Finite Vector Spaces as Model of Simply-Typed Lambda-Calculi

Benoît Valiron[1] and Steve Zdancewic[2]

[1] PPS, UMR 7126, Univ Paris Diderot, Sorbonne Paris Cité, F-75205 Paris, France *
[2] University of Pennsylvania, Philadelphia, US**

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. The second model is not complete, but we develop an algebraic extension of the finite lambda calculus that recovers completeness. The relationship between the two semantics is described, and several examples based on Church numerals are presented.

1 Introduction

A standard way to study properties of functional programming languages is via denotational semantics. A denotational semantics (or model) for a language is a mathematical representation of its programs [32], 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 [27] – the laboratory mouse of functional programming languages. PCF comes as an extension of simply-typed lambda-calculus with a call-by-name reduction strategy, basic types and 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.

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 [21], cpos [22] or game semantics [1], while more recent works use quantitative semantics of linear logic [12] and discuss probabilistic extensions [10] or non-determinism [6].

As a category, a model for a PCF language is at least required to be cartesian closed to model 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 modality "!". Although the models that are usually considered are

* Partially supported by the ANR project ANR-2010-BLAN-021301 LOGOI.
** This work has been funded in part by NFS grant CCF-1421193.

rich and expressive [6, 9, 10], "degenerate" models nevertheless exist [15, 24]. 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 [23], but the computational behavior of the corresponding coKleisli category $\mathbf{FinVec}_!$ as a model of PCF 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 two models **FinSet** and $\mathbf{FinVec}_!$ fit 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.

Results. This paper presents the first account of the interpretation of two PCF-like languages in finite vector spaces. More specifically, we show 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 of the 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. We discuss the relationship between the two languages and show how to encode the extension within $\mathbf{PCF}_f$.

Related works. In the literature, finite models for lambda-calculi are commonly used. For example, Hillebrand analyzes databases as finite models of the simply-typed lambda calculus [14]. Salvati presents a model based on finite sets [25], while Selinger presents models based on finite posets [28]. Finally, Solovev [29] relate 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 [6], finiteness spaces [9], probabilistic coherent spaces [10], *etc.*

As models of linear logic, finite vector spaces are folklore [23] and appear as side examples of more general constructions such as Chu spaces [24] or glueing [15]. Computationally, Chu spaces (and then to some extent finite vector spaces) have been used in connection with automata [24]. Finally, recently finite vector spaces have also been used as a toy model for quantum computation (see e.g. [16, 26]).

Algebraic lambda-calculi, that is, lambda-calculi with a vectorial structure have been first defined in connection with finiteness spaces [11, 31]. Another approach [2, 3] comes to a similar type of language from quantum computation. The former approach is call-by-name while the latter is call-by-value. A general categorical semantics has been developed [30] but no other concrete models have been considered.

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 shows how to build a model of linear logic from the adjunction with finite sets. Section 4.4 discusses the corresponding coKleisli category as a model of $\mathbf{PCF}_f$ and presents some examples based on Church numerals. As $\mathbf{PCF}_f$ is not fully-abstract, Section 5 explains how to extend the language to better match the model. Finally, Section 6 discusses various related aspects: the relationship between $\mathbf{PCF}_f$ and its extension, other categories in play, and potential generalization of fields.

2 A Finite PCF-style Lambda Calculus

We pick a minimal finite PCF-style language with pairs and booleans. We call it $\mathbf{PCF}_f$: it is intrinsically typed (i.e. Church-style: all subterms are defined with their type) and defined as follows.

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

Values, including "lazy" pairs (that is, pairs of arbitrary terms, as opposed to pairs of values), are inductively defined by $U, V ::= x \mid \lambda x.M \mid \langle\, M, N\,\rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff}$. The terms consist of the regular lambda-terms, plus specific term constructs. The terms $\mathtt{tt}$ and $\mathtt{ff}$ respectively stand for the booleans True and False, while $\mathtt{if} - \mathtt{then} - \mathtt{else} -$ is the boolean test operator. The type $\mathtt{Bool}$ 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 functions.

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

$$\frac{}{\Delta, x : A \vdash x : A} \quad \frac{}{\Delta \vdash \star : \mathbf{1}} \quad \frac{}{\Delta \vdash \mathtt{tt}, \mathtt{ff} : \mathtt{Bool}} \quad \frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \to B} \quad \frac{\Delta \vdash M : A_l \times A_r}{\Delta \vdash \pi_i(M) : A_i}$$

$$\frac{\Delta \vdash M : A \to B \quad \Delta \vdash N : A}{\Delta \vdash MN : B} \quad \frac{\Delta \vdash M : A \quad \Delta \vdash N : B}{\Delta \vdash \langle M, N \rangle : A \times B} \quad \frac{\Delta \vdash M : \mathtt{Bool} \quad \Delta \vdash N_1, N_2 : A}{\Delta \vdash \mathtt{if}\, M\, \mathtt{then}\, N_1\, \mathtt{else}\, N_2 : A} \quad \frac{\Delta \vdash M : \mathbf{1} \quad \Delta \vdash N : A}{\Delta \vdash \mathtt{let}\, \star = M\, \mathtt{in}\, N : A}$$

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 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 $\to$.

Lemma 2. *(1) For any well-typed term $M : A$, either M is a value or M reduces to some term $N : A$. (2) The only closed value of type $\mathbf{1}$ is $\star$ and the only closed values of type* $\mathtt{Bool}$ *are* $\mathtt{tt}$ *and* $\mathtt{ff}$*. (3) The language $\mathbf{PCF}_f$ is strongly normalizing.*

Proof. The fact that the language $\mathbf{PCF}_f$ is strongly normalizing comes from the fact that it can be easily encoded in the strongly normalizing language system F [13]. □

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

$$(\lambda x.M)N \to M[N/x] \qquad \pi_l\langle M, N \rangle \to M \qquad \mathtt{if}\,\mathtt{tt}\,\mathtt{then}\, M\, \mathtt{else}\, N \to M$$

$$\mathtt{let}\, \star = \star\, \mathtt{in}\, M \to M \qquad \pi_r\langle M, N \rangle \to N \qquad \mathtt{if}\,\mathtt{ff}\,\mathtt{then}\, M\, \mathtt{else}\, N \to N$$

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

$$\frac{M \to M'}{\mathtt{if}\, M\, \mathtt{then}\, N_1\, \mathtt{else}\, N_2 \to \mathtt{if}\, M'\, \mathtt{then}\, N_1\, \mathtt{else}\, N_2} \qquad \frac{M \to M'}{\mathtt{let}\, \star = M\, \mathtt{in}\, N \to \mathtt{let}\, \star = M'\, \mathtt{in}\, N}$$

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

$$[\![\Delta, x:A \vdash x:A]\!]^{\mathrm{set}} : (d,a) \longmapsto a \qquad [\![\Delta \vdash \langle M,N \rangle : A\times B]\!]^{\mathrm{set}} \ :\ d \longmapsto \langle\, [\![M]\!]^{\mathrm{set}}(d), [\![N]\!]^{\mathrm{set}}(d)\,\rangle$$

$$[\![\Delta \vdash \mathtt{tt} : \mathtt{Bool}]\!]^{\mathrm{set}} : d \longmapsto \mathtt{tt} \qquad [\![\Delta \vdash MN : B]\!]^{\mathrm{set}} \ :\ d \longmapsto [\![M]\!]^{\mathrm{set}}(d)([\![N]\!]^{\mathrm{set}}(d))$$

$$[\![\Delta \vdash \mathtt{ff} : \mathtt{Bool}]\!]^{\mathrm{set}} : d \longmapsto \mathtt{ff} \qquad [\![\Delta \vdash \pi_l(M) : A]\!]^{\mathrm{set}} = [\![M]\!]^{\mathrm{set}};\pi_l$$

$$[\![\Delta \vdash \star : \mathbf{1}]\!]^{\mathrm{set}} : d \longmapsto \star \qquad [\![\Delta \vdash \pi_r(M) : B]\!]^{\mathrm{set}} = [\![M]\!]^{\mathrm{set}};\pi_r$$

$$[\![\Delta \vdash \lambda x.M : A \to B]\!]^{\mathrm{set}} \ = \ d \longmapsto (a \mapsto [\![M]\!]^{\mathrm{set}}(d,a)) \qquad [\![\Delta \vdash \mathtt{let}\,\star = M\,\mathtt{in}\,N : A]\!]^{\mathrm{set}} \ = \ [\![N]\!]^{\mathrm{set}}$$

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

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 $C[N] \to^* b$.

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 [17].

Lemma 3. *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 [29]: 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 our PCF language $\mathbf{PCF}_f$. The denotation of types corresponds to the implicit meaning of the types: $[\![\mathbf{1}]\!]^{\mathrm{set}} := \{\star\}$,

$[\![\texttt{Bool}]\!]^{\text{set}} := \{\, \texttt{tt}, \texttt{ff} \,\}$, the product is the set-product $[\![A \times B]\!]^{\text{set}} := [\![A]\!]^{\text{set}} \times [\![B]\!]^{\text{set}}$, while the arrow is the set of morphisms: $[\![A \to B]\!]^{\text{set}} := \mathbf{FinSet}([\![A]\!]^{\text{set}}, [\![B]\!]^{\text{set}})$. The set $\{\texttt{tt}, \texttt{ff}\}$ is also written $\texttt{Bool}$. Similarly, the set $\{\star\}$ is also written $\mathbf{1}$. The denotation of a typing judgment $x_1 : A_1, \dots x_n : A_n \vdash M : B$ is a morphism $[\![A_1]\!]^{\text{set}} \times \dots \times [\![A_n]\!]^{\text{set}} \to [\![B]\!]^{\text{set}}$, and is inductively defined as in Table 3. The variable d is assumed to be an element of $[\![\Delta]\!]^{\text{set}}$, while a and b are elements of $[\![A]\!]^{\text{set}}$ and $[\![B]\!]^{\text{set}}$ respectively.

This denotation is sound with respect to the operational equivalence.

Lemma 4. *If $M \simeq_{\text{ax}} N : A$ then $[\![M]\!]^{\text{set}} = [\![N]\!]^{\text{set}}$.* □

Theorem 5. *The model is sound with respect to the operational equivalence: Suppose that $\Delta \vdash M, N : A$. If $[\![M]\!]^{\text{set}} = [\![N]\!]^{\text{set}}$ then $M \simeq_{\text{op}} N$.*

Proof. Suppose that $M \not\simeq_{\text{op}} N$ and let Δ be $\{x_i : A_i\}_i$. Then, because of Lemma 2, there exists a context $C[-]$ such that $C[M] \to^* \texttt{tt}$ and $C[N] \to^* \texttt{ff}$. It follows that $(\lambda z.C[z\, x_1 \dots x_n])(\lambda x_1 \dots x_n.M) \simeq_{\text{ax}} \texttt{tt}$ and $(\lambda z.C[z\, x_1 \dots x_n])(\lambda x_1 \dots x_n.N) \simeq_{\text{ax}} \texttt{ff}$. 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 $(\lambda x_1 \dots x_n.N)$. Lemmas 3 and 4 yield a contradiction. □

FinSet and the language $\mathbf{PCF}_f$ are somehow two sides of the same coin. Theorems 6 and 7 formalize this correspondence.

Theorem 6 (Full completeness)**.** *For every morphism $f : [\![A]\!]^{\text{set}} \to [\![B]\!]^{\text{set}}$ there exists a valid judgment $x : A \vdash M : B$ such that $f = [\![M]\!]^{\text{set}}$.*

Proof. We start by defining inductively on A two families of terms $M_a : A$ and $\delta_a : A \to \texttt{Bool}$ indexed by $a \in [\![A]\!]^{\text{set}}$, such that $[\![M_a]\!]^{\text{set}} = a$ and $[\![\delta_a]\!]^{\text{set}}$ sends a to $\texttt{tt}$ and all other elements to $\texttt{ff}$. For the types $\mathbf{1}$ and $\texttt{Bool}$, the terms $M_\star$, $M_{\texttt{tt}}$ and $M_{\texttt{ff}}$ are the corresponding constants. The term $\delta_\star$ is $\lambda x.\star$, $\delta_{\texttt{tt}}$ is $\lambda x.x$ while $\delta_{\texttt{ff}}$ 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 $[\![A]\!]^{\text{set}}$ is finite: if $g \in [\![A \to B]\!]^{\text{set}}$, 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 7 (Equivalence)**.** *Suppose that $\Delta \vdash M, N : A$. Then $[\![M]\!]^{\text{set}} = [\![N]\!]^{\text{set}}$ if and only if $M \simeq_{\text{op}} N$.*

Proof. The left-to-right implication is Theorem 5. We prove the right-to-left implication by contrapositive. Assume that $[\![M]\!]^{\text{set}} \neq [\![N]\!]^{\text{set}}$. Then there exists a function $f : \mathbf{1} \to [\![A]\!]^{\text{set}}$ and a function $g : [\![B]\!]^{\text{set}} \to [\![\texttt{Bool}]\!]^{\text{set}}$ such that the boolean $f; [\![M]\!]^{\text{set}}; g$ is different from $f; [\![N]\!]^{\text{set}}; g$. By Theorem 6, 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 $M \not\simeq_{\text{op}} N$. □

Corollary 8. *Since it is fully complete, the semantics* **FinSet** *is also adequate and fully abstract with respect to $\mathbf{PCF}_f$.* □

Example 9. 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 7, one can conclude that all Church numerals $\lambda fx.f(f(\cdots(fx)\cdots))$ 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 10. How many operationally distinct Church numerals based over `Bool` are there ? From Theorem 7, it is enough to count how many distinct denotations of Church numerals there are in $[\![(\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})]\!]^{\text{set}}$. There are exactly 4 distinct maps $\texttt{Bool} \to \texttt{Bool}$. Written as pairs (x, y) when $f(\texttt{tt}) = x$ and $f(\texttt{ff}) = y$, the maps tt, tf, ft and ff are respectively $(\texttt{tt}, \texttt{tt})$, $(\texttt{tt}, \texttt{ff})$, $(\texttt{ff}, \texttt{tt})$ and $(\texttt{ff}, \texttt{ff})$.

Then, if the Church numeral $\bar{n}$ is written as a tuple $(\bar{n}(tt), \bar{n}(tf), \bar{n}(ft), \bar{n}(ff))$, 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 $n \geq 1$, $[\![\bar{n}]\!]^{\text{set}} = [\![\overline{n+2}]\!]^{\text{set}}$. 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* [19] 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* [18] 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 of Table 6 (taken unordered). 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 $\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$ and is a finite vector space. It is also the vector space freely generated from the 2-elements set $\{\texttt{tt}, \texttt{ff}\}$: each vectors respectively corresponds to $0, \texttt{tt}, \texttt{ff}$, and $\texttt{tt} + \texttt{ff}$.

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 [20] that the category of finite sets and functions and the category of finite vector spaces and linear maps form an adjunction

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

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 [23]. Indeed, the adjunction is symmetric monoidal with the following two natural transformations:

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

This makes a *linear-non-linear category* [4], equivalent to a linear category, and is a model of intuitionistic linear logic [5].

4.3 Model of Linear Logic

The adjunction in Eq. (1) 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

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

$$\begin{array}{rr} [\![\Delta, x : A \vdash x : A]\!]^{\text{vec}} : d \otimes b_a \longmapsto a & [\![\Delta \vdash \langle M, N\rangle : A \times B]\!]^{\text{vec}} \ : \ d \longmapsto [\![M]\!]^{\text{vec}}(d) \otimes [\![N]\!]^{\text{vec}}(d) \\ [\![\Delta \vdash \texttt{tt} : \texttt{Bool}]\!]^{\text{vec}} : d \longmapsto \texttt{tt} & [\![\Delta \vdash MN : B]\!]^{\text{vec}} \ : \ d \longmapsto [\![M]\!]^{\text{vec}}(d)([\![N]\!]^{\text{vec}}(d)) \\ [\![\Delta \vdash \texttt{ff} : \texttt{Bool}]\!]^{\text{vec}} : d \longmapsto \texttt{ff} & [\![\Delta \vdash \pi_l(M) : A]\!]^{\text{vec}} = [\![M]\!]^{\text{vec}}; \pi_l \\ [\![\Delta \vdash \star : \mathbf{1}]\!]^{\text{vec}} : d \longmapsto \star & [\![\Delta \vdash \pi_r(M) : B]\!]^{\text{vec}} = [\![M]\!]^{\text{vec}}; \pi_r \end{array}$$

$$[\![\Delta \vdash \lambda x.M : A \to B]\!]^{\text{vec}} = d \longmapsto (b_a \mapsto [\![M]\!]^{\text{vec}}(d \otimes b_a))$$

$$[\![\Delta \vdash \texttt{let}\,\star = M\,\texttt{in}\,N : A]\!]^{\text{vec}} = d \longmapsto \alpha \cdot [\![N]\!]^{\text{vec}}(d) \qquad \text{where} \quad [\![M]\!]^{\text{vec}}(d) = \alpha \cdot \star.$$

$$[\![\Delta \vdash \texttt{if}\,M\,\texttt{then}\,N\,\texttt{else}\,P : A]\!]^{\text{vec}} = d \longmapsto \alpha \cdot [\![N]\!]^{\text{vec}}(d) + \beta \cdot [\![P]\!]^{\text{set}}(d)$$
$$\text{where} \quad [\![\Delta \vdash M : \texttt{Bool}]\!]^{\text{vec}}(d) = \alpha \cdot \texttt{tt} + \beta \cdot \texttt{ff}.$$

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.

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

There is a canonical full embedding E of categories sending **FinVec**$_!$ on **FinSet**. It sends an object U to the set of vectors of U (i.e. it acts as the forgetful functor on objects) and sends the linear map $f : !U \to V$ to the map $v \mapsto f(b_v)$.

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

Remark 11. The construction proposed as side example by Hyland and Schalk [15] considers finite vector spaces with a field of characteristic 2. There, the modality is built using the exterior product algebra, and it turns out 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.

Remark 12. Quantitative models of linear logic such as finiteness spaces [9] are also based on vector spaces; however, in these cases the procedure to build a comonad does not play well with the finite dimension the vector spaces considered in this paper: the definition of the comultiplication assumes that the space $!A$ is infinitely dimensional.

4.4 Finite Vector Spaces as a Model

Since **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 $[\![\mathtt{Bool}]\!]^{\mathrm{vec}} := \{\, \sum_i \alpha_i \cdot \mathtt{tt} + \beta_i \cdot \mathtt{ff} \mid \alpha_i, \beta_i \in K\,\}$. The product is the usual product space: $[\![A \times B]\!]^{\mathrm{vec}} := [\![A]\!]^{\mathrm{vec}} \times [\![B]\!]^{\mathrm{vec}}$, whereas the arrow type is $[\![A \to B]\!]^{\mathrm{vec}} :=$ **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 13. 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 $[\![\mathbf{1}]\!]^{\text{vec}} = \mathbf{1} = \{0 \cdot \star, 1 \cdot \star\} = \{0, \star\}$. 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 $[\![\mathbf{1} \to \mathbf{1}]\!]^{\text{vec}}$ 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 $!(!\mathbf{1} \multimap \mathbf{1}) \multimap (!\mathbf{1} \multimap \mathbf{1})$ can be represented by 2×4 matrices with coefficients in $\mathbb{F}_2$. The basis elements b_v are ordered as above, 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 $!\mathbf{1}$.

$$\begin{array}{cccc|l} \cdot & \cdot & \cdot & \cdot & f_0 \\ \cdot & \cdot & \cdot & \cdot & f_\star \\ b_0 & b_{f_0} & b_{f_\star} & b_{f_0+f_\star} & \end{array}$$

Note that this characterization is similar to the **FinSet** Example 10, except that there, the type over which the Church numerals were built was `Bool`. Over $\mathbf{1}$, Example 9 stated that all Church numerals collapse.

Example 14. 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, $[\![\mathbf{1}]\!]^{\text{vec}} = \{0, \star, 2 \cdot \star, \ldots, (p-1) \cdot \star\}$, 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.

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

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

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

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

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

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

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

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

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

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 15. Let us briefly reprise Example 10 in the context of **FinVec**$_!$. Even with a field of characteristic 2, the vector space $[\![(\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})]\!]^{\mathrm{vec}}$ is relatively large: $\texttt{Bool}$ has dimension 2 and consists of 4 vectors, $!\texttt{Bool}$ 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 then a matrix of size 256×8.

Let us represent a function $!\texttt{Bool} \multimap \texttt{Bool}$ as a tuple $(\mathtt{x}_{ij}^k)_{i,j,k}$ lexicographically ordered $\mathtt{x}_{00}^0, \mathtt{x}_{00}^1, \mathtt{x}_{01}^0, \mathtt{x}_{01}^1, \mathtt{x}_{10}^0, \mathtt{x}_{10}^1, \mathtt{x}_{11}^0, \mathtt{x}_{11}^1$, representing the map sending $b_{i\cdot\mathtt{tt}+j\cdot\mathtt{ff}}$ to $\mathtt{x}_{ij}^0 \cdot \mathtt{tt} + \mathtt{x}_{ij}^1 \cdot \mathtt{ff}$. 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 Church 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 **FinVec**$_!$, over the type $\texttt{Bool}$, 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 $i = 3, 4, \ldots 14$.

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 13, 14 and 15), the model fails to be fully abstract.

Lemma 16. *If* $M \simeq_{\mathrm{ax}} N : A$ *then* $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$. □

Theorem 17. *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 5 and proceeds by contrapositive, using Lemmas 2, 2, 3 and 16. □

Theorem 18 (Adequacy)**.** *Given two closed terms* M *and* N *of type* $\mathtt{Bool}$*,* $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$ *if and only if* $M \simeq_{\mathrm{op}} N$.

Proof. The left-to-right direction is Theorem 17. For the right-to-left direction, since the terms M and N are closed of type $\mathtt{Bool}$, one can choose the context $C[-]$ to be $[-]$, and we have $M \to^* b$ if and only if $N \to^* b$. From Lemma 2, there exists such a boolean b: we deduce from Lemma 3 that $M \simeq_{\mathrm{ax}} N$. We conclude with Lemma 16. □

Remark 19. The model **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 $![\![\mathtt{Bool}]\!]^{\mathrm{vec}} \to [\![\mathtt{Bool}]\!]^{\mathrm{vec}}$. 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 $v \in ![\![\mathtt{Bool}]\!]^{\mathrm{vec}}$ 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 $[\![x : \mathtt{Bool} \vdash x : \mathtt{Bool}]\!]^{\mathrm{vec}}(v) = \nu$, the denotation of the second term is the function sending v to $\nu \cdot [\![x : \mathtt{Bool} \vdash \mathtt{tt} : \mathtt{Bool}]\!]^{\mathrm{vec}}(v)$, 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 20. 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 $[\![A]\!]^{\mathrm{vec}}$. 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, $[\![T_{\mathtt{tt},\mathtt{tt}}]\!]^{\mathrm{vec}} = [\![T_{\mathtt{ff},\mathtt{ff}}]\!]^{\mathrm{vec}} + [\![T_{\mathtt{tt},\mathtt{ff}}]\!]^{\mathrm{vec}} + [\![T_{\mathtt{ff},\mathtt{tt}}]\!]^{\mathrm{vec}}$.

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

5 An Algebraic Lambda Calculus

To solve the problem, we extend the language $\mathbf{PCF}_f$ by adding an algebraic structure to mimic the notion of linear distribution existing in **FinVec**$_!$. The extended language

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

$$\begin{array}{cccc} \alpha \cdot M + \beta \cdot M \to (\alpha+\beta)\cdot M & M+N \to N+M & & (M+N)+P \to M+(N+P) \\ \alpha \cdot M + M \to (\alpha+1)\cdot M & 0 \cdot M \to 0 & 1 \cdot M \to M & \alpha\cdot(M+N) \to \alpha\cdot M + \alpha \cdot N \\ M+M \to (1+1)\cdot M & \alpha \cdot 0 \to 0 & 0+M \to M & \alpha\cdot(\beta\cdot M) \to (\alpha\beta)\cdot M \end{array}$$

$\mathbf{PCF}_f^{alg}$ is a call-by-name variation of [2, 3] and 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 values are now $U,V ::= x \mid \lambda x.M \mid \langle\, M,N \,\rangle \mid \star \mid \mathtt{tt} \mid \mathtt{ff} \mid 0 \mid U+V \mid \alpha\cdot U$.. The typing rules are the same for the regular constructs. The new constructs are typed as follows: for all A, $\Delta \vdash 0 : A$, and provided that $\Delta \vdash M,N : A$, then $\Delta \vdash M+N : A$ and $\Delta \vdash \alpha\cdot M : A$. The rewrite rules are extended as follows.

1) A set of algebraic rewrite rules shown in Table 6. We shall explicitly talk about *algebraic rewrite rules* when referring to these extended 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 $\alpha\cdot(M + P) + N$. The reduction rules from Γ will be called *non-algebraic*.

2) The relation 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) Two congruence rules. If $M \to M'$, then $M+N \to M'+N$ and $\alpha\cdot M \to \alpha\cdot M'$.

Remark 21. 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 [3]. 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 $\langle\, \mathtt{tt},\mathtt{tt} \,\rangle + \langle\, \mathtt{ff},\mathtt{ff} \,\rangle + \langle\, \mathtt{tt},\mathtt{ff} \,\rangle + \langle\, \mathtt{ff},\mathtt{tt} \,\rangle$. We'll come back to this distinction in Section 6.3.

The algebraic extension preserves the safety properties, the characterization of values and the strong normalization. Associativity and commutativity induce a subtlety.

Lemma 22. *The algebraic fragment of* $\mathbf{PCF}_f^{alg}$ *is strongly normalizing modulo AC.*

Proof. The proof can be done as in [3], using the same measure on terms that decreases with algebraic rewrites. 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 23 (Safety properties mod AC)**.** *A well-typed term $M : A$ is a value or, if not, reduces to some $N : A$ via a sequence of steps among which one is* not *algebraic.* □

Lemma 24. *Any value of type $\mathbf{1}$ has AC-normal form 0, $\star$ or $\alpha \cdot \star$, with $\alpha \neq 0, 1$.* □

Lemma 25. *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 $\lambda_{-\mathbf{w}\mathrm{LK}^{\rightarrow}}$ of [7], 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 23, 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 22, saying there is always a finite number of these eliminated algebraic rewrites. □

5.1 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 $\mathtt{Bool}$, we shall use contexts of type $\mathbf{1}$: thanks to Lemma 24, 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 $M \simeq_{\mathrm{op}} N$, if for all closed contexts $C[-]$ of type $\mathbf{1}$ where the hole binds Δ, for all b normal forms of type $\mathbf{1}$, $C[M] \to^* b$ if and only if $C[N] \to^* b$.

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

5.3 Finite Vector Spaces as a Model

The category **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 27. With the extended term constructs, the language $\mathbf{PCF}_f^{alg}$ does not share the drawbacks of $\mathbf{PCF}_f$ emphasized in Remark 19. 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 $C[-] = (\lambda x.[-])0$, 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 28. *If* $M \simeq_{\rm ax} N : A$ *in* $\mathbf{PCF}_f^{alg}$ *then* $[\![M]\!]^{\rm vec} = [\![N]\!]^{\rm vec}$. □

Theorem 29. *Let* $\Delta \vdash M, N : A$ *be two valid typing judgments in* $\mathbf{PCF}_f^{alg}$*. If* $[\![M]\!]^{\rm vec} = [\![N]\!]^{\rm vec}$ *then we also have* $M \simeq_{\rm op} N$.

Proof. The proof is similar to the proof of Theorem 5: Assume $M \not\simeq_{\rm op} N$. Then there exists a context $C[-]$ that distinguishes them. The call-by-name reduction preserves the type from Lemma 23, and $C[M]$ and $C[N]$ can be rewritten as the terms $(\lambda y.C[y\, x_1 \ldots x_n])\lambda x_1 \ldots x_n.M$ and $(\lambda y.C[y\, x_1 \ldots x_n])\lambda x_1 \ldots x_n.N$, and these are axiomatically equivalent to distinct normal forms, from Lemmas 25 and 26. We conclude from Lemmas 26 and 28 that the denotations of M and N are distinct. □

5.4 Two Auxiliary Constructs

Full completeness requires some machinery. It is obtained by showing that for every type A, for every vector v in $[\![A]\!]^{\rm vec}$, there are two terms $M_v^A : A$ and $\delta_v^A : A \to \mathbf{1}$ such that $[\![M_v^A]\!]^{\rm vec} = v$ and $[\![\delta_v^A]\!]^{\rm vec}$ 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 $[\![\mathtt{exp}^i(\alpha\cdot\star)]\!]^{\rm vec} = \alpha^i \cdot \star$. 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 $[\![\mathtt{izero}(\alpha\cdot\star)]\!]^{\rm vec} = 0$ 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 $M_{\alpha\cdot\star}^{\mathbf{1}}$ is simply $\alpha\cdot\star$. The term $\delta_{\alpha\cdot\star}^{\mathbf{1}}$ is $\lambda x.\mathtt{iszero}(x - \alpha\cdot\star)$.

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

At type $A = B \times C$. If $v \in [\![A]\!]^{\rm vec} = [\![B]\!]^{\rm vec} \times [\![C]\!]^{\rm vec}$, then $v = \langle u, w \rangle$, with $u \in [\![B]\!]^{\rm vec}$ and $w \in [\![C]\!]^{\rm vec}$. 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 $f \in [\![A]\!]^{\rm vec} = ![\![B]\!]^{\rm vec} \multimap [\![C]\!]^{\rm vec}$. 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 $\delta_{u_i}^B$ and $M_{w_i}^C$, one can define $M_v^{B\to C}$ as the term $\sum_i \lambda x.\mathtt{let}\,\star = \delta_{u_i}^B\,x\,\mathtt{in}\,M_{w_i}^C$. Similarly, one can construct $\delta_{w_i}^C$ and $M_{u_i}^B$, and from the construction in the previous paragraph we can also generate $\delta_{\langle w_1,\ldots w_n \rangle}^{C^{\times n}} : C^{\times n} \to \mathtt{Bool}$. The term $\delta_v^{B\to C}$ is then defined as $\lambda f.\delta_{\langle w_1,\ldots w_n \rangle}^{C^{\times n}} \langle f\, M_{u_1}^B, \ldots, f\, M_{u_1}^B \rangle$.

5.5 Full Completeness

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

Theorem 30 (Full completeness)**.** *For any type* A*, any vector* v *of* $[\![A]\!]^{\rm vec}$ *in* $\mathbf{FinVec}_!$ *is representable in the language* $\mathbf{PCF}_f^{alg}$. □

Theorem 31. *For all M and N, $M \simeq_{\mathrm{op}} N$ if and only if $[\![M]\!]^{\mathrm{vec}} = [\![N]\!]^{\mathrm{vec}}$.* $\square$

A corollary of the full completeness is that the semantics **FinVec** is also adequate and fully abstract with respect to $\mathbf{PCF}_f^{alg}$.

6 Discussion

6.1 Simulating the Vectorial Structure

As we already saw, there is a full embedding of category $E : \mathbf{FinVec}_! \hookrightarrow \mathbf{FinSet}$. This embedding can be understood as "mostly" saying that the vectorial structure "does not count" in $\mathbf{FinVec}_!$, as one can simulate it with finite sets. Because of Theorems 7 and 31, 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, it can be 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 32. The *vec-to-set* encoding of a type A, written $\mathrm{VtoS}A$, is defined inductively as follows: $\mathrm{VtoS}(\mathbf{1}) = \mathtt{Bool}$, $\mathrm{VtoS}(\mathtt{Bool}) = \mathtt{Bool} \times \mathtt{Bool}$, $\mathrm{VtoS}(A \times B) = \mathrm{VtoS}(A) \times \mathrm{VtoS}(B)$, and $\mathrm{VtoS}(A \to B) = \mathrm{VtoS}(A) \to \mathrm{VtoS}(B)$.

Theorem 33. *There are two typing judgments $x : A \vdash \phi_A^{\mathrm{vec}} : \mathrm{VtoS}(A)$ and $x : \mathrm{VtoS}(A) \vdash \bar{\phi}_A^{\mathrm{vec}} : A$, inverse of each other, in $\mathbf{PCF}_f^{alg}$ such that any typing judgment $x : A \vdash M : B$ can be factored into $A \xrightarrow{\phi_A^{\mathrm{vec}}} \mathrm{VtoS}(A) \xrightarrow{\tilde{M}} \mathrm{VtoS}(B) \xrightarrow{\bar{\phi}_B^{\mathrm{vec}}} B$, where $\tilde{M}$ is a regular lambda-term of $\mathbf{PCF}_f$.*

Proof. The two terms ϕ_A^{vec} and $\bar{\phi}_A^{\mathrm{vec}}$ are defined inductively on A. For the definition of $\phi_{\mathtt{Bool}}^{\mathrm{vec}}$ we are reusing the term δ_v of Section 5.4. The definition is in Table 7 $\square$

6.2 Categorical 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. Because of Theorem 7, 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

Table 7. Relation between $\mathbf{PCF}_f$ and $\mathbf{PCF}_f^{alg}$

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

$$\phi_{\mathtt{Bool}}^{\mathrm{vec}} = \mathtt{let}\,\star = \delta_0 x\,\mathtt{in}\,\langle\,\mathtt{tt},\mathtt{tt}\,\rangle + \mathtt{let}\,\star = \delta_{\mathtt{tt}} x\,\mathtt{in}\,\langle\,\mathtt{tt},\mathtt{ff}\,\rangle + \mathtt{let}\,\star = \delta_{\mathtt{ff}} x\,\mathtt{in}\,\langle\,\mathtt{ff},\mathtt{tt}\,\rangle + \mathtt{let}\,\star = \delta_{\mathtt{tt}+\mathtt{ff}} x\,\mathtt{in}\,\langle\,\mathtt{ff},\mathtt{ff}\,\rangle$$

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

$$\phi_{B\times C}^{\mathrm{vec}} = \langle\, x;\pi_l;\phi_B^{\mathrm{vec}}, x;\pi_r;\phi_C^{\mathrm{vec}}\,\rangle, \qquad \phi_{B\to C}^{\mathrm{vec}} = \lambda y.x(y;\bar{\phi}_B^{\mathrm{vec}});\phi_C^{\mathrm{vec}},$$

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

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 31, 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: it make a terminal object for the syntactic category.

Finally, Theorem 33 is essentially giving us a functor $\mathbf{PCF}_f^{alg} \to \mathbf{PCF}_f$ corresponding to the full embedding E. 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.

6.3 (Co)Eilenberg-Moore Category and Call-by-value

From a linear category with modality ! there are two canonical cartesian closed categories: the coKleisli category, but also the (co)Eilenberg-Moore category: here, objects are still those of **FinVec**, but morphisms are now $!A \to !B$.

According to [30], such a model would correspond to the call-by-value (or, as coined by [8] *call-by-base*) strategy for the algebraic structure discussed in Remark 21.

6.4 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 recover sets and relations. However, we heavily rely on the fact that we have a finite field K in the construction of Section 5.4, yielding the completeness result in Theorem 30.

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.

References

1. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. and Comp.*, 163:409–470, 2000.
2. P. Arrighi, A. Díaz-Caro, and B. Valiron. A type system for the vectorial aspects of the linear-algebraic lambda-calculus. In *Proc. of DCM*, 2011.
3. P. Arrighi and G. Dowek. Linear-algebraic λ-calculus. In *Proc. of RTA*, pages 17–31, 2008.
4. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. Technical report, Cambridge U., 1994.
5. G. Bierman. *On Intuitionistic Linear Logic*. PhD thesis, Cambridge U., 1993.

6. A. Bucciarelli, T. Ehrhard, and G. Manzonetto. A relational semantics for parallelism and non-determinism in a functional setting. *A. of Pure and App. Logic*, 163:918–934, 2012.
7. P. de Groote. Strong normalization in a non-deterministic typed lambda-calculus. In *Logical Foundations of Computer Science*, volume 813, pages 142–152, 1994.
8. A. Díaz-Caro. *Du Typage Vectoriel*. PhD thesis, U. de Grenoble, 2011.
9. T. Ehrhard. Finiteness spaces. *Math. Str. Comp. Sc.*, 15:615–646, 2005.
10. T. Ehrhard, M. Pagani, and C. Tasson. The computational meaning of probabilistic coherence spaces. In *Proc. of LICS*, 2011.
11. T. Ehrhard and L. Regnier. The differential lambda-calculus. *Th. Comp. Sc.*, 309:1–41, 2003.
12. J.-Y. Girard. Linear logic. *Th. Comp. Sc.*, 50:1–101, 1987.
13. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proof and Types*. CUP, 1990.
14. G. G. Hillebrand. *Finite Model Theory in the Simply Typed Lambda Calculus*. PhD thesis, Brown University, 1991.
15. M. Hyland and A. Schalk. Glueing and orthogonality for models of linear logic. *Th. Comp. Sc.*, 294:183–231, 2003.
16. R. P. James, G. Ortiz, and A. Sabry. Quantum computing over finite fields. Draft, 2011.
17. J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*. CUP, 1994.
18. S. Lang. *Algebra*. Springer, 2005.
19. R. Lidl. *Finite fields*, volume 20. CUP, 1997.
20. S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.
21. R. Milner. Fully abstract models of typed lambda-calculi. *Th. Comp. Sc.*, 4:1–22, 1977.
22. G. Plotkin. LCF considered as a programming language. *Th. Comp. Sc.*, 5:223–255, 1977.
23. V. R. Pratt. Re: Linear logic semantics (barwise). On the TYPES list, February 1992. `http://www.seas.upenn.edu/∼sweirich/types/archive/1992/msg00047.html`.
24. V. R. Pratt. Chu spaces: Complementarity and uncertainty in rational mechanics. Technical report, Stanford U., 1994.
25. S. Salvati. Recognizability in the simply typed lambda-calculus. In *Logic, Language, Information and Computation*, pages 48–60. 2009.
26. B. Schumacher and M. D. Westmoreland. Modal quantum theory. In *Proc. of QPL*, 2010.
27. D. S. Scott. A type-theoretic alternative to CUCH, ISWIM, OWHY. *Th. Comp. Sc.*, 121:411–440, 1993.
28. P. Selinger. Order-incompleteness and finite lambda reduction models. *Th. Comp. Sc.*, 309:43–63, 2003.
29. S. Soloviev. Category of finite sets and cartesian closed categories. *J. of Soviet Math.*, 22(3), 1983.
30. B. Valiron. A typed, algebraic, computational lambda-calculus. *Math. Str. Comp. Sc.*, 23:504–554, 2013.
31. L. Vaux. The algebraic lambda-calculus. *Math. Str. Comp. Sc.*, 19:1029–1059, 2009.
32. G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.