable/Proofs+Types.html.

[16] G. G. Hillebrand. *Finite Model Theory in the Simply Typed Lambda Calculus*. PhD thesis, Brown Univ., 1991. Also Tech. Report CS-94-24.

[17] M. Hyland and A. Schalk. Glueing and orthogonality for models of linear logic. *Theoretical Computer Science*, Vol. 294, pp. 183–231, 2003. doi:10.1016/0304-3975(87)90045-4.

[18] R. P. James, G. Ortiz, and A. Sabry. Quantum computing over finite fields. Draft, 2011. Found on arXiv:1101.3764.

[19] J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge Univ. Press, 1994.

[20] S. Lang. *Algebra*. Springer, 2005.

[21] R. Lidl. *Finite fields*, Cambridge Univ. Press, 1997.

[22] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.

[23] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, Vol. 4, pp. 1–22, 1977. doi:10.1016/0304-3975(77)90053-6.

[24] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, Vol. 5, pp. 223–255, 1977. doi:10.1016/0304-3975(77)90044-5.

[25] V. R. Pratt. Re: Linear logic semantics (barwise). On the TYPES mailing list, February 1992. http://www.seas.upenn.edu/~sweirich/types/archive/1992/msg00047.html.

[26] V. R. Pratt. Chu spaces: Complementarity and uncertainty in rational mechanics. Technical report, Stanford Univ., 1994. Course notes,

TEMPUS summer school, 35 pages. http://boole.stanford.edu/pub/bud.pdf.

[27] S. Salvati. Recognizability in the simply typed lambda-calculus. In *Proceedings of 16th International Workshop on Logic, Language, Information and Computation (WoLLIC 2009)*, Lecture Notes in Computer Science, Vol. 5514, pp. 48–60, 2009. doi:10.1007/978-3-642-02261-6_5.

[28] B. Schumacher and M. D. Westmoreland. Modal quantum theory. In *Proceedings of the 7th Workshop on Quantum Physics and Logic*, pp. 145–149, 2010. http://www.cs.ox.ac.uk/people/bob.coecke/QPL_proceedings.html.

[29] D. S. Scott. A type-theoretic alternative to CUCH, ISWIM, OWHY. *Theoretical Computer Science*, Vol. 121, pp. 411–440, 1993. doi:10.1016/0304-3975(93)90095-B.

[30] P. Selinger. Order-incompleteness and finite lambda reduction models. *Theoretical Computer Science*, Vol. 309, pp. 43–63, 2003. doi:10.1016/S0304-3975(02)00038-5.

[31] S. Soloviev. Category of finite sets and cartesian closed categories. *Journal of Soviet Mathematics*, Vol. 22, pp. 1387–1400, 1983. doi:10.1007/BF01084396.

[32] B. Valiron. A typed, algebraic, computational lambda-calculus. *Mathematical Structures in Computer Science*, Vol. 23, pp. 504–554, 2013. doi:10.1017/S0960129512000205.

[33] B. Valiron and S. Zdancewic. Finite vector spaces as model of simply-typed lambda-calculi. In *Proceedings of the 11th International Colloquium on Theoretical Aspects of Computing (ICTAC 2014)*, Lecture Notes in Computer Science, Vol. 8687, pp. 442–459, 2014. doi:10.1007/978-3-319-10882-7_26.

[34] L. Vaux. The algebraic lambda-calculus. *Mathematical Structures in Computer Science*, Vol. 19, pp. 1029–1059, 2009. doi:10.1017/S0960129509990089.

[35] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.