

QUANTUM CONTROL IN THE UNITARY SPHERE: LAMBDA- \mathcal{S}_1 AND ITS CATEGORICAL MODEL

ALEJANDRO DÍAZ-CARO ^{a,b} AND OCTAVIO MALHERBE ^{c,d}

^a Instituto de Ciencias de la Computación, CONICET–Universidad de Buenos Aires. Buenos Aires, Argentina

^b Depto. de Ciencia y Tecnología, Universidad Nacional de Quilmes. Bernal, Buenos Aires, Argentina
e-mail address: adiazcaro@icc.fcen.uba.ar

^c Instituto de Matemática y Estadística “Rafael Laguardia”, FIng, Universidad de la República. Montevideo, Uruguay

^d Depto. de Matemática y Aplicaciones, CURE, Universidad de la República. Maldonado, Uruguay
e-mail address: malherbe@fing.edu.uy

ABSTRACT. In a recent paper, a realizability technique has been used to give a semantics of a quantum lambda calculus. Such a technique gives rise to an infinite number of valid typing rules, without giving preference to any subset of those. In this paper, we introduce a valid subset of typing rules, defining an expressive enough quantum calculus. Then, we propose a categorical semantics for it. Such a semantics consists of an adjunction between the category of distributive-action spaces of value distributions (that is, linear combinations of values in the lambda calculus), and the category of sets of value distributions.

1. INTRODUCTION

In quantum programming languages, the control flow of programs divides models in two classes. On the one hand, there is the model of the QRAM [Kni96], or classical control [Sel04]. The classical control refers to a scheme where the quantum operations are performed in a specialized device, known as QRAM, attached to a classical computer, which instructs the device which operations to apply over which qubits. It is the more realistic and practical scenario. In this model, the quantum operations are given by a series of “black boxes”. An example of this is the quantum lambda calculus [SV06], as well as several high-level quantum programming languages such as Quipper [GLR⁺13] and QWIRE [PRZ17]. The kind of problems that this model dealt with is, for example, to forbid cloning unknown qubits, since the non-cloning theorem states that there is no universal cloning machine.

On the other hand, there is the model of quantum control, with a parallel agenda. The ultimate motivation of this model is to extend the Curry-Howard isomorphism relating type

Key words and phrases: Lambda calculus, Quantum computing, Categorical semantics.

Partially funded by PIP 11220200100368CO, PICT-2019-1272, 21STIC10 Qapla’, PUNQ 1342/19 and ECOS-Sud A17C03 QuCa.

theory with logics, to the quantum case. Indeed, there is a long line of research on Quantum Logic started by the pioneer work of Birkhoff and Von Neumann in the 30's [BVN36]. However, the connection from this logic to a lambda calculus is unknown.

One of the first works on the quantum control approach is the development of QML [AG05], where the quantum control is expressed by the quantum if “if^o” which, given a superposition of $|0\rangle$ and $|1\rangle$, produces a superposition of its two output branches. However, a superposition of the form $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ is a valid qubit only if its norm is equal to 1, so, if $|\alpha|^2 + |\beta|^2 = 1$. Therefore, superposing the two output branches would be valid, only if the norm of this term is equal to 1. Therefore, for example, the term if^o $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ then s else t is valid only if s and t are orthogonal, and so it preserves the norm. Thus, QML introduced a notion of norm for a small subset of terms. Indeed, consider the type *Bool*, with its two orthogonal values *true* and *false*. To detect if $t : \text{Bool}$ and $r : \text{Bool}$ are orthogonal, means to reduce t and r and to compare them. So the orthogonality question in typing have been an open question for many years, until the work [DCGMV19], which provided a partial answer.

The long path to this partial answer started at Lineal [AD08, AD17], which is an untyped extension to the lambda calculus allowing for linear combinations of terms. This way, if s and t are two terms, so is its formal linear combination $\alpha \cdot s + \beta \cdot t$, with $\alpha, \beta \in \mathbb{C}$. Unitary matrices are not expressed as given black boxes, but they can be constructed.

Quantum programs can be expressed in Lineal, except for the quantum measurement, which is left out of the system. However, Lineal is not restricted to only quantum programs. In particular, the vectors are not ensured to be of norm 1, since it would require checking for orthogonality between vectors. Neither functions are proved to behave as isometries, as needed by the quantum theory. One main feature of Lineal, although, is the fact that all the functions, even if they are not forced to be linear or isometries, are treated linearly: if a function $\lambda x.s$ is applied to a formal linear combination $\alpha \cdot v + \beta \cdot w$, it distributes linearly as follows:

$$(\lambda x.s)(\alpha \cdot v + \beta \cdot w) \longrightarrow \alpha \cdot (\lambda x.s)v + \beta \cdot (\lambda x.s)w$$

generalising the quantum-if from QML.

A drawback in taking all functions as linear, is that adding measurement was not trivial, since a measurement is not a linear operation: If M is a measurement operator, $M(\alpha \cdot v + \beta \cdot w)$ does not behave as $\alpha \cdot Mv + \beta \cdot Mw$.

Lambda- \mathcal{S} [DCD17, DCDR19] is a typed lambda calculus based on Lineal, mainly focused on adding measurement to the calculus. Instead of treating all functions as linear, its types enforce linearity when the argument is a superposition, and allow for duplication when it is not. This is done by labelling superpositions with a modality \mathcal{S} . Any term typed by \mathcal{S} is treated linearly, so only basis terms are duplicable. It is argued to be somehow the dual to Intuitionistic Linear Logic, where duplicable terms are marked (by a !). Indeed, in [DCM19, DCM20b, DCM20a] a categorical model for Lambda- \mathcal{S} has been proposed, obtained by a monoidal monad determined by a monoidal adjunction $(S, m) \dashv (U, n)$ and interpreting \mathcal{S} as the monad US —exactly the opposite to the ! of linear logic, which in the literature is often interpreted as the comonad SU (see [Mel03]). This implies that on the one hand there is a tight control of the Cartesian structure of the model, and on the other hand the world of superpositions lives inside the classical world, i.e. determined externally by classical rules until one decides to explore it. This is given by the following composition of maps:

$$USA \times USA \xrightarrow{n} U(SA \otimes SA) \xrightarrow{Um} US(A \times A)$$

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. $(!A) \otimes (!A)$ is a product inside a monoidal category. A concrete example is an adjunction between the categories **Set** of sets and **Vec** of vector spaces [DCM19, DCM20b].

The problem of orthogonality has been finally addressed in [DCGMV19], which provides another type system for **Lineal**, ensuring superpositions to be in the unitary sphere \mathcal{S}_1 (that is, norm-1 vectors). It also characterizes isometries via a specific type. On this system, measurement has been left out of the equation, since **Lambda- \mathcal{S}** already showed how to add measurement on **Lineal**, so, it is already known how to add measurement and it is no longer a problem. This system ensuring norm-1 vectors and characterizing isometries has been obtained by means of realizability techniques [Kle45, van08, Kri09, Miq11]. Instead of deriving a computational meaning of proofs once the type system is set up, the idea of realizability is to consider the type system as a by-product of the operational semantics—programs are then potential realizers of types. For example, a program behaving as the identity will be a realizer of $A \rightarrow A$, regardless of its inner structure. Realizability is a powerful and modular framework amenable to many systems (see [Bru14]). So, one particularity is that the typing rules are probable lemmas (any typing rule conforming the semantics, is a valid rule), hence, the set of rules is potentially infinite. On this scheme, there is a modality \sharp , with a similar behaviour to the \mathcal{S} of **Lambda- \mathcal{S}** . The claimed main goal of this system has been to solve the long-standing issue of how to ensure norm-1 superpositions, and characterize unitary functions.

The goal of the present paper is to extract a (finite) fixed type system following the realizability semantics [DCGMV19], a calculus we call **Lambda- \mathcal{S}_1** , ensuring norm-1 superpositions. We also give a categorical model for this calculus.

Hence, the main contributions are twofold. On the one hand, we give the definition of **Lambda- \mathcal{S}_1** , which is not trivial since [DCGMV19] provides only a method to produce an infinite type system. On the other hand, the second main contribution is the categorical model, which has some common grounds with the concrete model of **Lambda- \mathcal{S}** [DCM19, DCM20b], however, the chosen categories this time are not **Set** and **Vec**, but categories that use the fact that values in our calculus form a distributive-action space (an algebraic structure similar to a vector space, where its additive structure is a semi-group). Summarising, the main novelty and contribution of this paper is presenting a model for quantum computing in the quantum control paradigm, which we show to be complete on qubits (Theorem 5.6).

We left the measurement operator out of the obtained system, only for the sake of simplicity. The inclusion of the measurement operator was a problem for **Lineal**. However, after **Lambda- \mathcal{S}** [DCD17, DCDR19] tackled this problem by considering linear types and the modality \mathcal{S} , it is no longer a problem. Since **Lambda- \mathcal{S}_1** uses the same modality (here written \sharp), the inclusion or not of a measurement operator does not suppose a challenge any more. So, adding a measurement operator as the one from **Lambda- \mathcal{S}** is not difficult, since the problems were already solved on that system, and **Lambda- \mathcal{S}_1** follows the same line. However, adding a measurement operator implies to have a probabilistic rewrite system, which demands an extra monad (the probabilistic monad) to be added to the model. While this addition would be easy (cf. [DCM19, DCM20b]), it introduces superfluous complexity to the system making its model less clear.

In Section 2 we introduce the calculus **Lambda- \mathcal{S}_1** . We prove its main correctness properties such as progress, subject reduction, and strong normalization in Section 3. In

Pure values	$v, w := x \mid \lambda x. \vec{s} \mid * \mid (v_1, v_2) \mid \mathbf{inl}(v) \mid \mathbf{inr}(v)$	
Pure terms	$s, t := v \mid st \mid t; \vec{s} \mid \mathbf{let} (x_1, x_2) = t \mathbf{in} \vec{s} \mid \mathbf{match} t \{ \mathbf{inl}(x_1) \mapsto \vec{s}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{s}_2 \}$	
Value distrib.	$\vec{v}, \vec{w} := v \mid \vec{v} + \vec{w} \mid \alpha \cdot \vec{v}$	$(\alpha \in \mathbb{C})$
Term distrib.	$\vec{s}, \vec{t} := t \mid \vec{s} + \vec{t} \mid \alpha \cdot \vec{t}$	$(\alpha \in \mathbb{C})$

Table 1: Grammar of terms

$\vec{t}_1 + \vec{t}_2 \equiv \vec{t}_2 + \vec{t}_1$	$(\vec{t}_1 + \vec{t}_2) + \vec{t}_3 \equiv \vec{t}_1 + (\vec{t}_2 + \vec{t}_3)$	$1 \cdot \vec{t} \equiv \vec{t}$
$\alpha \cdot (\beta \cdot \vec{t}) \equiv \alpha\beta \cdot \vec{t}$	$(\alpha + \beta) \cdot \vec{t} \equiv \alpha \cdot \vec{t} + \beta \cdot \vec{t}$	$\alpha \cdot (\vec{t}_1 + \vec{t}_2) \equiv \alpha \cdot \vec{t}_1 + \alpha \cdot \vec{t}_2$

Table 2: Congruence rules on term distributions

Section 3.4 we show the expressiveness of Lambda- \mathcal{S}_1 , which includes the simply typed lambda calculus with addition and pairs, plus the isometries. In Section 4 we introduce its categorical model. In Section 5 we prove the soundness of the model, and the completeness of the type $\sharp(\mathbb{U} + \mathbb{U})$, which corresponds to \mathbb{C}^2 (the type of qubits). We conclude in Section 6 with some final remarks.

2. LAMBDA- \mathcal{S}_1

2.1. Terms. In Table 1 we give the grammar of terms for Lambda- \mathcal{S}_1 . Those fall into two categories: “pure” and “distributions”, and those in two subcategories, of terms and values. The idea is that a distribution is a linear combination of a pure terms. Formally, the set of distributions is equipped with a congruence \equiv that is generated from the 6 rules of Table 2. We say that a term $\vec{t} := \sum_{i=1}^n \alpha_i \cdot t_i$ is given in canonical form when $t_j \neq t_k$ for all j, k .

From now on, we consider term distributions modulo the congruence \equiv , and simply write $\vec{t} = \vec{t}'$ for $\vec{t} \equiv \vec{t}'$. This convention does not affect *inner*—or *raw*—distributions (which occur within a pure term, for instance in the body of an abstraction), that are still considered only up to α -conversion¹.

Pure terms and term distributions are intended to be evaluated according to a call-by-pure-values strategy², which is a declination of the call-by-value strategy in a computing environment where all functions are linear by construction. In its original form [AD08, AD17], a superposition $t(v + w)$ reduced to $(tv + tw)$, while in our case following [DCGMV19], the first term is not even in the grammar, but it is just a notation for the former. This notation extends the syntactic constructs of the language by linearity, proceeding as follows: for all value distributions $\vec{v} = \sum_{i=1}^n \alpha_i \cdot v_i$ and $\vec{w} = \sum_{j=1}^m \beta_j \cdot w_j$, and for all term distributions $\vec{s}_1, \vec{s}_2, \vec{t} = \sum_{k=1}^p \gamma_k \cdot t_k$, and $\vec{s} = \sum_{\ell=1}^q \delta_\ell \cdot s_\ell$ we have the notations given in Table 3. Notice that $\vec{t}s$ is not in the grammar nor in the notation: the term at the left of an application must

¹Intuitively, a distribution that appears in the body of an abstraction (or in the body of a let-construct, or in a branch of a match-construct) does not represent a real superposition, but *machine code* that will produce later a particular superposition, after some substitution has been performed.

²Called call-by-basis in [ADCP⁺14, DCGMV19], and simply call-by-value in [AD08, AD17].

$(\vec{v}, \vec{w}) := \sum_{i=1}^n \sum_{j=1}^k \alpha_i \beta_j \cdot (v_i, w_j)$	$t \vec{s} := \sum_{\ell=1}^q \delta_\ell \cdot t s_\ell$
$\text{inl}(\vec{v}) := \sum_{i=1}^n \alpha_i \cdot \text{inl}(v_i)$	$\vec{t}; \vec{s} := \sum_{k=1}^p \gamma_k \cdot (t_k; \vec{s})$
$\text{inr}(\vec{v}) := \sum_{i=1}^n \alpha_i \cdot \text{inr}(v_i)$	$\text{let } (x, y) = \vec{t} \text{ in } \vec{s} := \sum_{k=1}^p \gamma_k \cdot (\text{let } (x, y) = t_k \text{ in } \vec{s})$
$\text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{s}_1 \mid \text{inr}(x_2) \mapsto \vec{s}_2 \} :=$	
$\sum_{k=1}^p \gamma_k \cdot (\text{match } t_k \{ \text{inl}(x_1) \mapsto \vec{s}_1 \mid \text{inr}(x_2) \mapsto \vec{s}_2 \})$	
Where $\vec{v} = \sum_{i=1}^n \alpha_i \cdot v_i$, $\vec{w} = \sum_{j=1}^m \beta_j \cdot w_j$, $\vec{t} = \sum_{k=1}^p \gamma_k \cdot t_k$, $\vec{s} = \sum_{\ell=1}^q \delta_\ell \cdot s_\ell$	

Table 3: Notations for linear constructions

$(\lambda x . \vec{t}) v \longrightarrow \vec{t}[x := v]$
$*; \vec{s} \longrightarrow \vec{s}$
$\text{let } (x, y) = (v, w) \text{ in } \vec{s} \longrightarrow \vec{s}[x := v, y := w]$
$\text{match inl}(v) \{ \text{inl}(x_1) \mapsto \vec{s}_1 \mid \text{inr}(x_2) \mapsto \vec{s}_2 \} \longrightarrow \vec{s}_1[x_1 := v]$
$\text{match inr}(v) \{ \text{inl}(x_1) \mapsto \vec{s}_1 \mid \text{inr}(x_2) \mapsto \vec{s}_2 \} \longrightarrow \vec{s}_2[x_2 := v]$
$\frac{t \longrightarrow \vec{r}}{s t \longrightarrow s \vec{r}} \quad \frac{t \longrightarrow r}{t v \longrightarrow r v} \quad \frac{t \longrightarrow \vec{r}}{t; \vec{s} \longrightarrow \vec{r}; \vec{s}} \quad \frac{t \longrightarrow \vec{r}}{\text{let } (x, y) = t \text{ in } \vec{s} \longrightarrow \text{let } (x, y) = \vec{r} \text{ in } \vec{s}}$
$\frac{t \longrightarrow \vec{r}}{\text{match } t \{ \text{inl}(x_1) \mapsto \vec{s}_1 \mid \text{inr}(x_2) \mapsto \vec{s}_2 \} \longrightarrow \text{match } \vec{r} \{ \text{inl}(x_1) \mapsto \vec{s}_1 \mid \text{inr}(x_2) \mapsto \vec{s}_2 \}}$
$\frac{t \longrightarrow \vec{r}}{\alpha \cdot t + \vec{s} \longrightarrow \alpha \cdot \vec{r} + \vec{s}}$

Table 4: Rewrite rules

be a pure term. The reason is that we will focus in isometries, and the linear combination of isometries is not necessarily an isometry.

We write \mathbb{V} , $\vec{\mathbb{V}}$, Λ , and $\vec{\Lambda}$ to the sets of pure values, value distributions, pure terms, and term distributions respectively.

Finally, in Table 4 we give the rewrite relation. As usual, we write \longrightarrow^* for the reflexive and transitive closure of \longrightarrow .

As the reader may have noticed from the grammar in Table 1 and the congruence rules from Table 2, there is no such a thing as a “null vector” in the grammar, and so $0 \cdot \vec{t}$ does not simplify. Indeed, we do not want in general a null vector $\vec{0}$ within term distributions, since $0 \cdot \vec{v}$ must have a type compatible with \vec{v} , while $\vec{0}$ do not have any restriction. In fact, the set of value distributions do not form a vector space for this reason. However, we still can define an analogous to an inner product and, from it, a notion of orthogonality.

Definition 2.1 (Pseudo inner product and orthogonality). Let $\vec{v} = \sum_{i=1}^n \alpha_i \cdot v_i$ and $\vec{w} = \sum_{j=1}^m \beta_j \cdot w_j$ be two value distributions in canonical form. Then we define the pseudo inner

product $(\cdot | \cdot) : \vec{V} \times \vec{V} \rightarrow \mathbb{C}$ as

$$(\vec{v} | \vec{w}) := \sum_{i=1}^n \sum_{j=1}^m \bar{\alpha}_i \beta_j \delta_{v_i, w_j}$$

where δ_{v_i, w_j} is the Kronecker delta, i.e. it is 1 if $v_i = w_j$, and 0 otherwise. We write $\vec{v} \perp \vec{w}$ if $(\vec{v} | \vec{w}) = 0$.

Remark that in this section, we are not defining the mathematical structure we have, just pinpointing the fact that it is not a vector space, and so we cannot define an inner product. However, we have defined a function, which we call pseudo inner product, which is enough for the syntactic treatment of the calculus we are introducing. In Section 4 we will give the rigorous mathematical definitions needed to give a denotational semantics of such a calculus.

2.2. Types. Types are produced by the following grammar

$$A := \mathbb{U} \mid \sharp A \mid A + A \mid A \times A \mid A \rightarrow A$$

The type $\sharp A$ is meant to type term distributions of pure terms of type A . This is a subset of the grammar from in [DCGMV19]. In particular, we do not include the construction $\flat A$, however we use the notation A^\flat (read: A is flat) for the following property: A does not contain any \sharp , except, maybe, at the right of an arrow. We also write $A \oplus B := \sharp(A + B)$ and $A \otimes B := \sharp(A \times B)$. In [DCGMV19] there is also a type $A \Rightarrow B$, which contains the superposition of arrows, which are valid arrows. In our case, we decided to simplify the language by not allowing superpositions of arrows to be arrows, and so this particular type construct is not used.

In Table 5 we give a subtyping relation between types. In particular, $\sharp\sharp A < \sharp A$ since a term distribution of term distributions of type $\sharp A$ is just a term distribution of pure terms of type A ,

In Table 6 we give the typing rules, where we use the notation $\Gamma \vdash (\Delta_1 \vdash \vec{v}_1 \perp \Delta_2 \vdash \vec{v}_2) : A$ for

$$\left\{ \begin{array}{l} \Gamma, \Delta_1 \vdash \vec{v}_1 : A \\ \Gamma, \Delta_2 \vdash \vec{v}_2 : A \\ \theta_{\Gamma, \Delta_1}(\vec{v}_1) \perp \theta_{\Gamma, \Delta_2}(\vec{v}_2) \end{array} \right.$$

where for any context Γ , θ_Γ is a substitution of variables by pure values of the same type. Notice that substituting a variable in a value by a pure value, keep the term being a value. When $\Delta_1 = \Delta_2 = \emptyset$, we just write $\Gamma \vdash (\vec{v}_1 \perp \vec{v}_2) : A$.

The given type system is linear on types, except flat types (i.e. any type A such that A^\flat). Notice that the type system uses the notations from Table 3, for example, the rule

$$\frac{\Gamma \vdash \vec{v} : A \quad \Delta \vdash \vec{w} : B}{\Gamma, \Delta \vdash (\vec{v}, \vec{w}) : A \times B} \text{Pair}$$

is in fact

$$\frac{\Gamma \vdash \sum_i \alpha_i \cdot v_i : A \quad \Delta \vdash \sum_j \beta_j \cdot w_j : B}{\Gamma, \Delta \vdash \sum_{ij} \alpha_i \beta_j \cdot (v_i, w_j) : A \times B} \text{Pair}$$

$\overline{A \leq A}$	$\frac{A \leq B \quad B \leq C}{A \leq C}$	$\overline{A \leq \#A}$	$\overline{\#\#A \leq \#A}$
$\frac{A \leq A' \quad B \leq B'}{A' \rightarrow B \leq A \rightarrow B'}$	$\frac{A \leq A' \quad B \leq B'}{A \times B \leq A' \times B'}$	$\frac{A \leq A' \quad B \leq B'}{A + B \leq A' + B'}$	

Table 5: Subtyping

$\frac{}{x : A \vdash x : A}$ Ax	$\frac{\Gamma, x : A \vdash \vec{t} : B}{\Gamma \vdash \lambda x. \vec{t} : A \rightarrow B}$ Lam	$\frac{\Gamma \vdash t : A \rightarrow B \quad \Delta \vdash \vec{s} : A}{\Gamma, \Delta \vdash t \vec{s} : B}$ App
$\frac{}{\vdash * : \mathbb{U}}$ Void	$\frac{\Gamma \vdash t : \mathbb{U} \quad \Delta \vdash \vec{s} : A}{\Gamma, \Delta \vdash t; \vec{s} : A}$ PureSeq	$\frac{\Gamma \vdash \vec{t} : \#\mathbb{U} \quad \Delta \vdash \vec{s} : \#A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : \#A}$ UnitarySeq
$\frac{\Gamma \vdash \vec{v} : A \quad \Delta \vdash \vec{w} : B}{\Gamma, \Delta \vdash (\vec{v}, \vec{w}) : A \times B}$ Pair		
$\frac{\Gamma \vdash t : A \times B \quad \Delta, x : A, y : B \vdash \vec{s} : C}{\Gamma, \Delta \vdash \text{let } (x, y) = t \text{ in } \vec{s} : C}$ PureLet		
$\frac{\Gamma \vdash \vec{t} : A \otimes B \quad \Delta, x : \#A, y : \#B \vdash \vec{s} : \#C}{\Gamma, \Delta \vdash \text{let } (x, y) = \vec{t} \text{ in } \vec{s} : \#C}$ UnitaryLet		
$\frac{\Gamma \vdash v : A}{\Gamma \vdash \text{inl}(v) : A + B}$ InL		
$\frac{\Gamma \vdash v : B}{\Gamma \vdash \text{inr}(v) : A + B}$ InR		
$\frac{\Gamma \vdash t : A + B \quad \Delta \vdash (x_1 : A \vdash \vec{v}_1 \perp x_2 : B \vdash \vec{v}_2) : C}{\Gamma, \Delta \vdash \text{match } t \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : C}$ PureMatch		
$\frac{\Gamma \vdash \vec{t} : A \oplus B \quad \Delta \vdash (x_1 : \#A \vdash \vec{v}_1 \perp x_2 : \#B \vdash \vec{v}_2) : \#C}{\Gamma, \Delta \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : \#C}$ UnitaryMatch		
$\frac{(k \neq h) \vdash (\vec{v}_k \perp \vec{v}_h) : A \quad \sum_{j=1}^m \alpha_j ^2 = 1 \quad m \geq 1 \quad A \neq B \rightarrow C}{\vdash \sum_{j=1}^m \alpha_j \cdot \vec{v}_j : \#A}$ Sup		
$\frac{\Gamma \vdash \vec{t} : A \quad A \leq B \leq}{\Gamma \vdash \vec{t} : B}$ \leq		
$\frac{\Gamma \vdash \vec{t} : A \quad \vec{t} \equiv \vec{r} \equiv}{\Gamma \vdash \vec{r} : A}$ \equiv		
$\frac{\Gamma \vdash \vec{t} : B \quad A^b}{\Gamma, x : A \vdash \vec{t} : B}$ Weak		
$\frac{\Gamma, x : A, y : A \vdash \vec{t} : B \quad A^b}{\Gamma, x : A \vdash \vec{t}[y := x] : B}$ Contr		

Table 6: Typing system

3. SYNTACTIC PROPERTIES

This section is devoted to proving several syntactic properties of the calculus introduced in the previous section. In particular, Progress (Section 3.1), Subject Reduction (Section 3.2), and Strong Normalization (Section 3.3). The last property is shown by proving that the calculus is a valid fragment with respect to the realizability semantics given in [DCGMV19]. Since Lambda-S₁ is a fragment of a bigger calculus, we also show that it is expressive enough for quantum computing (Section 3.4).

3.1. Progress.

Theorem 3.1 (Progress). *If $\vdash \vec{t} : A$ and \vec{t} does not reduce, then $\vec{t} \in \vec{V}$.*

Proof. We proceed by induction on \vec{t} .

- If \vec{t} is a value distribution, we are done.
- Let $\vec{t} = s\vec{r}$. Then $\vdash s : B \Rightarrow A$, but since \vec{t} does not reduce, neither does s , so, by the induction hypothesis $s \in \vec{V}$, and, due to its type, the only possibility is $s \equiv \lambda x.\vec{s}'$, which is absurd since \vec{t} does not reduce.
- $\vec{t} = \vec{s};\vec{r}$. Then there are two possibilities:
 - $\vdash \vec{s} : \mathbb{U}$, but since \vec{t} does not reduce, neither does \vec{s} , so, by the induction hypothesis \vec{s} is a value, and, due its type, the only possibility is $\vec{s} \equiv *$, which is absurd since \vec{t} does not reduce.
 - $\vdash \vec{s} : \sharp\mathbb{U}$, but since \vec{t} does not reduce, neither does \vec{s} , so, by the induction hypothesis \vec{s} is a value, and, due its type, the only possibility is $\vec{s} \equiv \alpha \cdot *$, which is absurd since \vec{t} does not reduce.
- $\vec{t} = \mathbf{let} (x, y) = \vec{s} \mathbf{in} \vec{r}$. Then there are two possibilities:
 - $\vdash \vec{s} : B \times C$, but since \vec{t} does not reduce, neither does \vec{s} , so, by the induction hypothesis \vec{s} is a value, and, due its type, the only possibility is $\vec{s} \equiv (v_1, v_2)$, which is absurd since \vec{t} does not reduce.
 - $\vdash \vec{s} : \sharp(B \times C)$, but since \vec{t} does not reduce, neither does \vec{s} , so, by the induction hypothesis \vec{s} is a value, and, due its type, the only possibility is $\vec{s} \equiv (\vec{v}_1, \vec{v}_2)$, which is absurd since \vec{t} does not reduce.
- $\vec{t} = \mathbf{match} \vec{s} \{ \mathbf{inl}(x_1) \mapsto \vec{v}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{v}_2 \}$. Then there are two possibilities:
 - $\vdash \vec{s} : B + C$, but since \vec{t} does not reduce, neither does \vec{s} , so, by the induction hypothesis \vec{s} is a value, and, due its type, the only possibilities are $\vec{s} \equiv \mathbf{inl}(s')$ or $\vec{s} \equiv \mathbf{inr}(s')$, both of which are absurd since \vec{t} does not reduce.
 - $\vdash \vec{s} : \sharp(B + C)$, but since \vec{t} does not reduce, neither does \vec{s} , so, by the induction hypothesis \vec{s} is a value, and, due its type, the only possibilities are $\vec{s} \equiv \mathbf{inl}(s')$ or $\vec{s} \equiv \mathbf{inr}(s')$, both of which are absurd since \vec{t} does not reduce. \square

3.2. Subject reduction. The type preservation with our chosen typing rules is proven now. We first need a substitution lemma.

Lemma 3.2. *Let $\Gamma, x : A \vdash \vec{t} : B$, $\Delta \vdash \vec{v} : A$, and Δ^b , then $\Gamma, \Delta \vdash \vec{t}[x := \vec{v}] : B$.*

Proof. By induction on \vec{t} .

- If $x \notin FV(\vec{t})$ then a straightforward generation lemma shows that $x : A$ can be removed from $\Gamma, x : A \vdash \vec{t} : B$, and from $\Gamma \vdash \vec{t} : B$ we can derive $\Gamma, \Delta \vdash \vec{t} : B$ by rule **Weak**. Notice that $t[x := \vec{v}] = \vec{t}$.
- Let $\vec{t} = x$, then Γ^b and $A \leq B$, and so, by rules \leq and **Weak**, we have $\Gamma, \Delta \vdash \vec{v} : B$. Notice that $x[x := \vec{v}] = \vec{v}$.
- Let $\vec{t} = \lambda y.\vec{s}$, then $C \Rightarrow D \leq B$ and $\Gamma, x : A, y : C \vdash \vec{s} : D$. Hence, by the induction hypothesis, $\Gamma, \Delta, y : C \vdash \vec{s}[x := \vec{v}] : D$. Therefore, by rules **Lam** and \leq , $\Gamma, \Delta \vdash \lambda y.\vec{s}[x := \vec{v}] : B$. Notice that $\lambda y.\vec{s}[x := \vec{v}] = (\lambda y.\vec{s})[x := \vec{v}]$.
- Let $\vec{t} = (v_1, v_2)$, then $\Gamma_1 \vdash v_1 : B_1$ and $\Gamma_2 \vdash v_2 : B_2$ with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$ and $B_1 \times B_2 = B$. Assume that $\Gamma_1 = (\Gamma'_1, x : A)$. Then, by the induction hypothesis, $\Gamma'_1, \Delta \vdash v_1[x := \vec{v}] : B_1$, and so, by rule **Pair**, $\Gamma'_1, \Gamma_2, \Delta \vdash (v_1[x := \vec{v}], v_2) : B$. Notice that $(\Gamma'_1, \Gamma_2) = \Gamma$ and $(v_1[x := \vec{v}], v_2) = (v_1, v_2)[x := \vec{v}]$. The case where $\Gamma_2 = (\Gamma'_2, x : A)$ is analogous.

- Let $\vec{t} = \mathbf{inl}(w)$, then $\Gamma, x : A \vdash w : B_1$, with $B = B_1 + B_2$, so, by the induction hypothesis, $\Gamma, \Delta \vdash w[x := \vec{v}] : B_1$ and, by rule \mathbf{InL} , $\Gamma, \Delta \vdash \mathbf{inl}(w[x := \vec{v}]) : B$. Notice that $\mathbf{inl}(w[x := \vec{v}]) = \mathbf{inl}(w)[x := \vec{v}]$.
- Let $\vec{t} = \mathbf{inr}(w)$, this case is analogous to the previous.
- Let $\vec{t} = s\vec{r}$, then $\Gamma_1 \vdash s : C \Rightarrow D$ and $\Gamma_2 \vdash \vec{r} : C$, with $D \leq B$ and $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$. Assume $\Gamma_1 = (\Gamma'_1, x : A)$. Then, by the induction hypothesis, $\Gamma'_1, \Delta \vdash s[x := \vec{v}] : C \Rightarrow D$. Thus, by rules \mathbf{App} , and \leq , $\Gamma'_1, \Gamma_2, \Delta \vdash s[x := \vec{v}]\vec{r} : B$. Notice that $(\Gamma'_1, \Gamma_2) = \Gamma$ and $s[x := \vec{v}]\vec{r} = (s\vec{r})[x := \vec{v}]$. The case where $\Gamma_2 = (\Gamma'_2, x : A)$ is analogous.
- Let $\vec{t} = \vec{r}; \vec{s}$, then there are two possibilities:
 - (1) Either $\Gamma_1 \vdash \vec{r} : \mathbb{U}$ and $\Gamma_2 \vdash s : B$, with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$,
 - (2) or $\Gamma_1 \vdash \vec{r} : \sharp\mathbb{U}$ and $\Gamma_2 \vdash \vec{s} : \sharp C$, with $B = \sharp C$, and $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$.
 In any case, assume $\Gamma_1 = (\Gamma'_1, x : A)$. Then, by the induction hypothesis, $\Gamma'_1, \Delta \vdash \vec{r}[x := \vec{v}] : E$ (with $E = \mathbb{U}$ in the first case or $E = \sharp\mathbb{U}$ in the second). Thus, by rules $\mathbf{PureSeq}$ or $\mathbf{UnitarySeq}$, $\Gamma'_1, \Gamma_2, \Delta \vdash \vec{r}[x := \vec{v}]; \vec{s} : B$. Notice that $(\Gamma'_1, \Gamma_2) = \Gamma$ and $\vec{r}[x := \vec{v}]; \vec{s} = (\vec{r}; \vec{s})[x := \vec{v}]$. The case where $\Gamma_2 = (\Gamma'_2, x : A)$ is analogous.
- Let $\vec{t} = \mathbf{let} (x_1, x_2) = \vec{r} \mathbf{in} \vec{s}$, then there are two possibilities:
 - (1) Either $\Gamma_1 \vdash \vec{r} : C \times D$ and $\Gamma_2, x_1 : C, x_2 : D \vdash \vec{s} : B$, with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$,
 - (2) or $\Gamma_1 \vdash \vec{r} : \sharp(C \times D)$ and $\Gamma_2, x_1 : \sharp C, x_2 : \sharp D \vdash \vec{s} : \sharp C$, with $B = \sharp C$ and $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$.
 In any case, assume $\Gamma_1 = (\Gamma'_1, x : A)$. Then, by the induction hypothesis, $\Gamma'_1, \Delta \vdash \vec{r}[x := \vec{v}] : E$ (with $E = C \times D$ in the first case or $E = \sharp(C \times D)$ in the second). Thus, by rules $\mathbf{PureLet}$ or $\mathbf{UnitaryLet}$, $\Gamma'_1, \Gamma_2, \Delta \vdash \mathbf{let} (x_1, x_2) = \vec{r}[x := \vec{v}] \mathbf{in} \vec{s} : B$. Notice that $(\Gamma'_1, \Gamma_2) = \Gamma$ and $\mathbf{let} (x_1, x_2) = \vec{r}[x := \vec{v}] \mathbf{in} \vec{s} = (\mathbf{let} (x_1, x_2) = \vec{r} \mathbf{in} \vec{s})[x := \vec{v}]$. The case where $\Gamma_2 = (\Gamma'_2, x : A)$ is analogous.
- Let $\vec{t} = \mathbf{match} \vec{r} \{ \mathbf{inl}(x_1) \mapsto \vec{v}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{v}_2 \}$, then there are two possibilities:
 - (1) Either $\Gamma_1 \vdash \vec{r} : C + D$ and $\Gamma_2 \vdash (x_1 : C \vdash \vec{v}_1 \perp x_2 : D \vdash \vec{v}_2) : B$, with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$,
 - (2) or $\Gamma_1 \vdash \vec{r} : \sharp(C + D)$ and $\Gamma_2 \vdash (x_1 : \sharp C \vdash \vec{v}_1 \perp x_2 : \sharp D \vdash \vec{v}_2) : \sharp C$, with $B = \sharp C$ and $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$.
 In any case, assume $\Gamma_1 = (\Gamma'_1, x : A)$. Then, by the induction hypothesis, $\Gamma'_1, \Delta \vdash \vec{r}[x := \vec{v}] : E$ (with $E = C + D$ in the first case or $E = \sharp(C + D)$ in the second). Thus, by rules $\mathbf{PureMatch}$ or $\mathbf{UnitaryMatch}$, $\Gamma'_1, \Gamma_2, \Delta \vdash \mathbf{match} \vec{r}[x := \vec{v}] \{ \mathbf{inl}(x_1) \mapsto \vec{v}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{v}_2 \} : B$. Notice that $(\Gamma'_1, \Gamma_2) = \Gamma$ and $\mathbf{match} \vec{r}[x := \vec{v}] \{ \mathbf{inl}(x_1) \mapsto \vec{v}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{v}_2 \} = (\mathbf{match} \vec{r} \{ \mathbf{inl}(x_1) \mapsto \vec{v}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{v}_2 \})[x := \vec{v}]$.
 Now assume $\Gamma_2 = (\Gamma'_2, x : A)$, then by the induction hypothesis, $\Gamma_2, \Delta, x_1 : C \vdash \vec{v}_1[x := \vec{v}] : B$ and $\Gamma_2, \Delta, x_2 : D \vdash \vec{v}_2[x := \vec{v}] : B$. Notice that the \perp condition is preserved under substitution, therefore, $\Gamma_2, \Delta \vdash (x_1 : C \vdash \vec{v}_1[x := \vec{v}] \perp x_2 : D \vdash \vec{v}_2[x := \vec{v}]) : B$, and we close analogously to the previous case.
- Let $\vec{t} = \sum_{i=1}^n \alpha_i \cdot \vec{v}_i$, then $x \notin FV(\vec{t})$ and we are in the first case. □

Theorem 3.3 (Subject reduction). *If $\Gamma \vdash \vec{t} : A$ and $\vec{t} \longrightarrow \vec{s}$, then $\Gamma \vdash \vec{s} : A$.*

Proof. By induction on the relation \longrightarrow . The proof is straightforward, using Lemma 3.2. □

3.3. Strong normalization. We prove that $\text{Lambda-}\mathcal{S}_1$ is valid with respect to the realizability model given in [DCGMV19] (Theorem 3.4), which implies strong normalization (Corollary 3.5)³.

In [DCGMV19] a realizability semantics has been defined, and the type system of the language is determined by any rule following such a semantics. In particular, the realizability predicate [DCGMV19, Def. IV.2] states that a term \vec{t} is a realizer of a type A (notation $\vec{t} \Vdash A$) if and only if \vec{t} rewrites to a value in the interpretation of A . Then, a typing judgement $\vdash \vec{t} : A$ is valid if and only if, $\vec{t} \Vdash A$. In this paper, we have fixed a set of typing rules in Table 6, some of which are already proven to be valid in [DCGMV19], while others are proved to be correct next (Theorem 3.4). For the sake of self-containment, we include the needed definitions from the realizability semantics [DCGMV19].

Let $\mathcal{S}_1 = \{\vec{v} : (\vec{v}|\vec{v}) = 1\}$. The interpretation $(\cdot)_R$ of types is given by

$$\begin{aligned} (\mathbb{U})_R &= \{*\} \\ (\sharp A)_R &= \text{span}((A)_R) \cap \mathcal{S}_1 \\ (A + B)_R &= \{\text{inl}(\vec{v}) : \vec{v} \in (A)_R\} \cup \{\text{inr}(\vec{w}) : \vec{w} \in (B)_R\} \\ (A \times B)_R &= \{(\vec{v}, \vec{w}) : \vec{v} \in (A)_R, \vec{w} \in (B)_R\} \\ (A \rightarrow B)_R &= \{\lambda x. \vec{t} : \forall \vec{v} \in (A)_R, \vec{t}\langle x := \vec{v} \rangle \Vdash B\} \end{aligned}$$

where $\vec{t} \Vdash A$ means that \vec{t} reduces to a value in $(A)_R$, and $\vec{t}\langle x := \vec{v} \rangle$ is the bilinear substitution defined as follows: Let $\vec{t} = \sum_i \alpha_i \cdot t_i$ and $\vec{v} = \sum_j \beta_j \cdot v_j$. Then, $\vec{t}\langle x := \vec{v} \rangle := \sum_i \sum_j \alpha_i \beta_j \cdot t_i[x := v_j]$.

If σ is a substitution, we may write $\vec{t}\langle\sigma\rangle$ for the term distribution \vec{t} substituted by σ . In addition, we write $\sigma \in (\Gamma)_R$ if for all $x : A \in \Gamma$, $x\langle\sigma\rangle \Vdash A$.

Finally, the realizability semantics defines the typing rules as follows: $\Gamma \vdash \vec{t} : A$ is a notation for $\forall \sigma \in (\Gamma)_R, \vec{t}\langle\sigma\rangle \Vdash A$. Hence, we need to prove that the typing system presented in Table 6 is correct, which is done by Theorem 3.4.

Theorem 3.4 (Correctness). *If $\Gamma \vdash \vec{t} : A$, then for all $\sigma \in (\Gamma)_R$, we have $\vec{t}\langle\sigma\rangle \Vdash A$.*

Proof. We only prove the judgements that are not already proved in [DCGMV19]. We proceed by induction on the typing derivation.

- **Rule Lam.** By the induction hypothesis, $\forall(\sigma, x := \vec{w}) \in (\Gamma, x : A)_R, \vec{t}\langle\sigma, x := \vec{w}\rangle \Vdash B$. Therefore, by definition, $(\lambda x. \vec{t})\langle\sigma\rangle = \lambda x. \vec{t}\langle\sigma\rangle \in (A \rightarrow B)_R$. So, $(\lambda x. \vec{t})\langle\sigma\rangle \Vdash A \rightarrow B$.
- **Rule App.** By the induction hypothesis, $\forall \sigma \in (\Gamma)_R, t\langle\sigma\rangle \Vdash A \rightarrow B$ and $\forall \theta \in (\Delta)_R, \vec{s}\langle\theta\rangle \Vdash A$. Then, by definition, $t\langle\sigma\rangle \rightarrow^* \lambda x. \vec{r}$ and $\vec{s}\langle\theta\rangle \rightarrow^* \vec{v} \in (A)_R$ such that $\vec{r}\langle x := \vec{v} \rangle \Vdash B$. Since $(t\vec{s})\langle\sigma, \theta\rangle \rightarrow^* \vec{r}\langle x := \vec{v} \rangle$, we have, $(t\vec{s})\langle\sigma, \theta\rangle \Vdash B$.
- **Rule Sup.** For all j , by the induction hypothesis, we have that $\vec{v}_j \in (A)_R$. Since for all $k \neq h$ we have $\vec{v}_k \perp \vec{v}_h$, and $\sum_j |\alpha_j|^2 = 1$, we have $\sum_j \alpha_j \cdot \vec{v}_j \in \mathcal{S}_1$, and so $\sum_j \alpha_j \cdot \vec{v}_j \in (\sharp A)_R$. Hence, $\sum_j \alpha_j \cdot \vec{v}_j \Vdash \sharp A$. \square

The following corollary is a direct consequence of Theorems 3.1 and 3.4.

Corollary 3.5 (Strong normalization). *If $\Gamma \vdash \vec{t} : A$ then \vec{t} is strongly normalizing.* \square

³Notice, however, that subject reduction is not implied by the correctness with respect to the realizability semantics, since it may be the case that a term \vec{t} reduces to a term \vec{t}' , both in the semantics, but the second not typable with the typing rules chosen. This is why we have given a direct syntactic proof of such a property (Section 3.2).

The realizability model also implies that $\vec{V} \subseteq \mathcal{S}_1$.

3.4. Expressivity. First we define the following encoding of norm-1 vectors from \mathbb{C}^{2^n} to terms in Lambda-S₁:

Definition 3.6. Let $b_i \in \{0, 1\}$ and $|\underline{k}\rangle = |b_0 \dots b_{2^n-1}\rangle$, where $b_0 \dots b_{2^n-1}$ is the binary representation of $k \in \mathbb{N}$. Then, we encode $|\underline{k}\rangle$ in Lambda-S₁ as $\hat{k} = (\hat{b}_0, (\hat{b}_2, (\dots, \hat{b}_{2^n-1})))$, with $\hat{0} = \text{inl}(\ast)$ and $\hat{1} = \text{inr}(\ast)$. Thus, if $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$ and $\vec{v} = (\alpha_0, \dots, \alpha_{2^n-1})^T \in \mathbb{C}^{2^n}$, its encoding in Lambda-S₁ is $\hat{v} = \sum_{k=0}^{2^n-1} \alpha_k \cdot \hat{k}$.

From now on, we may write \mathbb{B} for $\mathbb{U} + \mathbb{U}$, A^n for $\prod_{i=1}^n A = A \times A \times \dots \times A$, and $A^{\otimes n}$ for $\sharp A^n = \sharp(A \times A \times \dots \times A)$.

Example 3.7. $(\frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}}, 0, 0, 0, 0)^T = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|011\rangle = \frac{1}{\sqrt{2}}|\underline{0}\rangle + \frac{1}{\sqrt{2}}|\underline{3}\rangle$ in \mathbb{C}^{2^3} is encoded as

$$\frac{1}{\sqrt{2}} \cdot (\text{inl}(\ast), (\text{inl}(\ast), \text{inl}(\ast))) + \frac{1}{\sqrt{2}} \cdot (\text{inl}(\ast), (\text{inr}(\ast), \text{inr}(\ast)))$$

The previous construction is typable as shown in the following example.

Example 3.8. Let $\vec{v} = (\alpha_0, \alpha_1, \dots, \alpha_7)^T = \sum_{k=0}^7 \alpha_k |\underline{k}\rangle \in \mathbb{C}^{2^3}$, with $\sum_{k=0}^7 |\alpha_k|^2 = 1$. Hence, $\hat{v} = \sum_{k=0}^7 \alpha_k \cdot \hat{k}$, using the encoding from Definition 3.6.

We check that $\vdash \hat{v} : \mathbb{B} \otimes \mathbb{B} \otimes \mathbb{B}$. We have

$$\frac{\frac{\frac{\text{Void}}{\vdash \ast : \mathbb{U}}}{\vdash \text{inl}(\ast) : \mathbb{B}} \text{InL} \quad \frac{\frac{\text{Void}}{\vdash \ast : \mathbb{U}}}{\vdash \text{inl}(\ast) : \mathbb{B}} \text{InL} \quad \frac{\text{Void}}{\vdash \text{inl}(\ast) : \mathbb{B}} \text{InL}}{\vdash (\text{inl}(\ast), \text{inl}(\ast)) : \mathbb{B} \times \mathbb{B}} \text{Pair}}{\vdash \hat{0} : \mathbb{B} \times \mathbb{B} \times \mathbb{B}} \text{Pair}$$

Similarly, we derive $\vdash \hat{1} : \mathbb{B} \times \mathbb{B} \times \mathbb{B}, \dots, \vdash \hat{7} : \mathbb{B} \times \mathbb{B} \times \mathbb{B}$. Therefore,

$$\frac{(j \neq k) \quad \vdash (\hat{j} \perp \hat{k}) : \mathbb{B} \times \mathbb{B} \times \mathbb{B} \quad \sum_{i=0}^7 |\alpha_i|^2 = 1}{\vdash \hat{v} : \mathbb{B} \otimes \mathbb{B} \otimes \mathbb{B}} \text{Sup}$$

We can define a case construction for the elements of $\mathbb{B}^{\otimes n}$, noted as **case** s of $\{\hat{k} \mapsto \vec{v}_k\}$, as shown in the following example.

Example 3.9. Let $\lambda z.\text{case } z \text{ of } \{\hat{0} \mapsto \vec{v}_0 \mid \hat{1} \mapsto \vec{v}_1 \mid \hat{2} \mapsto \vec{v}_2 \mid \hat{3} \mapsto \vec{v}_3\}$ be defined as

$$\lambda z.\text{let } (x, y) = z \text{ in match } x \{ \text{inl}(x') \mapsto x'; \text{match } y \{ \text{inl}(y') \mapsto y'; \vec{v}_0 \mid \text{inr}(y') \mapsto y'; \vec{v}_1 \} \\ \text{inr}(x') \mapsto x'; \text{match } y \{ \text{inl}(y') \mapsto y'; \vec{v}_2 \mid \text{inr}(y') \mapsto y'; \vec{v}_3 \} \}$$

Assuming \vec{v}_i are orthogonal two by two, this term can be typed with $(\mathbb{B} \otimes \mathbb{B}) \rightarrow (\mathbb{B} \otimes \mathbb{B})$. Extending this construction to any n is an easy exercise.

Example 3.10. We can use the case construction from Example 3.9 to construct any quantum operator. For example, the CNOT operator corresponds to the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

which sends $|0x\rangle$ to $|0x\rangle$, $|10\rangle$ to $|11\rangle$, and $|11\rangle$ to $|10\rangle$. This operator can be written in Lambda- \mathcal{S}_1 as

$$\lambda z.\text{case } z \text{ of } \{\hat{0} \mapsto \hat{0} \mid \hat{1} \mapsto \hat{1} \mid \hat{2} \mapsto \hat{3} \mid \hat{3} \mapsto \hat{2}\}$$

Any quantum operator can be written as a matrix, and this encoding provides exactly that.

The expressivity of the language is stated as follows.

Theorem 3.11 (Expressivity).

- (1) *Simply Typed Lambda Calculus extended with pairs and sums, in call-by-value, is included in Lambda- \mathcal{S}_1 .*
- (2) *If U is an isometry acting on \mathbb{C}^{2^n} , then there exists a lambda term \hat{U} such that $\vdash \hat{U} : \mathbb{B}^{\otimes n} \rightarrow \mathbb{B}^{\otimes n}$ and for all $\vec{v} \in \mathbb{C}^{2^n}$, if $\vec{w} = U\vec{v}$, we have $\hat{U}\hat{\vec{v}} \longrightarrow^* \hat{\vec{w}}$.*

Proof.

- (1) The terms from the lambda calculus with pairs and sums are included in the grammar of pure terms. The set of types $\{A \mid A^b\}$ includes the simply types. Simply types, together with rules Ax, Lam, App, Pair, PureLet, InL, InR, PureMatch, Weak, and Cont allow to type the lambda calculus extended with pairs and sums.
- (2) Let U be an isometry. We can define it by giving its behaviour on a base of \mathbb{C}^{2^n} . So, consider the canonical base $B = \{|0\rangle, \dots, |2^n - 1\rangle\}$ and for all $|k\rangle \in B$, let $U|k\rangle = (\beta_{0,k}, \dots, \beta_{2^n-1,k})^T$. That is, $U = (\beta_{ij})_{ij}$. We can define a term \hat{U} using the case construction introduced before (cf. Example 3.9).

$$\hat{U} = \lambda x.\text{case } x \text{ of } \{\hat{k} \mapsto \sum_{i=0}^{2^n-1} \beta_{ik} \cdot \hat{i}\}$$

We have $\vdash \hat{U} : \mathbb{B}^{\otimes n} \rightarrow \mathbb{B}^{\otimes n}$ (cf. Example 3.9).

Let $\vec{v} = (\alpha_0, \dots, \alpha_{2^n-1})^T = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle \in \mathbb{C}^{2^n}$, with $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. Then, we have $\hat{\vec{v}} = \sum_{k=0}^{2^n-1} \alpha_k \cdot \hat{k}$, with $\vdash \hat{\vec{v}} : \mathbb{B}^{\otimes n}$ (cf. Example 3.8), therefore, $\vdash \hat{U}\hat{\vec{v}} : \mathbb{B}^{\otimes n}$.

Notice that,

$$\begin{aligned} \hat{U}\hat{\vec{v}} &\equiv \sum_{k=0}^{2^n-1} \alpha_k \cdot \hat{U}\hat{k} \longrightarrow^* \sum_{k=0}^{2^n-1} \alpha_k \cdot \sum_{i=0}^{2^n-1} \beta_{ik} \cdot \hat{i} \\ &\equiv \sum_{i=0}^{2^n-1} \sum_{k=0}^{2^n-1} \alpha_k \beta_{ik} \cdot \hat{i} = \widehat{U\vec{v}} \quad \square \end{aligned}$$

Since Lambda- \mathcal{S}_1 is a fragment of the calculus that can be defined using the realizability model, and the previous lemma shows that any isometry can be represented in it, then the following theorem is still valid.

Theorem 3.12 ([DCGMV19, Theorem IV.12]). *A closed λ -abstraction $\lambda x.t$ is a value of type $\sharp\mathbb{B} \rightarrow \sharp\mathbb{B}$ if and only if it represents an isometry $U : \mathbb{C}^2 \rightarrow \mathbb{C}^2$. \square*

Extending this result to a bigger dimension is straightforward.

Remark 3.13. Even if we can check orthogonality on open terms, e.g. $\Gamma \vdash (\Delta_1 \vdash \vec{v}_1 \perp \Delta_2 \vdash \vec{v}_2) : A$, we cannot type a constructor of an oracle (as, for example, the oracle needed for the Deutsch's algorithm, cf. [NC10, §1.4.3]) parametrized by a given function f . That is,

the oracle U_f sending $|b_1 b_2\rangle$ to $|b_1, b_2 \oplus f(b_1)\rangle$ can be typed with $\mathbb{B}^{\otimes 2} \rightarrow \mathbb{B}^{\otimes 2}$ for any given f in our language, however, we cannot type a term $\lambda f.U_f$ such as

$$\begin{aligned} \lambda f.U_f := \lambda f.\lambda x.\text{case } x \text{ of } \{ & \hat{0} \mapsto (\text{inl}(*), f \text{inl}(*)), \\ & \hat{1} \mapsto (\text{inl}(*), \text{not } (f \text{inl}(*))), \\ & \hat{2} \mapsto (\text{inr}(*), f \text{inr}(*)), \\ & \hat{3} \mapsto (\text{inr}(*), \text{not } (f \text{inr}(*))) \} \end{aligned}$$

Indeed, the branches of the case are not values and so the orthogonality cannot be verified.

In [DCGMV19] this term $\lambda f.U_f$ is valid, with typing $\vdash \lambda f.U_f : (\mathbb{B} \rightarrow \mathbb{B}) \rightarrow \mathbb{B}^{\otimes 2} \rightarrow \mathbb{B}^{\otimes 2}$. Certainly, the orthogonality verification is done by the realizability model, by considering all the reductions of the application of the term to any possible argument, something that is not desirable in a static type system, where reducing a term in order to type it is not a good practice. In any case, the Deutsch's algorithm, or any other quantum algorithm using such kind of oracle, does not construct the gate dynamically with a term of the kind $\lambda f.U_f$, but the unitary U_f is given.

4. DENOTATIONAL SEMANTICS

We define two categories, $\text{Set}_{\vec{V}}$ and $\text{SVec}_{\vec{V}}$, and an adjunction between them. Types are interpreted as objects in $\text{Set}_{\vec{V}}$, and terms as maps using the adjunction.

As stated in Section 2, value distributions do not form a vector space. Here we make this concept more precise by defining the non-standard notion of distributive-action space⁴.

Definition 4.1 (Distributive-action space). A distributive-action space over a field K is a commutative semi-group⁵ $(V, +)$ equipped with a scalar multiplication $(\cdot) : K \times V \rightarrow V$ such that for all $\vec{v}, \vec{w} \in V$, $\alpha, \beta \in K$, we have the following axioms.

$$\begin{aligned} 1 \cdot \vec{v} &= \vec{v} & (\alpha + \beta) \cdot \vec{v} &= \alpha \cdot \vec{v} + \beta \cdot \vec{v} \\ \alpha \cdot (\beta \cdot \vec{v}) &= \alpha\beta \cdot \vec{v} & \alpha \cdot (\vec{v} + \vec{w}) &= \alpha \cdot \vec{v} + \alpha \cdot \vec{w} \end{aligned}$$

In analogy with vector spaces, the elements of V are called vectors and the elements of K scalars.

The notion of distributive-action space differs from the traditional notion of vector space in that the underlying additive structure $(V, +)$ is an arbitrary commutative semi-group, whose elements in general do not include a neutral element, and so do not have an additive inverse. In a distributive-action space, the vector $(-1) \cdot \vec{v}$ is in general not the additive inverse of \vec{v} , and the product $0 \cdot \vec{v}$ does not simplify to a neutral element $\vec{0}$. Indeed, term distributions do not have a null vector.

The notions of inner product and norm can be generalized to the case of distributive-action spaces in the following way.

Definition 4.2 (Inner product of a distributive-action space over \mathbb{C}). An inner product of a distributive-action space V is a function $\langle \cdot | \cdot \rangle : V \times V \rightarrow \mathbb{C}$ satisfying the following properties. For all $\alpha \in \mathbb{C}$, $\vec{u}, \vec{v}, \vec{w} \in V$,

⁴In [DCGMV19] the notion of weak vector space is defined, which is based on a commutative monoid. Here we use a commutative semi-group instead, since a null vector is not needed.

⁵That is, an associative and commutative magma.

- (1) $\langle \vec{u} \mid \vec{v} \rangle = \overline{\langle \vec{v} \mid \vec{u} \rangle}$, where $\bar{\alpha}$ is the conjugate of α .
- (2) $\langle \vec{u} \mid \alpha \cdot \vec{v} \rangle = \alpha \langle \vec{u} \mid \vec{v} \rangle$
 $\langle \vec{u} \mid \vec{v} + \vec{w} \rangle = \langle \vec{u} \mid \vec{v} \rangle + \langle \vec{u} \mid \vec{w} \rangle$.
- (3) $\langle \vec{u} \mid \vec{u} \rangle > 0$ for all $\vec{u} \in \{\vec{v} \in V \mid \forall \vec{w}, \vec{v} \neq 0 \cdot \vec{w}\}$.

Definition 4.3 (Norm of a distributive-action space). A norm of a distributive-action space V is a function $\|\cdot\| : V \rightarrow \mathbb{R}^+$ satisfying the following properties. For all $\alpha \in K$, $\vec{v}, \vec{w} \in V$,

- (1) $\|\vec{v} + \vec{w}\| \leq \|\vec{v}\| + \|\vec{w}\|$.
- (2) $\|\alpha \cdot \vec{v}\| = |\alpha| \|\vec{v}\|$.
- (3) $\|\vec{v}\| = 0$ if and only if $\vec{v} = 0 \cdot \vec{w}$, for some \vec{w} .

Theorem 4.4. \vec{V} is a normed distributive-action space over \mathbb{C} .

Proof. Verifying that \vec{V} is a distributive-action space over \mathbb{C} is straightforward by checking that the congruence given in Table 2 coincides with the requirements from Definition 4.1.

We also check that Definition 2.1 verifies the definition of an inner product (Definition 4.2).

Finally, we define the norm

$$\|\vec{v}\| := \sqrt{\langle \vec{v} \mid \vec{v} \rangle} = \sqrt{\sum_{i=1}^n |\alpha_i|^2}$$

and check that such a norm verifies Definition 4.3. □

We write $\mathcal{P}_*(\cdot) = \mathcal{P}(\cdot) \setminus \{\emptyset\}$, and Def to the set of (computable) functions that can be defined in $\text{Lambda-}\mathcal{S}_1$.

With all the previous definitions, we can define the categories which will give the adjunction to model the calculus.

Definition 4.5 (Category $\text{Set}_{\vec{V}}$). The monoidal category $\text{Set}_{\vec{V}}$ has the following elements:

- $\text{Ob}(\text{Set}_{\vec{V}}) = \mathcal{P}_*(\vec{V})$.
- For all $A, B \in \text{Ob}(\text{Set}_{\vec{V}})$,

$$\text{Hom}_{\text{Set}_{\vec{V}}}(A, B) = \{f \in \text{Def} \mid f : A \rightarrow B \in \text{Arr}(\text{Set})\}$$

- For all $A, B \in \text{Ob}(\text{Set}_{\vec{V}})$,

$$A \boxtimes B = \left\{ \sum_i \sum_j \alpha_i \beta_j \cdot (v_i, w_j) \mid \begin{array}{l} \sum_i \alpha_i \cdot v_i \in A \\ \sum_j \beta_j \cdot w_j \in B \end{array} \right\}$$

$$\stackrel{\text{def}}{=} \{(\vec{v}, \vec{w}) \mid \vec{v} \in A \text{ and } \vec{w} \in B\}$$

- $1 = \{*\} \in \text{Ob}(\text{Set}_{\vec{V}})$.
- The obvious structural maps: $A \xrightarrow{\lambda_{\boxtimes}} 1 \boxtimes A$, $A \xrightarrow{\rho_{\boxtimes}} A \boxtimes 1$, and $(A_1 \boxtimes A_2) \boxtimes A_3 \xrightarrow{\alpha_{\boxtimes}} A_1 \boxtimes (A_2 \boxtimes A_3)$.

Remark 4.6. Let $A \xrightarrow{f} B$ and $C \xrightarrow{g} D$ in $\text{Arr}(\text{Set}_{\vec{V}})$. Then $f, g \in \text{Def}$. So, let \hat{f} and \hat{g} be the terms in $\text{Lambda-}\mathcal{S}_1$ implementing f and g respectively. Therefore, a term implementing $f \boxtimes g$ is the following $\lambda x. \text{let } (y, z) = x \text{ in } (\hat{f}y, \hat{g}z)$.

Remark 4.7. Notice that while the category $\text{Set}_{\vec{V}}$ is generated by the syntax of the calculus, it is not a syntactic category in the sense of [LS86, Part I §10], where the objects are types and arrows are terms. Indeed, the objects at $\text{Set}_{\vec{V}}$ are non-empty powersets of values, and the interpretation of the language will be done by using an adjunction with a richer category $\text{SVec}_{\vec{V}}$ yet to be define (cf. Definition 4.9 and Proposition 4.11 establishing the adjunction). Hence, this construction is far from trivial.

In order to define the category $\text{SVec}_{\vec{V}}$, we define the set $\mathcal{H}_{\vec{V}}$ and the map S .

Definition 4.8.

- (1) $\mathcal{H}_{\vec{V}}$ is the set of all the sub-distributive action spaces⁶ of \vec{V} , with inner product induced by \vec{V} .
- (2) $S : \mathcal{P}_*(\vec{V}) \rightarrow \mathcal{H}_{\vec{V}}$ is a map defined by $A \mapsto \{\sum_i \alpha_i \cdot \vec{v}_i \mid \vec{v}_i \in A\}$.
- (3) Let $A, B \in \mathcal{P}_*(\vec{V})$ and $f : A \rightarrow B$ be a map. Then, $Sf : SA \rightarrow SB$ is the map defined by $\sum_i \alpha_i \cdot \vec{v}_i \mapsto \sum_i \alpha_i \cdot f\vec{v}_i$.

Definition 4.9 (Category $\text{SVec}_{\vec{V}}$). The monoidal category $\text{SVec}_{\vec{V}}$ has the following elements:

- $\text{Ob}(\text{SVec}_{\vec{V}}) = \mathcal{H}_{\vec{V}}$.
- For all $V, W \in \text{Ob}(\text{SVec}_{\vec{V}})$,

$$\text{Hom}_{\text{SVec}_{\vec{V}}}(V, W) = \{f \in \text{Def} \mid V \xrightarrow{f} W \text{ is a linear map}\}$$

- For all $V, W \in \text{Ob}(\text{SVec}_{\vec{V}})$,

$$V \otimes W = S(V \boxtimes W)$$

- $I = \{\alpha \cdot * \mid \alpha \in \mathbb{C}\} \in \text{Ob}(\text{SVec}_{\vec{V}})$.
- The obvious structural maps: $V \xrightarrow{\lambda_{\otimes}} I \otimes V$, $V \xrightarrow{\rho_{\otimes}} V \otimes I$, and $(V_1 \otimes V_2) \otimes V_3 \xrightarrow{\alpha_{\otimes}} V_1 \otimes (V_2 \otimes V_3)$.

Lemma 4.10. *The following maps are monoidal functors:*

- (1) $S : \text{Set}_{\vec{V}} \rightarrow \text{SVec}_{\vec{V}}$, defined as in Definition 4.8.
- (2) $U : \text{SVec}_{\vec{V}} \rightarrow \text{Set}_{\vec{V}}$, the forgetful functor.

Proof.

- (1) If $A \in \text{Ob}(\text{Set}_{\vec{V}})$, then, by definition $SA \in \text{Ob}(\text{SVec}_{\vec{V}})$.

Let $A \xrightarrow{f} B \in \text{Arr}(\text{Set}_{\vec{V}})$ and $\sum_{i=1}^n \alpha_i \cdot a_i, \sum_{j=1}^m \beta_j \cdot a'_j \in SA$. Then,

$$\begin{aligned} & (Sf)(\delta_1 \cdot \sum_{i=1}^n \alpha_i \cdot a_i + \delta_2 \cdot \sum_{j=1}^m \beta_j \cdot a'_j) \\ &= (Sf)(\sum_{i=1}^n \delta_1 \alpha_i \cdot a_i + \sum_{j=1}^m \delta_2 \beta_j \cdot a'_j) \\ &= \sum_{i=1}^n \delta_1 \alpha_i \cdot f a_i + \sum_{j=1}^m \delta_2 \beta_j \cdot f a'_j \\ &= \delta_1 \cdot \sum_{i=1}^n \alpha_i \cdot f a_i + \delta_2 \cdot \sum_{j=1}^m \beta_j \cdot f a'_j \\ &= \delta_1 \cdot (Sf)(\sum_{i=1}^n \alpha_i \cdot a_i) + \delta_2 \cdot (Sf)(\sum_{j=1}^m \beta_j \cdot a'_j) \end{aligned}$$

Therefore, $Sf : SA \rightarrow SB$ is lineal and since $f \in \text{Def}$, $Sf \in \text{Def}$. Thus $Sf \in \text{Arr}(\text{SVec}_{\vec{V}})$. Functoriality is fulfilled by being a span.

⁶ V is a sub-distributive action space of W if $V \subseteq W$ and V forms a distributive-action space with the operations from W restricted to V .

(2) If $V \in \text{Ob}(\text{SVec}_{\vec{V}}) \subseteq \mathcal{P}_*(\vec{V}) = \text{Ob}(\text{Set}_{\vec{V}})$, then, $UV = V \in \text{Ob}(\text{Set}_{\vec{V}})$. Let $f \in \text{Arr}(\text{SVec}_{\vec{V}})$, then f is a linear map in Def , so it is a map and then $f \in \text{Arr}(\text{Set}_{\vec{V}})$.

Finally, we prove the functors to be monoidal by proving the existence of the maps

$$SA \otimes SB \xrightarrow{m_{AB}} S(A \boxtimes B) \quad \text{and} \quad I \xrightarrow{m_1} S1$$

in $\text{Arr}(\text{SVec}_{\vec{V}})$, and

$$UV \boxtimes UW \xrightarrow{n_{VW}} U(V \otimes W) \quad \text{and} \quad 1 \xrightarrow{n_I} UI$$

in $\text{Arr}(\text{Set}_{\vec{V}})$, trivially satisfying some well-known axioms.

- Since $SA \otimes SB = S(SA \boxtimes SB) = S(A \boxtimes B)$, we take $m_{AB} = \text{Id}$.
- Since $I = S1$, we take $m_1 = \text{Id}$.
- Since $UV \boxtimes UW \subseteq US(UV \boxtimes UW) = U(V \otimes W)$, we take n_{VW} as the inclusion map.
- Since $1 \subseteq UI$, we take n_I as the inclusion map. \square

We can now establish the adjunction between the two categories, which will give us the framework to interpret the calculus.

Proposition 4.11. *The following construction is a monoidal adjunction:*

$$\begin{array}{ccc} & \xrightarrow{(S,m)} & \\ (\text{Set}_{\vec{V}}, \boxtimes, 1) & \perp & (\text{SVec}_{\vec{V}}, \otimes, I) \\ & \xleftarrow{(U,n)} & \end{array}$$

where m and n are mediating arrows of the monoidal structure.

Proof. We need to prove that $\text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V) \simeq \text{Hom}_{\text{Set}_{\vec{V}}}(A, UV)$.

- Let $\text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V) \xrightarrow{\varphi_{A,V}} \text{Hom}_{\text{Set}_{\vec{V}}}(A, UV)$ given by $h \mapsto h \upharpoonright A$.
- Let $\text{Hom}_{\text{Set}_{\vec{V}}}(A, UV) \xrightarrow{\psi_{A,V}} \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V)$ given by $s \mapsto Ss$.

Notice that $\varphi_{A,V} \circ \psi_{A,V} = \text{id}$ and $\psi_{A,V} \circ \varphi_{A,V} = \text{id}$. Therefore, we rename $\varphi_{A,V}^{-1} = \psi_{A,V}$.

We must prove that if $B \xrightarrow{f} A$, $V \xrightarrow{g} W$ the following diagrams commute:

$$\begin{array}{ccc} \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V) - \varphi_{A,V} \rightarrow \text{Hom}_{\text{Set}_{\vec{V}}}(A, UV) & \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V) - \varphi_{A,V} \rightarrow \text{Hom}_{\text{Set}_{\vec{V}}}(A, UV) \\ \downarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(Sf, V) & \downarrow \text{Hom}_{\text{Set}_{\vec{V}}}(f, UV) & \downarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, g) & \downarrow \text{Hom}_{\text{Set}_{\vec{V}}}(A, Ug) \\ \text{Hom}_{\text{SVec}_{\vec{V}}}(SB, V) - \varphi_{B,V} \rightarrow \text{Hom}_{\text{Set}_{\vec{V}}}(B, UV) & \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, W) - \varphi_{A,W} \rightarrow \text{Hom}_{\text{Set}_{\vec{V}}}(A, iW) \end{array}$$

$$\begin{array}{ccc} \text{Hom}_{\text{Set}_{\vec{V}}}(A, UV) - \varphi_{A,V}^{-1} \rightarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V) & \text{Hom}_{\text{Set}_{\vec{V}}}(A, UV) - \varphi_{A,V}^{-1} \rightarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, V) \\ \downarrow \text{Hom}_{\text{Set}_{\vec{V}}}(f, UV) & \downarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(Sf, V) & \downarrow \text{Hom}_{\text{Set}_{\vec{V}}}(A, Ug) & \downarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, g) \\ \text{Hom}_{\text{Set}_{\vec{V}}}(B, UV) - \varphi_{B,V}^{-1} \rightarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(SB, V) & \text{Hom}_{\text{Set}_{\vec{V}}}(A, iW) - \varphi_{A,W}^{-1} \rightarrow \text{Hom}_{\text{SVec}_{\vec{V}}}(SA, W) \end{array}$$

- In the first diagram we need to prove that $(h \upharpoonright A) \circ f = (h \circ Sf) \upharpoonright B$.
We have, for any $b \in B$,

$$\begin{aligned} ((h \upharpoonright A) \circ f)(b) &= h(f(b)) \\ &= h(Sf(b)) \\ &= ((h \circ Sf) \upharpoonright B)(b) \end{aligned}$$

- In the second diagram we need to prove that $Ug \circ (h \upharpoonright A) = (g \circ h) \upharpoonright A$.
We have, for any $a \in A$,

$$\begin{aligned} (Ug \circ (h \upharpoonright A))(a) &= g(h(a)) \\ &= ((g \circ h) \upharpoonright A)(a) \end{aligned}$$

- The third diagram follows by considering $S(s \circ f) = Ss \circ Sf$.
- The last one follows by $S(Ug \circ s) = S(Ug) \circ S(s) = g \circ S(s)$.

Finally, the monoidality axioms of the adjunction are trivially satisfied. \square

Before giving an interpretation of types, we need to define two particular objects $A + B$ and $[A, B]$ in $\text{Set}_{\vec{V}}$ to interpret the types $A + B$ and $A \rightarrow B$ respectively:

Definition 4.12. Let $A, B \in \text{Ob}(\text{Set}_{\vec{V}})$, we define the following objects.

$$\begin{aligned} A + B &= \{\text{inl}(\vec{v}) \mid \vec{v} \in A\} \cup \{\text{inr}(\vec{w}) \mid \vec{w} \in B\} \subseteq \vec{V} \\ [A, B] &= \{\hat{f} \mid f : A \rightarrow B \in \text{Arr}(\text{Set}_{\vec{V}})\} \subseteq \vec{V} \end{aligned}$$

where \hat{f} is the term in Lambda- \mathcal{S}_1 representing the map f .

In particular, we need to show that $A + B$ is actually a coproduct, as stated by the following lemma.

Lemma 4.13. $A + B$ is a coproduct. That is, given $A \xrightarrow{f} C$ and $B \xrightarrow{g} C$, there is a unique map $[f, g]$ such that the following diagram commutes.

$$\begin{array}{ccccc} A & \xrightarrow{i_1} & (A + B) & \xleftarrow{i_2} & B \\ & \searrow f & \downarrow [f, g] & \swarrow g & \\ & & C & & \end{array}$$

Proof. Since $A + B \in \text{Ob}(\text{Set}_{\vec{V}}) \subset \text{Ob}(\text{Set})$ and $f, g \in \text{Arr}(\text{Set}_{\vec{V}}) \subset \text{Arr}(\text{Set})$, we can take $[f, g] \in \text{Arr}(\text{Set})$ defined by

$$\begin{aligned} [f, g] : A + B &\rightarrow C \\ x &\mapsto \begin{cases} f(\vec{a}) & \text{if } x = \text{inl}(\vec{a}) \\ g(\vec{b}) & \text{if } x = \text{inr}(\vec{b}) \end{cases} \end{aligned}$$

and prove that $[f, g] \in \text{Arr}(\text{Set}_{\vec{V}})$. All we need to prove is that $[f, g] \in \text{Def}$. Take $[f, g] = \lambda x. \text{match } x \{ \text{inl}(\vec{a}) \mapsto \hat{f}(\vec{a}) \mid \text{inr}(\vec{b}) \mapsto \hat{g}(\vec{b}) \}$. \square

The following lemma allows us to use the home $[A, B]$ in the expected way.

Lemma 4.14. There is an adjunction $-\boxtimes B \dashv [B, -]$. That is

$$\text{Hom}_{\text{Set}_{\vec{V}}}(A \boxtimes B, C) \simeq \text{Hom}_{\text{Set}_{\vec{V}}}(A, [B, C])$$

Proof.

- Let $A \boxtimes B \xrightarrow{f} C \in \text{Arr}(\text{Set}_{\vec{\vee}})$. Then take $A \xrightarrow{\text{curry}f} [B, C]$ defined by $\vec{a} \mapsto \lambda x. \hat{f}(\vec{a}, x)$, which can be represented in $\text{Lambda-}\mathcal{S}_1$ as $\lambda y. \lambda x. \hat{f}(y, x)$.
- Let $A \xrightarrow{g} [B, C] \in \text{Arr}(\text{Set}_{\vec{\vee}})$. Then take $A \boxtimes B \xrightarrow{\text{uncurry}g} C$ defined by $(\vec{a}, \vec{b}) \mapsto (g\vec{a})\vec{b}$, with $\text{uncurry}g$ being $\lambda x. \text{let } (y, z) = x \text{ in } \hat{g}yz$.

Notice that $\text{uncurry}(\hat{\text{curry}}f)(a, b) \rightarrow^* \hat{f}(a, b)$ and $\text{curry}(\hat{\text{uncurry}}g)a \rightarrow^* \hat{g}a$. Finally, naturality follows from the fact that $\text{Set}_{\vec{\vee}}$ is a subcategory of Set . \square

Definition 4.15. Types are interpreted in the category $\text{Set}_{\vec{\vee}}$, as follows:

$$\begin{aligned} \llbracket \mathbb{U} \rrbracket &= 1 \\ \llbracket \sharp A \rrbracket &= US[A] \\ \llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket \\ \llbracket A \times B \rrbracket &= \llbracket A \rrbracket \boxtimes \llbracket B \rrbracket \\ \llbracket A \rightarrow B \rrbracket &= \llbracket \llbracket A \rrbracket, \llbracket B \rrbracket \rrbracket \end{aligned}$$

To avoid cumbersome notation, we write A for $\llbracket A \rrbracket$, when there is no ambiguity.

Before giving the interpretation of typing derivation trees in the model, we need to define certain maps that will serve to implement some of the constructions in the language.

To interpret the *match* construction we define the following map.

Definition 4.16. Let $A \boxtimes \Delta \xrightarrow{f} C \in \text{Arr}(\text{Set}_{\vec{\vee}})$ and $B \boxtimes \Delta \xrightarrow{g} C \in \text{Arr}(\text{Set}_{\vec{\vee}})$. Then, we define the arrow $(A + B) \boxtimes \Delta \xrightarrow{[f, g]_1} C$ by $(\text{inl}(a), d) \mapsto f(a, d)$ and $(\text{inr}(b), d) \mapsto g(b, d)$.

To interpret the sequence construction, and the rules *Weak* and *Contr* we need the following maps.

Lemma 4.17. *Let A^b . Then,*

- (1) $A \boxtimes B \xrightarrow{\pi_B} B \in \text{Arr}(\text{Set}_{\vec{\vee}})$ and $B \boxtimes A \xrightarrow{\pi'_B} B \in \text{Arr}(\text{Set}_{\vec{\vee}})$.
- (2) $A \xrightarrow{\hat{\delta}} A \boxtimes A \in \text{Arr}(\text{Set}_{\vec{\vee}})$.

Proof.

- (1) Take $\hat{\pi}_B := \lambda x. \text{let } (y, z) = x \text{ in } z$, which can be typed as follows

$$\frac{\frac{\frac{}{x : A \times B \vdash x : A \times B} \text{Ax}}{x : A \times B \vdash \text{let } (y, z) = x \text{ in } z : B} \text{PureLet}}{\vdash \lambda x. \text{let } (y, z) = x \text{ in } z : A \times B \rightarrow B} \text{Lam}}{\frac{\frac{\frac{}{z : B \vdash z : B} \text{Ax}}{y : A, z : B \vdash z : B} \text{Pair}}{z : B \vdash z : B} \text{Weak}}{y : A, z : B \vdash z : B} \text{PureLet}}{y : A, z : B \vdash z : B} \text{Weak}}{y : A, z : B \vdash z : B} \text{PureLet}}{x : A \times B \vdash \text{let } (y, z) = x \text{ in } z : B} \text{PureLet}}{\vdash \lambda x. \text{let } (y, z) = x \text{ in } z : A \times B \rightarrow B} \text{Lam}$$

$\hat{\pi}'_B$ is analogous.

- (2) Take $\hat{\delta} := \lambda x. (x, x)$, which can be typed as follows

$$\frac{\frac{\frac{\frac{}{x : A \vdash x : A} \text{Ax}}{x : A, y : A \vdash (x, y) : A \times A} \text{Pair}}{x : A \vdash (x, x) : A \times A} \text{Contr}}{\vdash \lambda x. (x, x) : A \rightarrow A \times A} \text{Lam}}{\vdash \lambda x. (x, x) : A \rightarrow A \times A} \text{Lam}$$

\square

We give the interpretation of a type derivation tree in our model. If $\Gamma \vdash t : A$ with a derivation π , we write generically $\llbracket \pi \rrbracket$ as $\Gamma \xrightarrow{t_A} A$. When A is clear from the context, we may write just t for t_A .

Definition 4.18. If π is a type derivation tree, we define $\llbracket \pi \rrbracket$ inductively as follows,

$$\begin{aligned}
\llbracket \overline{x : A \vdash x : A} \text{ Ax} \rrbracket &= A \xrightarrow{\text{Id}} A \\
\llbracket \frac{\Gamma, x : A \vdash \vec{t} : B}{\Gamma \vdash \lambda x. \vec{t} : A \rightarrow B} \text{ Lam} \rrbracket &= \Gamma \xrightarrow{\eta^A} [A, \Gamma \boxtimes A] \xrightarrow{[A, \vec{t}]} [A, B] \\
\llbracket \frac{\Gamma \vdash t : A \rightarrow B \quad \Delta \vdash \vec{s} : A}{\Gamma, \Delta \vdash t \vec{s} : B} \text{ App} \rrbracket &= \Gamma \boxtimes \Delta \xrightarrow{t \boxtimes \vec{s}} [A, B] \boxtimes A \xrightarrow{\varepsilon'} B \quad \text{where } \varepsilon' = \varepsilon \circ \text{swap} \\
\llbracket \overline{\vdash * : \mathbb{U}} \text{ Void} \rrbracket &= 1 \xrightarrow{\text{Id}} 1 \\
\llbracket \frac{\Gamma \vdash \vec{t} : \mathbb{U} \quad \Delta \vdash \vec{s} : A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : A} \text{ PureSeq} \rrbracket &= \Gamma \boxtimes \Delta \xrightarrow{\vec{t} \boxtimes \vec{s}} 1 \boxtimes A \xrightarrow{\pi_A} A \\
\llbracket \frac{\Gamma \vdash \vec{t} : \sharp \mathbb{U} \quad \Delta \vdash \vec{s} : \sharp A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : \sharp A} \text{ UnitarySeq} \rrbracket &= \Gamma \boxtimes \Delta \xrightarrow{\vec{t} \boxtimes \vec{s}} UI \boxtimes USA \xrightarrow{n} U(I \otimes SA) \xrightarrow{U\lambda_{\otimes}^{-1}} USA \\
\llbracket \frac{\Gamma \vdash \vec{v} : A \quad \Delta \vdash \vec{w} : B}{\Gamma, \Delta \vdash (\vec{v}, \vec{w}) : A \times B} \text{ Pair} \rrbracket &= \Gamma \boxtimes \Delta \xrightarrow{\vec{v} \boxtimes \vec{w}} A \boxtimes B \\
\llbracket \frac{\Gamma \vdash \vec{t} : A \times B \quad \Delta, x : A, y : B \vdash \vec{s} : C}{\Gamma, \Delta \vdash \text{let } (x, y) = \vec{t} \text{ in } \vec{s} : C} \text{ PureLet} \rrbracket \\
&= \Gamma \boxtimes \Delta \xrightarrow{\vec{t} \boxtimes \eta^{A \boxtimes B}} (A \boxtimes B) \boxtimes [A \boxtimes B, \Delta \boxtimes A \boxtimes B] \\
&\xrightarrow{\text{Id} \boxtimes [A \boxtimes B, \vec{s}]} (A \boxtimes B) \boxtimes [A \boxtimes B, C] \xrightarrow{\varepsilon} C \\
\llbracket \frac{\Gamma \vdash \vec{t} : A \otimes B \quad \Delta, x : \sharp A, y : \sharp B \vdash \vec{s} : \sharp C}{\Gamma, \Delta \vdash \text{let } (x, y) = \vec{t} \text{ in } \vec{s} : \sharp C} \text{ UnitaryLet} \rrbracket \\
&= \Gamma \boxtimes \Delta \xrightarrow{\vec{t} \boxtimes \eta^{USA \otimes USB}} US(A \boxtimes B) \boxtimes [USA \boxtimes USB, \Delta \boxtimes USA \boxtimes USB] \\
&\xrightarrow{\text{Id} \boxtimes [USA \boxtimes USB, \vec{s}]} US(A \boxtimes B) \boxtimes [USA \boxtimes USB, USC] \\
&\xrightarrow{\text{Id} \boxtimes \eta} US(A \boxtimes B) \boxtimes US[USA \boxtimes USB, USC] \\
&\xrightarrow{n} U(S(A \boxtimes B) \otimes S[USA \boxtimes USB, USC]) \\
&\xrightarrow{Um} US((A \boxtimes B) \boxtimes [USA \boxtimes USB, USC]) \\
&\xrightarrow{US((\eta \boxtimes \eta) \boxtimes \text{Id})} US((USA \boxtimes USB) \boxtimes [USA \boxtimes USB, USC]) \xrightarrow{US\varepsilon} USUSC \xrightarrow{\mu} USC \\
\llbracket \frac{\Gamma \vdash \vec{v} : A}{\Gamma \vdash \text{inl}(\vec{v}) : A + B} \text{ InL} \rrbracket &= \Gamma \xrightarrow{\vec{v}} A \xrightarrow{i_1} A + B \\
\llbracket \frac{\Gamma \vdash \vec{v} : B}{\Gamma \vdash \text{inr}(\vec{v}) : A + B} \text{ InR} \rrbracket &= \Gamma \xrightarrow{\vec{v}} B \xrightarrow{i_2} A + B \\
\llbracket \frac{\Gamma \vdash \vec{t} : A + B \quad \Delta \vdash (x_1 : A \vdash \vec{v}_1 \perp x_2 : B \vdash \vec{v}_2) : C}{\Gamma, \Delta \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : C} \text{ PureMatch} \rrbracket
\end{aligned}$$

$$\begin{aligned}
&= \Gamma \boxtimes \Delta \xrightarrow{\text{id}} (A + B) \boxtimes \Delta \xrightarrow{[\vec{v}_1, \vec{v}_2]_1} C \\
&\left[\frac{\Gamma \vdash \vec{t} : A \oplus B \quad \Delta \vdash (x_1 : \sharp A \vdash \vec{v}_1 \perp x_2 : \sharp B \vdash \vec{v}_2) : \sharp C}{\Gamma, \Delta \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : \sharp C} \text{UnitaryMatch} \right] \\
&= \Gamma \boxtimes \Delta \xrightarrow{\text{id}} US(A + B) \boxtimes \Delta \xrightarrow{US(\eta + \eta) \boxtimes \eta} US(USA + USB) \boxtimes US\Delta \\
&\xrightarrow{\eta} U(S(USA + USB) \otimes S\Delta) \xrightarrow{U_m} US((USA + USB) \boxtimes \Delta) \xrightarrow{US[\vec{v}_1, \vec{v}_2]_1} USC \\
&\left[\frac{(k \neq h) \quad \vdash (\vec{v}_k \perp \vec{v}_h) : A \quad \sum_{j=1}^m |\alpha_j|^2 = 1 \quad m \geq 1 \quad A \neq B \rightarrow C}{\vdash \sum_{j=1}^m \alpha_j \cdot \vec{v}_j : \sharp A} \text{Sup} \right] \\
&= 1 \xrightarrow{\eta} US1 \xrightarrow{U_{\sum_j \alpha_j \cdot S\vec{v}_j}} USA \\
&[\overline{A \leq A}] = A \xrightarrow{\text{Id}} A \\
&\left[\frac{A \leq B \quad B \leq C}{A \leq C} \right] = A \xrightarrow{[A \leq B]} B \xrightarrow{[B \leq C]} C \\
&[\overline{A \leq \sharp A}] = A \xrightarrow{\eta} USA \\
&[\overline{\sharp \sharp A \leq \sharp A}] = USUSA \xrightarrow{\mu} USA \quad \text{where } \mu = U\varepsilon_S \\
&\left[\frac{A \leq A' \quad B \leq B'}{A' \rightarrow B \leq A \rightarrow B'} \right] = [A', B] \xrightarrow{[[A \leq A'], [B \leq B']]} [A, B'] \\
&\left[\frac{A \leq A' \quad B \leq B'}{A \times B \leq A' \times B'} \right] = A \boxtimes B \xrightarrow{[A \leq A'] \boxtimes [B \leq B']} A' \boxtimes B' \\
&\left[\frac{A \leq A' \quad B \leq B'}{A + B \leq A' + B'} \right] = A + B \xrightarrow{[A \leq A'] + [B \leq B']} A' + B' \\
&\left[\frac{\Gamma \vdash \vec{t} : A \quad A \leq B}{\Gamma \vdash \vec{t} : B} \leq \right] = \Gamma \xrightarrow{\vec{t}} A \xrightarrow{[A \leq B]} B \\
&\left[\frac{\Gamma \vdash \vec{t} : A \quad \vec{t} \equiv \vec{r}}{\Gamma \vdash \vec{r} : A} \equiv \right] = \Gamma \xrightarrow{\vec{t}} A \\
&\left[\frac{\Gamma \vdash \vec{t} : B \quad A^\flat}{\Gamma, x : A \vdash \vec{t} : B} \text{Weak} \right] = \Gamma \boxtimes A \xrightarrow{\pi'_\Gamma} \Gamma \xrightarrow{\vec{t}} B \\
&\left[\frac{\Gamma, x : A, y : A \vdash \vec{t} : B \quad A^\flat}{\Gamma, x : A \vdash \vec{t}[y := x] : B} \text{Contr} \right] = \Gamma \boxtimes A \xrightarrow{\text{Id} \boxtimes \delta} \Gamma \boxtimes A \boxtimes A \xrightarrow{\vec{t}} B
\end{aligned}$$

Lemma 4.19 allows us to write the semantics of a sequent, independently of its derivation. Hence, due to this independence, we can write $[\Gamma \vdash t : A]$, without ambiguity.

Lemma 4.19 (Independence of derivation). *If $\Gamma \vdash t : A$ can be derived with two different derivations π and π' , then $[\pi] = [\pi']$.*

Proof. We first give a rewrite system on derivation trees such that if one rule can be applied after or before another rule, we choose a direction to rewrite the tree to one of these forms. Then, we prove that every rule preserves the semantics of the tree. This rewrite system is clearly confluent and normalizing, hence for each tree π we can take the semantics of its normal form, and so every sequent will have one way to calculate its semantics: as the semantics of the normal tree.

The introduction rules (Ax, Lam, Void, Pair, InL, InR, and Sup) are syntax-directed. So, whenever we have an introduction term, we know precisely what is the last rule applied in its derivation tree.

For the structural rules (\leq , \equiv , Weak, and Contr) if they can be applied as last rule, or can be applied before, then we can always choose to rewrite the tree to apply them at the end, and also choose and order between them.

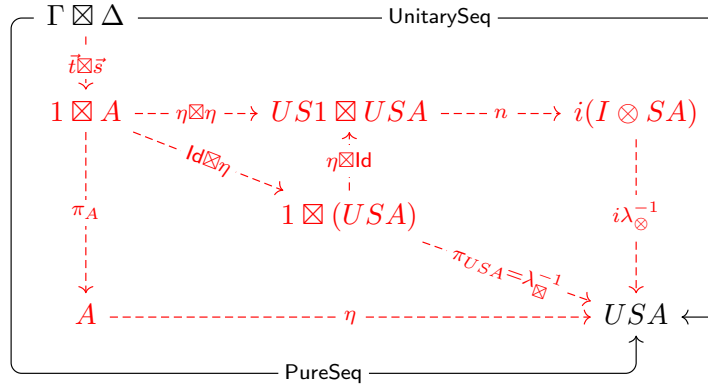
Finally, the elimination rule App is the only one which is syntax-directed, all the others are not, since for each elimination term there exists a rule preceded with Pure and another with Unitary. However, if a term can be typed with one, cannot be typed with the other, except for Seq, which, in combination with \leq , can be interchanged, and then we have to choose a direction to rewrite one to the other:

$$\frac{\frac{\Gamma \vdash \vec{t} : \mathbb{U} \leq \Delta \vdash \vec{s} : A \leq}{\Gamma \vdash \vec{t} : \sharp \mathbb{U}} \quad \Delta \vdash \vec{s} : \sharp A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : \sharp A} \text{UnitarySeq} \quad \longrightarrow \quad \frac{\Gamma \vdash \vec{t} : \mathbb{U} \quad \Delta \vdash \vec{s} : A}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : A} \text{PureSeq} \leq \frac{}{\Gamma, \Delta \vdash \vec{t}; \vec{s} : \sharp A}$$

The confluence of this rewrite system is easily inferred from the fact that there are not critical pairs. The normalization follows from the fact that the trees are finite and all the rewrite rules push the structural rules to the root of the trees.

It only remains to check that each rule preserves the semantics.

- The structural rules follow trivially by naturality.
- UnitarySeq \longrightarrow PureSeq The diagrams for the left-side and the right-side of the rewrite rule, commutes as it is shown below:



□

5. SOUNDNESS AND (PARTIAL) COMPLETENESS

We prove the soundness of our interpretation with respect to reduction, and the completeness only on type $\sharp(\mathbb{U} + \mathbb{U})$, which corresponds to \mathbb{C}^2 .

Lemma 5.1 (Substitution). *If $\Gamma, x : A \vdash \vec{t} : B$ and $\Delta \vdash v : A$, then the following diagram commutes:*

$$\begin{array}{ccc} \Gamma \boxtimes \Delta & \xrightarrow{\vec{t}[x:=v]} & B \\ & \searrow \text{Id} \boxtimes v & \nearrow \vec{t} \\ & \Gamma \boxtimes A & \end{array}$$

That is, $\llbracket \Gamma, \Delta \vdash \vec{t}[x := v] : B \rrbracket = \llbracket \Gamma, x : A, \Gamma \vdash \vec{t} : B \rrbracket \circ (\text{Id} \boxtimes \llbracket \Delta \vdash v : A \rrbracket)$.

Proof. By induction on the derivation of $\Gamma, x : A \vdash \vec{t} : B$. The details can be found in A \square

Theorem 5.2 (Soundness). *If $\Gamma \vdash t : A$, and $t \longrightarrow r$, then $\llbracket \Gamma \vdash t : A \rrbracket = \llbracket \Gamma \vdash r : A \rrbracket$.*

Proof. By induction on the rewrite relation. The details can be found in A. \square

Lemma 5.3 (Completeness of values on \mathbb{C}^2). *If $\llbracket \vdash \vec{v} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket = \llbracket \vdash \vec{w} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket$, then $\vec{v} \equiv \vec{w}$.*

Proof. Since \vec{v} and \vec{w} are values of type $\sharp(\mathbb{U} + \mathbb{U})$, they have the following shape: \vec{v} is \equiv -equivalent to either

- (1) $\alpha \cdot \mathbf{inl}(\ast) + \beta \cdot \mathbf{inr}(\ast)$,
- (2) $\mathbf{inl}(\ast)$, or
- (3) $\mathbf{inr}(\ast)$,

and \vec{w} is \equiv -equivalent to either

- (1) $\gamma \cdot \mathbf{inl}(\ast) + \delta \cdot \mathbf{inr}(\ast)$,
- (2) $\mathbf{inl}(\ast)$, or
- (3) $\mathbf{inr}(\ast)$.

Indeed, we have consider three cases since $1 \cdot \mathbf{inl}(\ast) + 0 \cdot \mathbf{inr}(\ast) \not\equiv \mathbf{inl}(\ast)$ because \vec{V} is a distributive-action space and not a vector space.

We analyse the different cases:

- If \vec{v} and \vec{w} are both in case 1, i.e. $\vec{v} \equiv \alpha \cdot \mathbf{inl}(\ast) + \beta \cdot \mathbf{inr}(\ast)$ and $\vec{w} \equiv \gamma \cdot \mathbf{inl}(\ast) + \delta \cdot \mathbf{inr}(\ast)$, we have

$$\begin{aligned} \llbracket \vdash \vec{v} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket &= 1 \xrightarrow{\eta} US1 \xrightarrow{U(\alpha \cdot Si_1 + \beta \cdot Si_2)} \sharp(\mathbb{U} + \mathbb{U}) \\ \llbracket \vdash \vec{w} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket &= 1 \xrightarrow{\eta} US1 \xrightarrow{U(\gamma \cdot Si_1 + \delta \cdot Si_2)} \sharp(\mathbb{U} + \mathbb{U}) \end{aligned}$$

So, the maps $\ast \mapsto \alpha \cdot \mathbf{inl}(\ast) + \beta \cdot \mathbf{inr}(\ast)$ and $\ast \mapsto \gamma \cdot \mathbf{inl}(\ast) + \delta \cdot \mathbf{inr}(\ast)$ are the same, and so, since $\mathbf{inl}(\ast) \perp \mathbf{inr}(\ast)$, we have $\alpha = \gamma$ and $\beta = \delta$, thus, $\vec{v} \equiv \vec{w}$.

- If \vec{v} and \vec{w} are both in case 2 or both in case 3, then $\vec{v} \equiv \vec{w}$.
- It is easy to see that \vec{v} and \vec{w} cannot be in different cases, since $\llbracket \vdash \vec{v} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket$ must be equal to $\llbracket \vdash \vec{w} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket$ and this is not the case when they are in different cases. For example, let $\vec{v} = \mathbf{inl}(\ast)$ and $\vec{w} = 1 \cdot \mathbf{inl}(\ast) + 0 \cdot \mathbf{inr}(\ast)$. In this case we have

$$\begin{aligned} \llbracket \vdash \vec{v} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket &= 1 \xrightarrow{i_1} 1 + 1 \xrightarrow{\eta} \sharp(\mathbb{U} + \mathbb{U}) \\ \llbracket \vdash \vec{w} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket &= 1 \xrightarrow{\eta} US1 \xrightarrow{U(1 \cdot Si_1 + 0 \cdot Si_2)} \sharp(\mathbb{U} + \mathbb{U}) \end{aligned}$$

the first arrow with the mapping $\ast \mapsto \mathbf{inl}(\ast)$ while the second $\ast \mapsto 1 \cdot \mathbf{inl}(\ast) + 0 \cdot \mathbf{inr}(\ast)$, which are not equal in a distributive-action space \square

Definition 5.4 (Equivalence on terms). We write $\vec{t} \sim \vec{r}$ whenever $\vec{t} \longrightarrow^* \vec{v}$ and $\vec{r} \longrightarrow^* \vec{w}$, with $\vec{v} \equiv \vec{w}$.

Theorem 5.5 (Completeness on \mathbb{C}^2). *If $\llbracket \vdash \vec{t} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket = \llbracket \vdash \vec{r} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket$, then $\vec{t} \sim \vec{r}$.*

Proof. By progress (Theorem 3.1) and strong normalization (Corollary 3.5), we have that $\vec{t} \longrightarrow^* \vec{v}$ and $\vec{r} \longrightarrow^* \vec{w}$. By subject reduction (Theorem 3.3), we have $\vdash \vec{v} : \sharp(\mathbb{U} + \mathbb{U})$ and $\vdash \vec{w} : \sharp(\mathbb{U} + \mathbb{U})$. Hence, by soundness (Theorem 5.2), $\llbracket \vdash \vec{v} : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket = \llbracket \vdash t : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket = \llbracket \vdash r : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket = \llbracket \vdash w : \sharp(\mathbb{U} + \mathbb{U}) \rrbracket$. Thus, by the completeness of values on \mathbb{C}^2 (Lemma 5.3), we have $\vec{v} \equiv \vec{w}$. So, by definition, $\vec{t} \sim \vec{r}$. \square

Theorem 5.6 (Completeness on qubits). *If $\llbracket \vdash \vec{t} : \mathbb{B}^{\otimes n} \rrbracket = \llbracket \vdash \vec{r} : \mathbb{B}^{\otimes n} \rrbracket$, then $\vec{t} \sim \vec{r}$.*

Proof. We prove it for $n = 2$. The generalization is straightforward. Since $\mathbb{B}^{\otimes 2} = \sharp(\mathbb{B} \times \mathbb{B}) = \sharp((\mathbb{U} + \mathbb{U}) \times (\mathbb{U} + \mathbb{U}))$, the set of closed values with this type is

$$Q_2 = \left\{ \sum_i \alpha_i \cdot (v_i, w_i) \mid v_i, w_i \in \{\mathbf{inl}(*), \mathbf{inr}(*)\} \right\}$$

Hence, if $\vec{v} \in Q_2$, \vec{v} is \equiv -equivalent to one of

$$\begin{aligned} & \alpha_0 \cdot v_0 + \alpha_1 \cdot v_1 + \alpha_2 \cdot v_2 + \alpha_3 \cdot v_3 \\ & \alpha_0 \cdot v_0 + \alpha_1 \cdot v_1 + \alpha_2 \cdot v_2 \\ & \alpha_0 \cdot v_0 + \alpha_1 \cdot v_1 \\ & \alpha_0 \cdot v_0 \end{aligned}$$

with $v_i \in \{(w_1, w_2) \mid w_1, w_2 \in \{\mathbf{inl}(*), \mathbf{inr}(*)\}\}$.

Following the same reasoning from Lemma 5.3, we get that if $\llbracket \vdash \vec{v} : \mathbb{B}^{\otimes 2} \rrbracket = \llbracket \vdash \vec{w} : \mathbb{B}^{\otimes 2} \rrbracket$, then $\vec{v} \equiv \vec{w}$, and following the same reasoning from Theorem 5.5, we have that if $\llbracket \vdash \vec{t} : \mathbb{B}^{\otimes 2} \rrbracket = \llbracket \vdash \vec{r} : \mathbb{B}^{\otimes 2} \rrbracket$, then $\vec{t} \sim \vec{r}$. \square

6. CONCLUSION

In this paper we have introduced Lambda- \mathcal{S}_1 , a quantum calculus issued from a realizability semantics [DCGMV19], and presented a categorical construction to interpret it.

Comparison with Lambda- \mathcal{S} . The main difference between Lambda- \mathcal{S}_1 and Lambda- \mathcal{S} [DCD17, DCDR19] is the fact that Lambda- \mathcal{S}_1 enforces norm 1 vectors by defining a distributive-action space on values. This gives us a related but different model than those for Lambda- \mathcal{S} [DCM19, DCM20b, DCM20a]. In addition, in Lambda- \mathcal{S}_1 it is allowed to type a superposition with a product type, whenever it is a separable state. Indeed, since $(\vec{v}, \vec{w}) := \sum_{i=1}^n \sum_{j=1}^m \alpha_i \beta_j \cdot (v_i, w_j)$. We have

$$\frac{\Gamma \vdash \vec{v} : A \quad \Delta \vdash \vec{w} : B}{\Gamma, \Delta \vdash (\vec{v}, \vec{w}) : A \times B}$$

That is,

$$\frac{\Gamma \vdash \sum_{i=1}^n v_i : A \quad \Delta \vdash \sum_{j=1}^m w_j : B}{\Gamma, \Delta \vdash \sum_{i=1}^n \sum_{j=1}^m \alpha_i \beta_j \cdot (v_i, w_j) : A \times B}$$

So, the type $A \times B$ is telling us that the term is separable, while a generic term of the form $\sum_{i=1}^n \sum_{j=1}^m \alpha_i \beta_j \cdot (v_i, w_j)$ would have type $\sharp(A \times B) = A \otimes B$. In Lambda- \mathcal{S} the only way to type such a term is with a tensor, and hence we need a casting operator in order to lose separability information when needed to allow reduction without losing type preservation.

The main property of Lambda- \mathcal{S}_1 is the fact that any isometry can be represented in the calculus (Theorem 3.11) and any term of type $\sharp(\mathbb{U} + \mathbb{U}) \rightarrow \sharp(\mathbb{U} + \mathbb{U})$ represents an isometry (Theorem 3.12).

Comparison with the full calculus. In its original presentation [DCGMV19], any arbitrary type A is defined by its semantics $(\cdot)_R$ as a set of values, and the notation $\vdash t : A$ means that t reduces to a value in $(A)_R$ (notation $t \Vdash A$). For example, let $|+\rangle = \frac{1}{\sqrt{2}} \cdot \text{inl}(\ast) + \frac{1}{\sqrt{2}} \cdot \text{inr}(\ast)$. We can consider $(A)_R = \{|+\rangle\} \subseteq (\sharp(\mathbb{U} + \mathbb{U}))_R = (\sharp\mathbb{B})_R$ (even if this is not a type we can construct with the given syntax of types).

The realizability semantics is so strong that it even allows defining a set $(A \Rightarrow B)_R$ of linear combinations of abstractions. However, not every linear combination of values in $(A \rightarrow B)_R$ is valid in the semantics. Indeed, if $\lambda x.\vec{t}$ and $\lambda x.\vec{s}$ are both in $(A \rightarrow B)_R$, the linear combination $\alpha \cdot \lambda x.\vec{t} + \beta \cdot \lambda x.\vec{s}$ with $|\alpha|^2 + |\beta|^2 = 1$ will be in $(A \Rightarrow B)_R$, if and only if $\alpha \cdot \vec{t}[v/x] + \beta \cdot \vec{s}[v/x]$ have norm 1 for any $v \in (A)_R$. Hence, $(A \Rightarrow B)_R$ is contained, but not equal to $(\sharp(A \rightarrow B))_R$.

The set $(A \Rightarrow B)_R$ can be easily constructed by the realizability semantics (since the typing $\vdash \vec{t} : C$ is done by reducing the term \vec{t} and checking that the resulted value is in $(C)_R$), but not with static methods. Therefore, we decided to exclude the type $A \Rightarrow B$ in $\text{Lambda-}\mathcal{S}_1$. Remark that even without superposition of abstractions, we do not lose expressivity, since $\alpha \cdot \lambda x.\vec{t} + \beta \cdot \lambda x.\vec{s}$ behaves in the same way that $\lambda x.(\alpha \cdot \vec{t} + \beta \cdot \vec{s})$, which can be typed in $\text{Lambda-}\mathcal{S}_1$.

REFERENCES

- [AD08] Pablo Arrighi and Gilles Dowek. Linear-algebraic λ -calculus: higher-order, encodings, and confluence. In Andrei Voronkov, editor, *Rewriting Techniques and Applications*, pages 17–31, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-70590-1_2.
- [AD17] Pablo Arrighi and Gilles Dowek. Lineal: A linear-algebraic λ -calculus. *Logical Methods in Computer Science*, 13(1:8):1–33, 2017. doi:10.23638/LMCS-13(1:8)2017.
- [ADCP⁺14] 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. doi:10.2168/LMCS-10(4:8)2014.
- [AG05] T. Altenkirch and J. Grattage. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 249–258, 2005. doi:10.1109/LICS.2005.1.
- [Bru14] Aloïs Brunel. *The monitoring power of forcing transformations*. PhD thesis, Université Paris 13, France, 2014.
- [BVN36] Garrett Birkhoff and John Von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37(4):823–843, 1936. doi:10.2307/1968621.
- [DCD17] Alejandro Díaz-Caro and Gilles Dowek. Typing quantum superpositions and measurement. In Carlos Martín-Vide, Roman Neruda, and Miguel A. Vega-Rodríguez, editors, *Theory and Practice of Natural Computing (TPNC 2017)*, volume 10687 of *Lecture Notes in Computer Science*, pages 281–293. Springer, Cham, 2017. doi:10.1007/978-3-319-71069-3_22.
- [DCDR19] Alejandro Díaz-Caro, Gilles Dowek, and Juan Pablo Rinaldi. Two linearities for quantum computing in the lambda calculus. *BioSystems*, 186:104012, 2019. Postproceedings of TPNC 2017. doi:10.1016/j.biosystems.2019.104012.
- [DCGMV19] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. Realizability in the unitary sphere. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019)*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785834.
- [DCM19] Alejandro Díaz-Caro and Octavio Malherbe. A concrete categorical semantics for Lambda-S . In Beniamino Accattoli and Carlos Olarte, editors, *Proceedings of the 13th Workshop on Logical and Semantic Frameworks with Applications (LSFA'18)*, volume 344 of *Electronic Notes in Theoretical Computer Science*, pages 83–100. Elsevier, 2019. doi:10.1016/j.entcs.2019.07.006.

- [DCM20a] Alejandro Díaz-Caro and Octavio Malherbe. A categorical construction for the computational definition of vector spaces. *Applied Categorical Structures*, 28(5):807–844, 2020. doi:10.1007/s10485-020-09598-7.
- [DCM20b] Alejandro Díaz-Caro and Octavio Malherbe. A concrete model for a typed linear algebraic lambda calculus. Draft at [arXiv:1806.09236](https://arxiv.org/abs/1806.09236), 2020.
- [GLR⁺13] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. *ACM SIGPLAN Notices (PLDI'13)*, 48(6):333–342, 2013. doi:10.1145/2499370.2462177.
- [Kle45] Stephen C. Kleene. On the interpretation of intuitionistic number theory. *The Journal of Symbolic Logic*, 10(4):109–124, 1945. doi:10.2307/2269016.
- [Kni96] Emanuel H. Knill. Conventions for quantum pseudocode. Technical Report LA-UR-96-2724, Los Alamos National Lab., 1996.
- [Kri09] Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses: Interactive models of computation and program behaviour*, 27:197–229, 2009.
- [LS86] J Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Cambridge studies in advances mathematics 7. Cambridge University Press, 1986.
- [Mel03] Paul-André Melliès. Categorical models of linear logic revisited. hal:00154229, 2003.
- [Miq11] Alexandre Miquel. A survey of classical realizability. In Luke Ong, editor, *Proceedings of TLCA-2011*, volume 6690 of *Lecture Notes in Computer Science*, pages 1–2, 2011. doi:10.1007/978-3-642-21691-6_1.
- [NC10] Michael Nielsen and Isaac Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2010.
- [PRZ17] Jennifer Paykin, Robert Rand, and Steve Zdancewic. Qwire: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 846–858, New York, NY, USA, 2017. ACM. doi:10.1145/3009837.3009894.
- [Sel04] Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004. doi:10.1017/S0960129504004256.
- [SV06] Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006. doi:10.1017/S0960129506005238.
- [van08] Jaap van Oosten. *Realizability. An Introduction to its Categorical Side*, volume 152 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2008.

APPENDIX A. SOUNDNESS

Lemma 5.1 (Substitution). *If $\Gamma, x : A \vdash \vec{t} : B$ and $\Delta \vdash v : A$, then the following diagram commutes:*

$$\begin{array}{ccc} \Gamma \boxtimes \Delta & \xrightarrow{\vec{t}[x:=v]} & B \\ & \searrow \text{Id} \boxtimes v & \nearrow \vec{t} \\ & \Gamma \boxtimes A & \end{array}$$

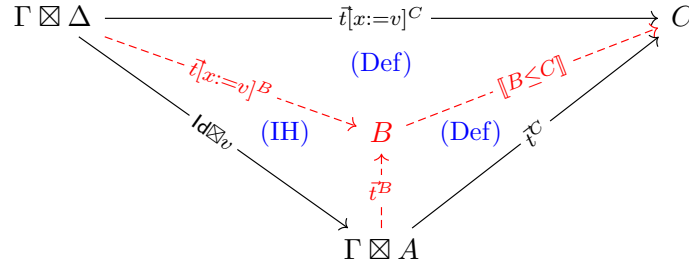
That is, $\llbracket \Gamma, \Delta \vdash \vec{t}[x := v] : B \rrbracket = \llbracket \Gamma, x : A, \Gamma \vdash \vec{t} : B \rrbracket \circ (\text{Id} \boxtimes \llbracket \Delta \vdash v : A \rrbracket)$.

Proof. By induction on the derivation of $\Gamma, x : A \vdash \vec{t} : B$.

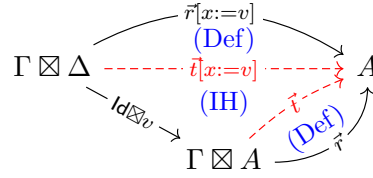
- $\frac{}{x : A \vdash x : A} \text{Ax}$

$$\begin{array}{ccc} 1 \boxtimes \Delta \cong \Delta & \xrightarrow{v} & A \\ & \searrow \text{Id} \boxtimes v & \nearrow \text{Id} \\ & 1 \boxtimes A \cong A & \end{array}$$

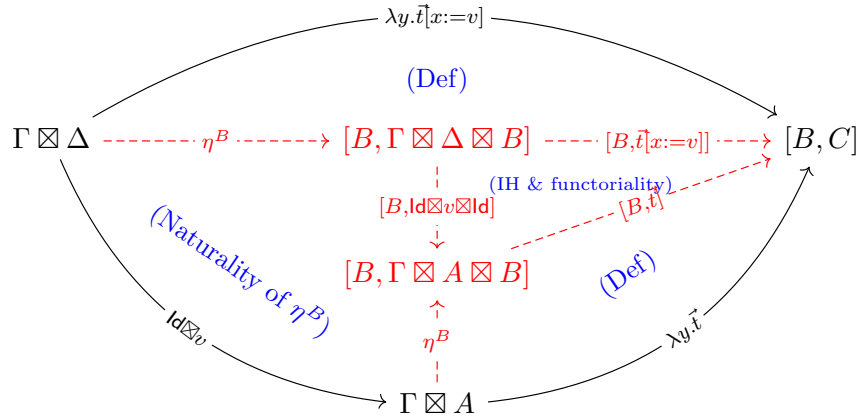
- $\frac{\Gamma, x : A \vdash \vec{t} : B \quad B \leq C}{\Gamma, x : A \vdash \vec{t} : C} \leq$



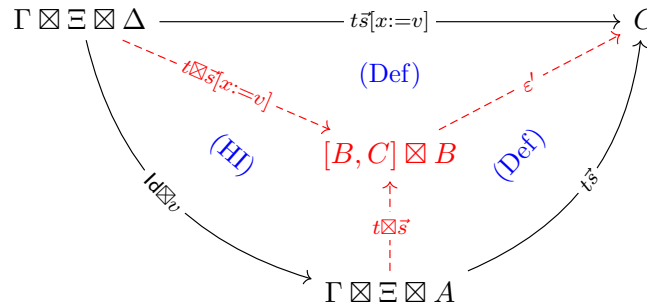
- $\frac{\Gamma \vdash \vec{t} : A \quad \vec{t} \equiv \vec{r}}{\Gamma \vdash \vec{r} : A} \equiv$



- $\frac{\Gamma, x : A, y : B \vdash \vec{t} : C}{\Gamma, x : A \vdash \lambda y. \vec{t} : B \rightarrow C} \text{Lam}$



- $\frac{\Gamma \vdash t : B \rightarrow C \quad \Xi, x : A \vdash \vec{s} : B}{\Gamma, \Xi, x : A \vdash t \vec{s} : C} \text{App}$

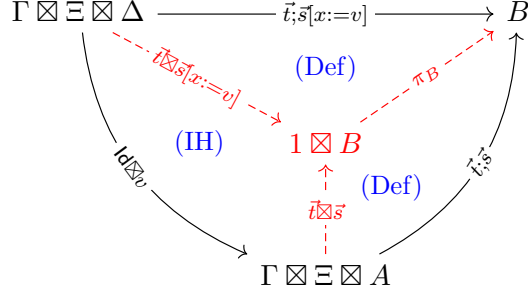


The case where $x \in FV(t)$ is analogous.

- $\frac{}{\vdash * : \mathbb{U}} \text{Void}$

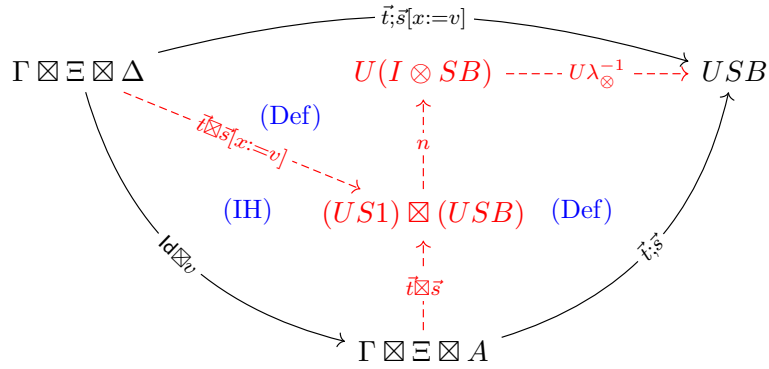
This case does not follow the hypothesis of the lemma.

- $$\frac{\Gamma \vdash \vec{t} : \mathbb{U} \quad \Xi, x : A \vdash \vec{s} : B}{\Gamma, \Xi, x : A \vdash \vec{t}; \vec{s} : B} \text{PureSeq}$$



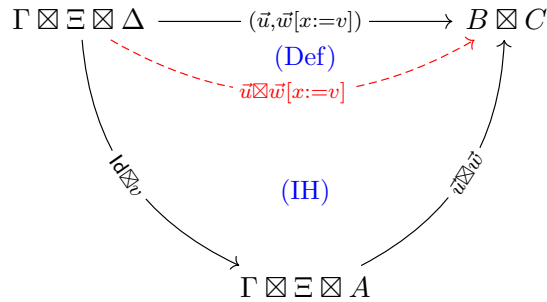
The case where $x \in FV(\vec{t})$ is analogous.

- $$\frac{\Gamma \vdash \vec{t} : \sharp \mathbb{U} \quad \Xi, x : A \vdash \vec{s} : \sharp B}{\Gamma, \Xi, x : A \vdash \vec{t}; \vec{s} : \sharp B} \text{UnitarySeq}$$



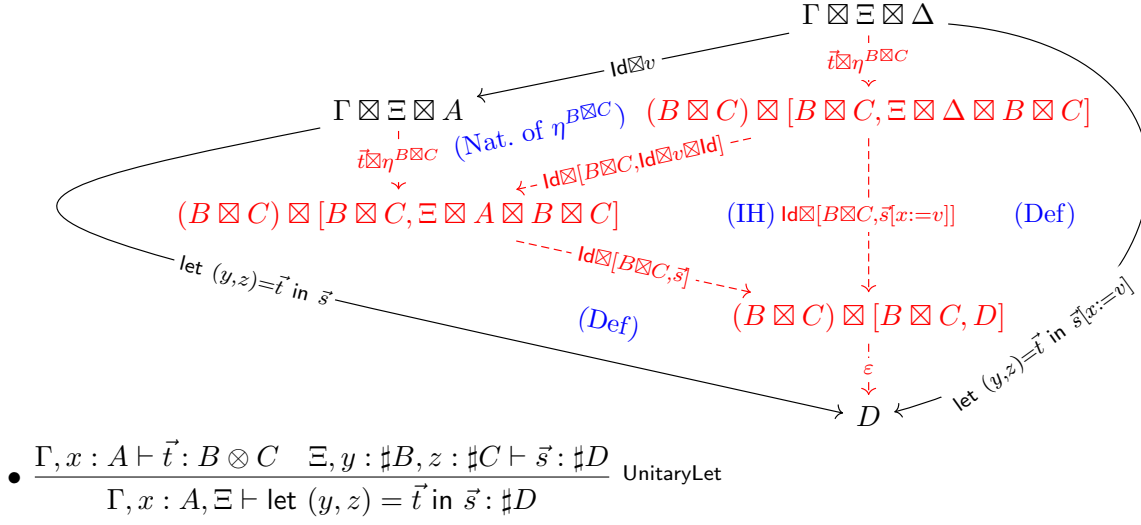
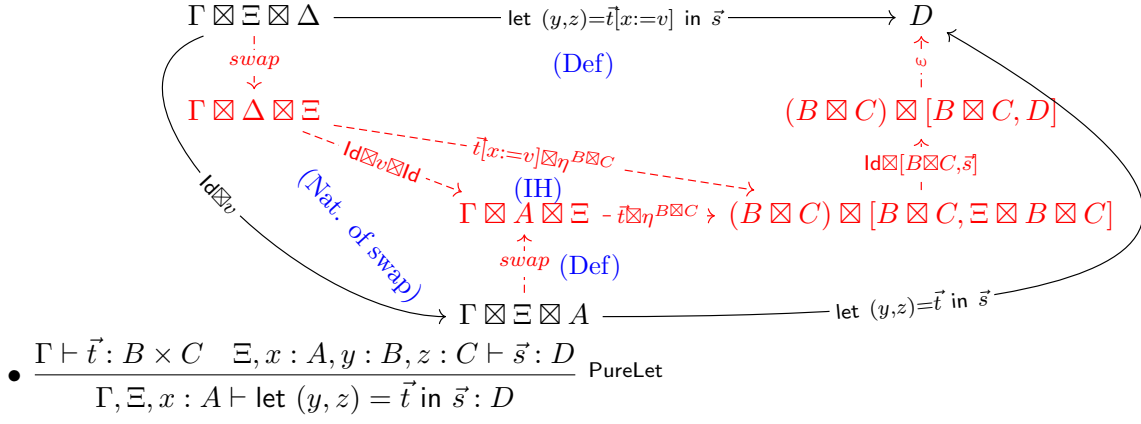
The case where $x \in FV(\vec{t})$ is analogous.

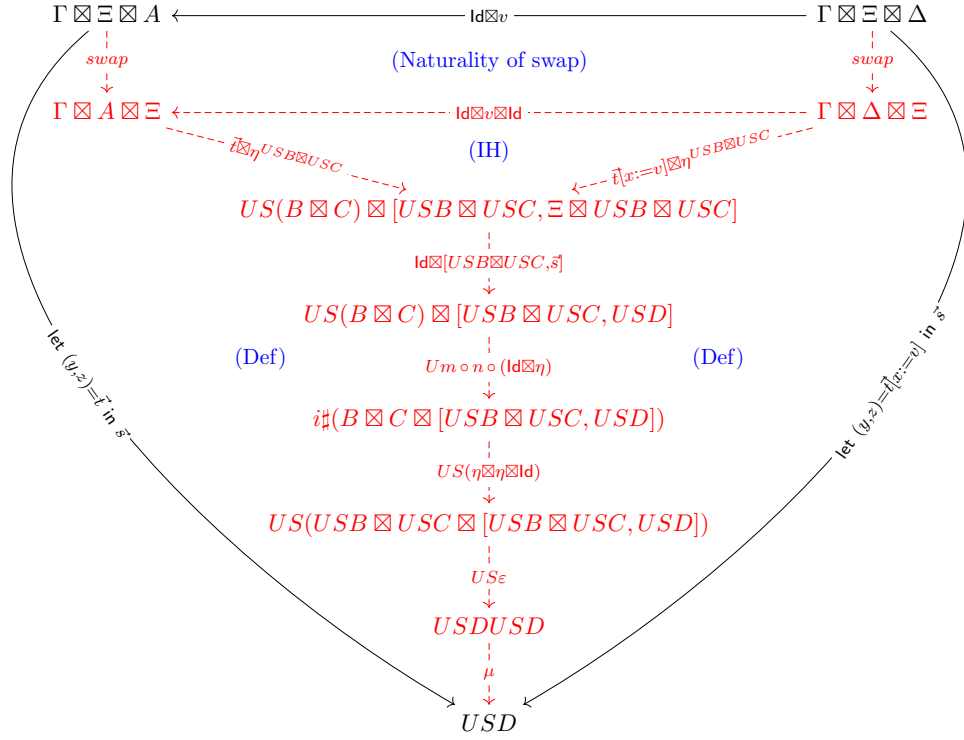
- $$\frac{\Gamma \vdash \vec{u} : B \quad \Xi, x : A \vdash \vec{w} : C}{\Gamma, \Xi, x : A \vdash (\vec{u}, \vec{w}) : B \times C} \text{Pair}$$



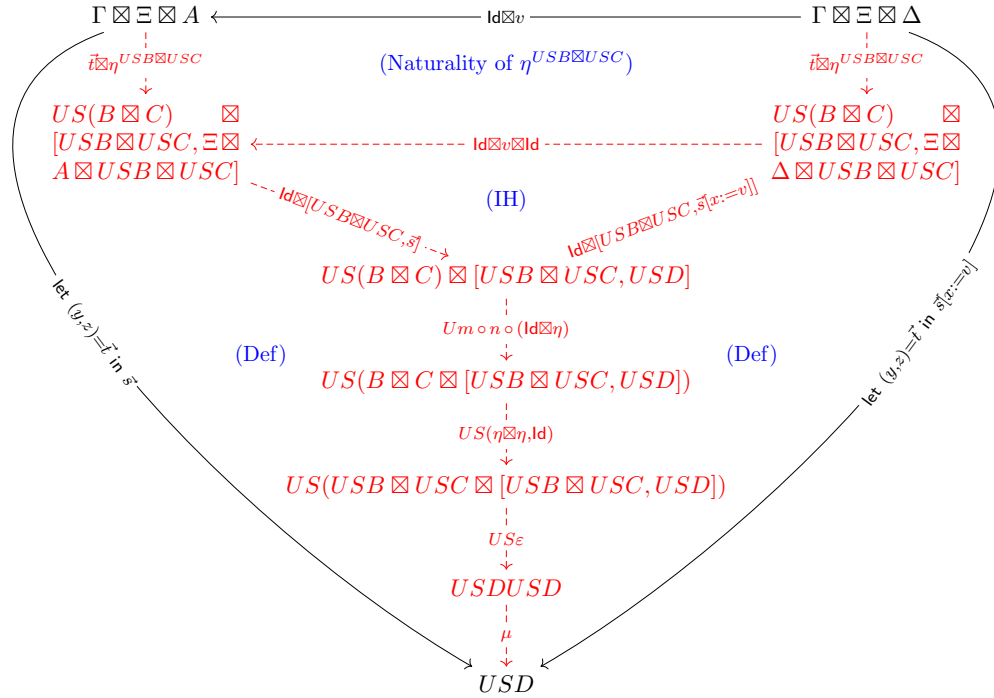
The case where $x \in FV(\vec{u})$ is analogous.

- $$\frac{\Gamma, x : A \vdash \vec{t} : B \times C \quad \Xi, y : B, z : C \vdash \vec{s} : D}{\Gamma, x : A, \Xi \vdash \text{let } (y, z) = \vec{t} \text{ in } \vec{s} : D} \text{PureLet}$$

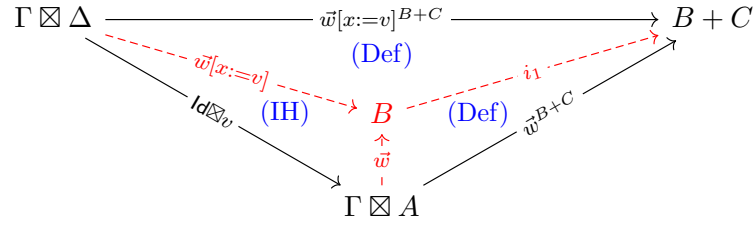




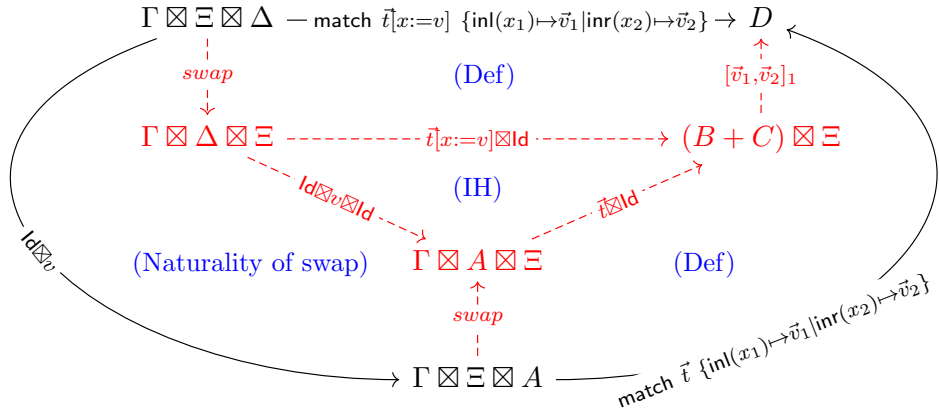
- $$\frac{\Gamma \vdash \vec{t} : B \otimes C \quad \Xi, x : A, y : \#B, z : \#C \vdash \vec{s} : \#D}{\Gamma, \Xi, x : A \vdash \text{let } (y, z) = \vec{t} \text{ in } \vec{s} : \#D} \text{UnitaryLet}$$



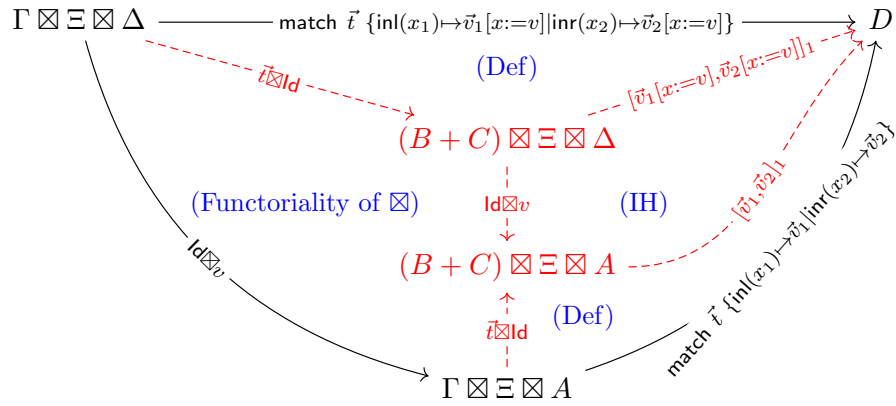
- $$\frac{\Gamma, x : A \vdash \vec{v} : B}{\Gamma, x : A \vdash \text{inl}(\vec{v}) : B + C} \text{InL}$$



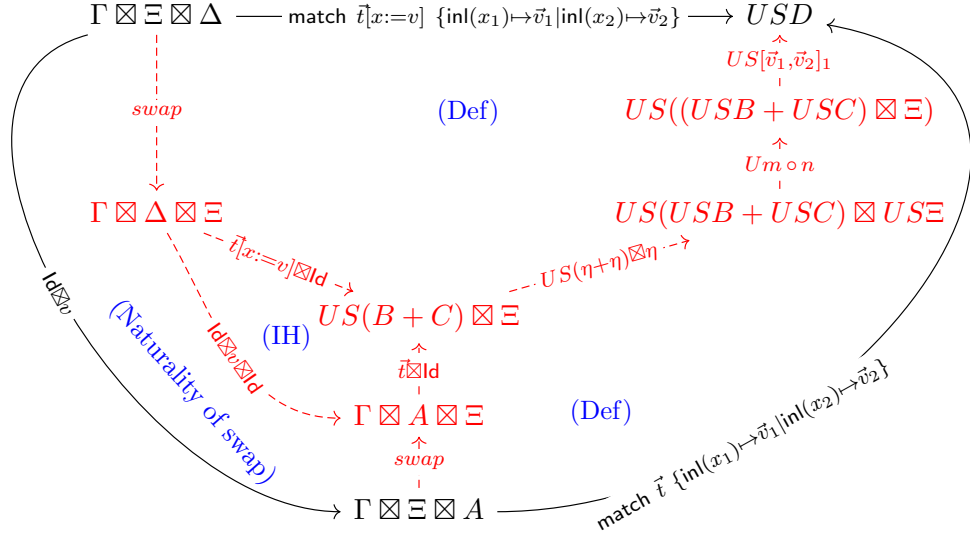
- $\frac{\Gamma, x : A \vdash \vec{v} : C}{\Gamma, x : A \vdash \text{inr}(\vec{v}) : B + C} \text{InR}$ Analogous to previous case.
- $\frac{\Gamma, x : A \vdash \vec{t} : B + C \quad \Xi \vdash (x_1 : B \vdash \vec{v}_1 \perp x_2 : C \vdash \vec{v}_2) : D}{\Gamma, x : A, \Xi \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : D} \text{PureMatch}$



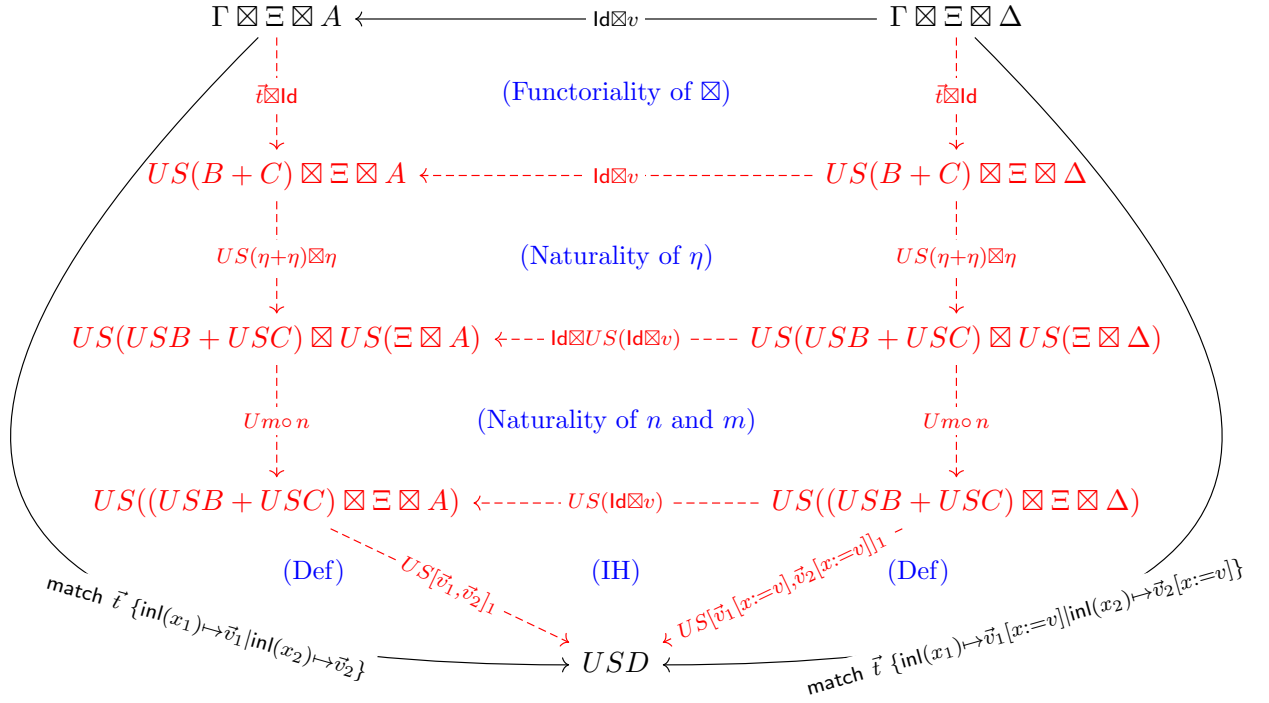
- $\frac{\Gamma \vdash \vec{t} : B + C \quad \Xi, x : A \vdash (x_1 : B \vdash \vec{v}_1 \perp x_2 : C \vdash \vec{v}_2) : D}{\Gamma, \Xi, x : A \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : D} \text{PureMatch}$



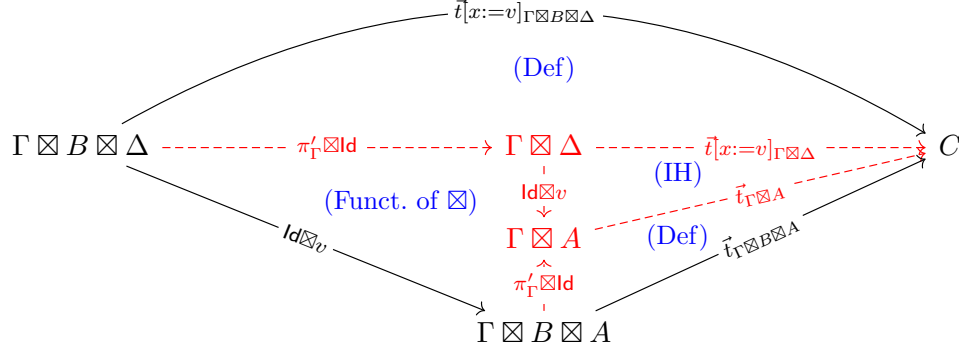
- $$\frac{\Gamma, x : A \vdash \vec{t} : B \oplus C \quad \Xi \vdash (x_1 : \#B \vdash \vec{v}_1 \perp x_2 : \#C \vdash \vec{v}_2) : \#D}{\Gamma, x : A, \Xi \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : \#D} \text{UnitaryMatch}$$



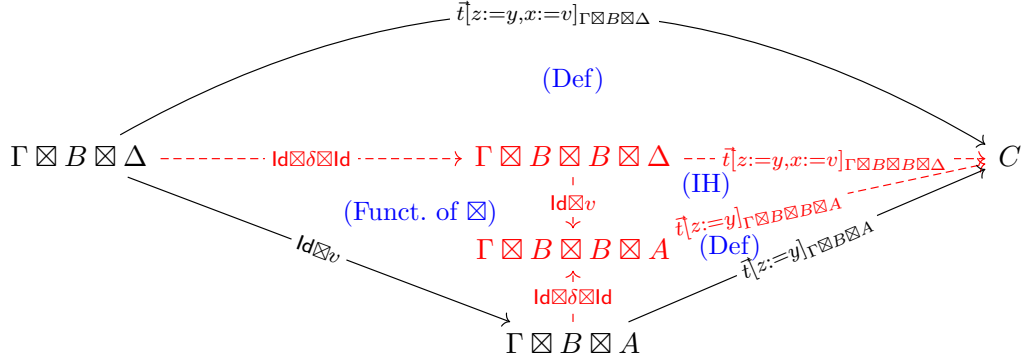
- $$\frac{\Gamma \vdash \vec{t} : B \oplus C \quad \Xi, x : A \vdash (x_1 : \#B \vdash \vec{v}_1 \perp x_2 : \#C \vdash \vec{v}_2) : \#D}{\Gamma, \Xi, x : A \vdash \text{match } \vec{t} \{ \text{inl}(x_1) \mapsto \vec{v}_1 \mid \text{inr}(x_2) \mapsto \vec{v}_2 \} : \#D} \text{UnitaryMatch}$$



- $\frac{\Gamma, x : A \vdash \vec{t} : C \quad B^b}{\Gamma, x : A, y : B \vdash \vec{t} : C} \text{Weak}$



- $\frac{\Gamma, y : B, x : A, z : B \vdash \vec{t} : C \quad B^b}{\Gamma, y : B, x : A \vdash \vec{t}[z := y] : C} \text{Contr}$



- $\frac{(k \neq h) \quad \vdash (\vec{v}_k \perp \vec{v}_h) : B \quad \sum_{j=1}^m |\alpha_j|^2 = 1 \quad m \geq 1 \quad B \neq C \rightarrow D}{\vdash \sum_{j=1}^m \alpha_j \cdot \vec{v}_j : \#B} \text{Sup}$

This case does not follow the hypothesis of the lemma. \square

Theorem 5.2 (Soundness). *If $\Gamma \vdash t : A$, and $t \longrightarrow r$, then $\llbracket \Gamma \vdash t : A \rrbracket = \llbracket \Gamma \vdash r : A \rrbracket$.*

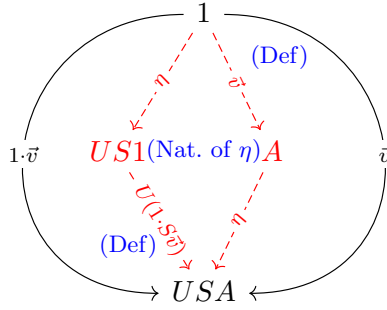
Proof. By induction on the rewrite relation, using the first derivable type for each term.

First we check the congruence rules, since the rewrite relation is defined modulo such congruence. Notice that the Unitary rules allow to type term distributions. However, at some point, such distributions were typed by the rule Sup, which only occur on closed values. It is easy to check that we can always use the congruence at the level of the closed values, and inherit such form in the final term distributions. Therefore, we only check the congruence rules for closed values.

- $\vec{v}_1 + \vec{v}_2 \equiv \vec{v}_2 + \vec{v}_1$ and $(\vec{v}_1 + \vec{v}_2) + \vec{v}_3 \equiv \vec{v}_1 + (\vec{v}_2 + \vec{v}_3)$ follow from the commutativity and associativity of $+$ in $\text{SVec}_{\mathbb{V}}$.
- $1 \cdot \vec{v} \equiv \vec{v}$. We have

$$\frac{\vdash \vec{v} : A}{\vdash 1 \cdot \vec{v} : \#A} \text{Sup} \quad \text{and} \quad \frac{\vdash \vec{v} : A}{\vdash \vec{v} : \#A} \leq$$

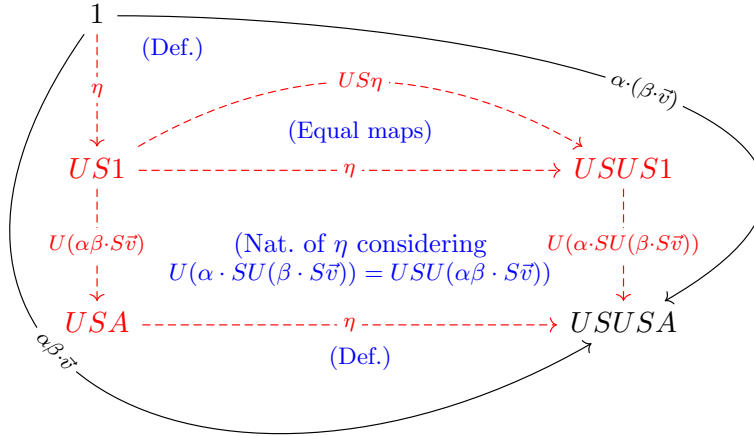
Then,



- $\alpha \cdot (\beta \cdot \vec{v}) \equiv \alpha\beta \cdot \vec{v}$. We have

$$\frac{\frac{\vdash \vec{v} : A}{\vdash \beta \cdot \vec{v} : \#A} \text{Sup}}{\vdash \alpha \cdot (\beta \cdot \vec{v}) : \#\#A} \text{Sup} \quad \text{and} \quad \frac{\frac{\vdash \vec{v} : A}{\vdash \alpha\beta \cdot \vec{v} : \#A} \text{Sup}}{\vdash \alpha\beta \cdot \vec{v} : \#\#A} \leq$$

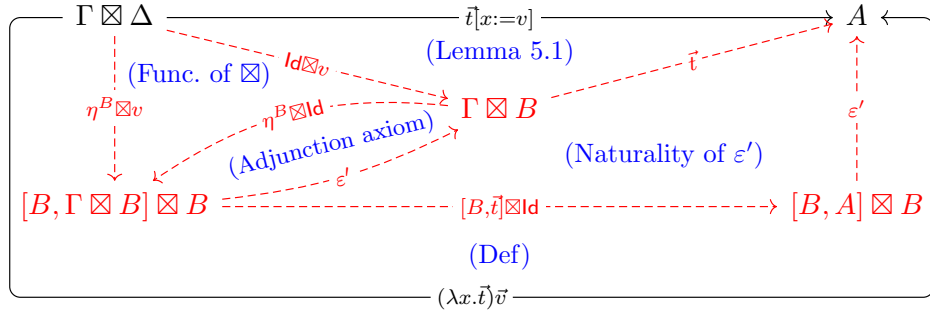
Then,



- $(\alpha + \beta) \cdot \vec{v} \equiv \alpha \cdot \vec{v} + \beta \cdot \vec{v}$. Notice that $\alpha \cdot \vec{v} + \beta \cdot \vec{v}$ is not typable in our calculus, so, this equivalence will always be taken in the form $(\alpha + \beta) \cdot \vec{v}$.
- $\alpha \cdot (\vec{v}_1 + \vec{v}_2) \equiv \alpha \cdot \vec{v}_1 + \alpha \cdot \vec{v}_2$. Same as before: if $\alpha \cdot (\vec{v}_1 + \vec{v}_2)$ is typable, then $\alpha \cdot \vec{v}_1 + \alpha \cdot \vec{v}_2$ is not, and vice-versa. Hence, only one of these two forms will be valid at each time.
- $(\lambda x.\vec{t})v \longrightarrow \vec{t}[x := v]$. We have

$$\frac{\frac{\Delta, x : B \vdash \vec{t} : A}{\Gamma \vdash \lambda x.\vec{t} : B \rightarrow A} \text{Lam}}{\Gamma, \Delta \vdash (\lambda x.\vec{t})v : A} \text{App} \quad \text{and} \quad \Gamma, \Delta \vdash \vec{t}[x := v] : A$$

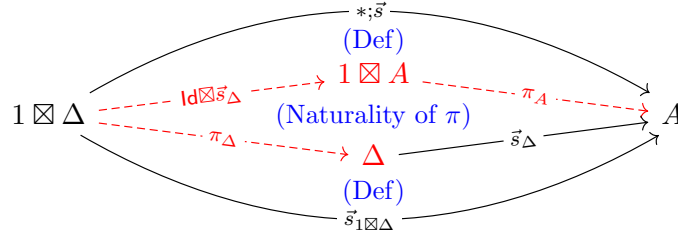
Then,



- $*; \vec{s} \longrightarrow \vec{s}$. We have

$$\frac{\overline{\vdash * : \mathbb{U}} \text{ Void} \quad \Delta \vdash \vec{s} : A}{\emptyset, \Delta \vdash *; \vec{s} : A} \text{ PureSeq} \quad \text{and} \quad \emptyset, \Delta \vdash \vec{s} : A$$

We write the \emptyset to stress the fact that there is a hidden $\times 1$, which can be projected out. Then,



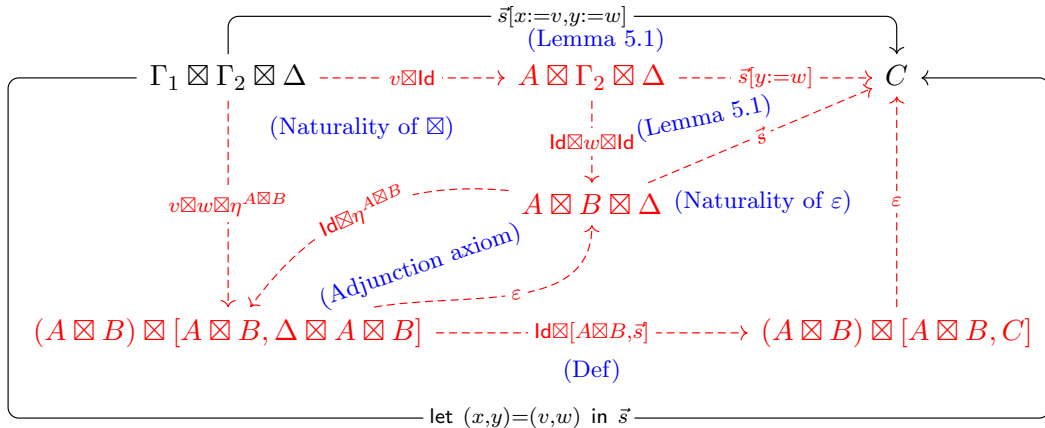
- $\text{let } (x, y) = (v, w) \text{ in } \vec{s} \longrightarrow \vec{s}[x := v, y := w]$. We have

$$\frac{\frac{\Gamma_1 \vdash v : A \quad \Gamma_2 \vdash w : B}{\Gamma_1, \Gamma_2 \vdash (v, w) : A \times B} \text{ Pair} \quad \Delta, x : A, y : B \vdash \vec{s} : C}{\Gamma_1, \Gamma_2, \Delta \vdash \text{let } (x, y) = (v, w) \text{ in } \vec{s} : C} \text{ PureLet}$$

and

$$\Gamma_1, \Gamma_2, \Delta \vdash \vec{s}[x := v, y := w] : C$$

Then,



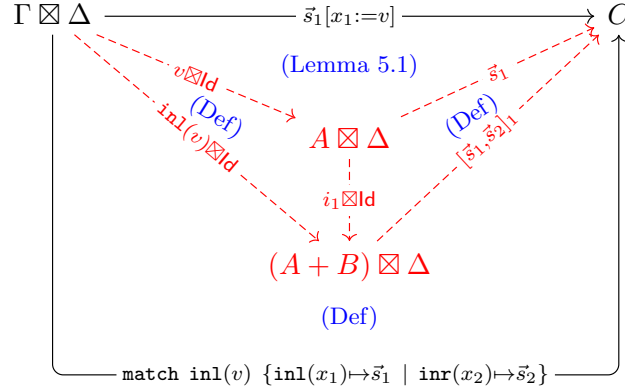
- **match inl**(v) {**inl**(x_1) \mapsto \vec{s}_1 | **inr**(x_2) \mapsto \vec{s}_2 } \longrightarrow $\vec{s}_1[x_1 := v]$. We have

$$\frac{\frac{\Gamma \vdash v : A}{\Gamma \vdash \mathbf{inl}(v) : A + B} \text{InL} \quad \Delta, x_1 : A \vdash \vec{s}_1 : C \quad \Delta, x_2 : B \vdash \vec{s}_2 : C}{\Gamma, \Delta \vdash \mathbf{match inl}(v) \{ \mathbf{inl}(x_1) \mapsto \vec{s}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{s}_2 \} : C} \text{PureMatch}}$$

and

$$\Gamma, \Delta \vdash \vec{s}_1[x_1 := v] : C$$

Then,



- **match inr**(v) {**inl**(x_1) \mapsto \vec{s}_1 | **inr**(x_2) \mapsto \vec{s}_2 } \longrightarrow $\vec{s}_2[x_2 := v]$. Analogous to previous case.

The inductive cases are straightforward. However, we have to check the notation for linear constructions, which give us the typing derivations using the “Unitary” rules.

Let $\vec{v} = \sum_{j=1}^n \alpha_j \cdot v_j$, $\vec{w} = \sum_{k=1}^m \beta_k \cdot w_k$, $\vec{t} = \sum_{h=1}^p \gamma_h \cdot t_h$, and $\vec{s} = \sum_{\ell=1}^q \delta_\ell \cdot s_\ell$.

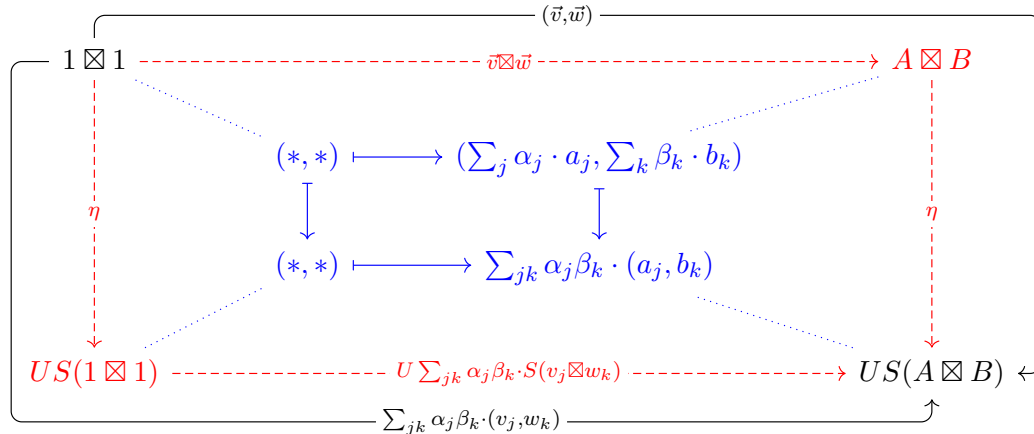
- $(\vec{v}, \vec{w}) := \sum_{jk} \alpha_j \beta_k \cdot (v_j, w_k)$.

We have

$$\frac{\frac{\vdash \vec{v} : A \quad \vdash w : B}{\vdash (\vec{v}, \vec{w}) : A \times B} \text{Pair}}{\vdash (\vec{v}, \vec{w}) : \#(A \times B)} \#$$

and

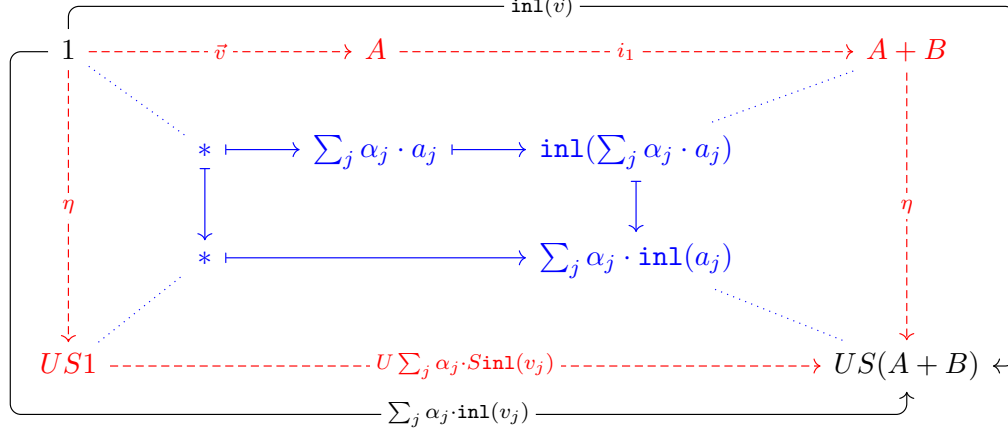
$$\frac{\frac{\vdash v_j : A \quad \vdash w_k : B}{\vdash ((v_j, w_k) \perp (v_h, w_\ell)) : A \times B} \text{Pair} \quad \sum_{jk} |\alpha_j \beta_k|^2 = 1}{\vdash \sum_{jk} \alpha_j \beta_k \cdot (v_j, w_k) : \#(A \times B)} \text{Sup}$$



- $\mathbf{inl}(\vec{v}) := \sum_{j=1}^n \alpha_j \cdot \mathbf{inl}(v_j)$.

We have

$$\frac{\frac{\vdash \vec{v} : A}{\vdash \mathbf{inl}(\vec{v}) : A + B} \text{InL}}{\vdash \mathbf{inl}(\vec{v}) : \sharp(A + B)} \sharp \quad \text{and} \quad \frac{\frac{\vdash v_j : A}{\vdash (v_h \perp v_k) : A + B} \text{InL} \quad \sum_j |\alpha_j|^2 = 1}{\vdash \sum_j \alpha_j \cdot \mathbf{inl}(v_j) : \sharp(A + B)} \text{Sup}$$



- $\mathbf{inr}(\vec{v}) := \sum_{j=1}^n \alpha_j \cdot \mathbf{inr}(v_j)$. Analogous to previous case.
- $t\vec{s} := \sum_{\ell=1}^q \delta_\ell \cdot ts_\ell$. In this case, there is only one way to type it, which is first doing the **Sup** to type \vec{s} , and then the **App**, because we cannot apply **Sup** on ts_ℓ , since these are not values.
- $\vec{t}; \vec{s} := \sum_{h=1}^p \gamma_h \cdot (t_h; \vec{s})$. Since $\Gamma \vdash \vec{t} : \sharp\mathbf{U}$, because of the orthogonality restriction on rule **Sup** the only possibility is $p = 1$. Thus, $\gamma_1 = \gamma$ and $t_1 = t$, with $|\gamma_1|^2 = 1$.

Furthermore, in this case there is only one way to type it, which is first doing the **Sup** to type \vec{t} , and then the **UnitarySeq**, because we cannot use the rule **Sup** on $t; \vec{s}$, which is not a value.

- $\mathbf{let} (x, y) = \vec{t} \mathbf{in} \vec{s} := \sum_{h=1}^p \gamma_h \cdot (\mathbf{let} (x, y) = t_h \mathbf{in} \vec{s})$. In this case, there is only one way to type it, which is first doing the **Sup** to type \vec{t} , and then the **UnitaryLet**, because we cannot superpose **let** terms since these are not values.
- $\mathbf{match} \vec{t} \{ \mathbf{inl}(x_1) \mapsto \vec{s}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{s}_2 \} := \sum_{h=1}^p \gamma_h \cdot (\mathbf{match} t_h \{ \mathbf{inl}(x_1) \mapsto \vec{s}_1 \mid \mathbf{inr}(x_2) \mapsto \vec{s}_2 \})$. In this case, there is only one way to type it, which is first doing the **Sup** to type \vec{t} , and then the **UnitaryMatch**, because we cannot superpose **match** terms since these are not values. \square