A categorical construction for the computational definition of vector spaces

Alejandro Díaz-Caro[1,2] and Octavio Malherbe[3]

[1]Instituto de Ciencias de la Computación (CONICET-Universidad de Buenos Aires), Buenos Aires, Argentina

[2]Universidad Nacional de Quilmes, Bernal, BA, Argentina

[3]Universidad de la República, Montevideo, Uruguay

This is an **extended abstract** of a full paper submitted to a journal for review. The full draft can be found at

http://arxiv.org/abs/1905.01305

Algebraic lambda calculi aim to embed to the lambda calculus, the notion of vector spaces over programs. This way a linear combination $\alpha.v + \beta.w$ of programs v and w, for some scalars α and β, is also a program [2]. This kind of construction has two independent origins. The *Algebraic Lambda Calculus* (ALC for short) [8] has been introduced as a fragment of the *Differential Lambda Calculus* [6], which is itself originated from Linear Logic [7]. ALC can be seen as the Differential Lambda Calculus without a differential operator. In the ALC the notion of vector spaces is embedded in the calculus with an equational theory, so the axioms of vector spaces, such as $\alpha.v + \beta.v = (\alpha + \beta).v$ are seen as equalities between programs. On the other hand, the *Linear Algebraic Lambda Calculus* (Lineal for short) [1] was meant for quantum computation. The aim of Lineal is to provide a computational definition of vector space and bilinear functions, and so, it defines the axioms of vector spaces as rewrite rules, providing a confluent calculus (so obtaining a canonical representation of vectors). This way, an equality such as $-v + v + 3.w - 2.w = w$ is described computationally step by step as

$$\begin{aligned}
&(-1).v + v + 3.w + (-2).w \\
&\longrightarrow 0.v + 3.w + (-2).w \\
&\longrightarrow 0.v + 1.w \\
&\longrightarrow \mathbf{0} + 1.w \\
&\longrightarrow 1.w \\
&\longrightarrow w
\end{aligned}$$

Rules like $\alpha.v + \beta.v \longrightarrow (\alpha + \beta).v$ say that these expressions are not *the same* but one reduces to the other, and so, a computational step has been performed. The backbone of this computation can be described as having an element $\alpha.v + \beta.v$ without properties, which is decomposed into its constituents parts α, β, and v, and reconstructed in another way. Otherwise, if we consider $\alpha.v + \beta.v$ being just a vector, as in

Alejandro Díaz-Caro: adiazcaro@icc.fcen.uba.ar

Octavio Malherbe: malherbe@fing.edu.uy

the ALC, then it would be equal to $(\alpha+\beta).v$ and the computation needed to arrive from the former to the latter would be ignored. The main idea in the present paper is to study the construction of Lineal from a categorical point of view, with an adjunction between a Cartesian closed category, which will threat the elements as not having properties, and an additive symmetric monoidal closed category, where the underlying properties will allow to do the needed algebraic manipulation. A concrete example is an adjunction between the category **Set** of sets and the category **Vec** of vector spaces. This way, a functor from **Set** to **Vec** will allow to do the needed manipulation, while a forgetful functor from **Vec** to **Set** will return the result of the computation.

The calculus Lambda-$\mathcal{S}^*$ [4] is a first-order typed fragment of Lineal. The type system has been designed as a quantum lambda calculus, where the main goal was to study the non-cloning restrictions. In quantum computing a known vector, such as a basis vector from the base considered for the measurements, can be duplicated freely (normally the duplication process is just a preparation of a new qubit in the same known basis state), while an unknown vector cannot. For this reason, a linear-logic like type system has been put in place. In linear logic we would write A the types of terms that cannot be duplicated while $!A$ types duplicable terms. In Lambda-$\mathcal{S}^*$ instead A are the types of the terms that represent basis vectors, while $S(A)$ are linear combinations of those (the span of A). Hence, A means that we can duplicate, while $S(A)$ means that we cannot duplicate. So the S is not the same as the bang "!", but somehow the opposite. This can be explained by the fact that linear logic is focused on the possibility of duplication, while here we focus on the possibility of superposition, which implies the impossibility of duplication.

In [4] a first denotational semantics (in environment style) is given where the type $\mathbb{B}$ is interpreted as $\{|0\rangle, |1\rangle\}$ while $S(\mathbb{B})$ is interpreted as $\mathsf{Span}(\{|0\rangle, |1\rangle\}) = \mathbb{C}^2$, and, in general, a type A is interpreted as a basis while $S(A)$ is the vector space generated by such a basis. In [5] we went further and gave a preliminary concrete categorical interpretation of Lambda-$\mathcal{S}$[1] where S is a functor of an adjunction between the category **Set** and the category **Vec**. Explicitly, when we evaluate S we obtain formal finite linear combinations of elements of a set with complex numbers as coefficients and the other functor of the adjunction, U, allows us to forget the vectorial structure. In this paper, we define the abstract categorical semantics of Lambda-$\mathcal{S}^*$, so we focus on the computational definition of vector spaces.

The main structural feature of our model is that it is expressive enough to describe the bridge between the propertyless elements such as $\alpha.v+\beta.v$, without any equational theory, and the result of its alge-

[1] Lambda-$\mathcal{S}$ is the same as Lambda-$\mathcal{S}^*$, but extended with a measurement operator.

braic manipulation into $(\alpha + \beta).v$, explicitly controlling its interaction. In the literature, intuitionistic linear (as in linear-logic) models are obtained by a monoidal comonad determined by a monoidal adjunction $(S, m) \dashv (U, n)$, i.e. the bang ! is interpreted by the comonad SU (see [3]). In a different way, a crucial ingredient of our model is to consider the monad US for the interpretation of S determined by a similar monoidal adjunction. This implies that on the one hand we have tight control of the Cartesian structure of the model (i.e. duplication, etc) and on the other hand the world of superpositions lives in some sense inside the classical world, i.e. determined externally by classical rules until we decide to explore it. This is given by the following composition of maps:

$$\begin{array}{ccc} US\mathbb{B} \times US\mathbb{B} & \xrightarrow{n} & U(S\mathbb{B} \otimes S\mathbb{B}) \\ & & \downarrow{\scriptstyle Um} \\ & & US(\mathbb{B} \times \mathbb{B}) \end{array}$$

that allows us to operate in a monoidal structure explicitly allowing the algebraic manipulation and then to return to the Cartesian product.

This is different from linear logic, where the ! stops any algebraic manipulation, i.e. $(!\mathbb{B}) \otimes (!\mathbb{B})$ is a product inside a monoidal category.

References

[1] Pablo Arrighi and Gilles Dowek. Lineal: a linear-algebraic lambda-calculus. *Logical Methods in Computer Science*, 13(1:8), 2017.

[2] Ali Assaf, Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson, and Benoît Valiron. Call-by-value, call-by-name and the vectorial behaviour of the algebraic λ-calculus. *Logical Methods in Computer Science*, 10(4:8), 2014.

[3] Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic (CSL 1994)*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. Springer, Berlin, Heidelberg, 1994.

[4] Alejandro Díaz-Caro and Gilles Dowek. Typing quantum superpositions and measurement. In *Theory and Practice of Natural Computing (TPNC 2017)*, volume 10687 of *Lecture Notes in Computer Science*, pages 281–293. Springer, Cham, 2017.

[5] Alejandro Díaz-Caro and Octavio Malherbe. A concrete categorical semantics for lambda-s. LSFA 2018. To appear in ENTCS. Pre-print at `arXiv:1806.09236`, 2018.

[6] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1):1–41, 2003.

[7] Jean-Yves Girard. Linear logic. *Theoretical Compututer Science*, 50:1–102, 1987.

[8] Lionel Vaux. The algebraic lambda calculus. *Mathematical Structures in Computer Science*, 19:1029–1059, 2009.