A Type System for the Vectorial Aspect of the Linear-Algebraic Lambda-Calculus

Pablo Arrighi

LIP, École Normale Supérieure de Lyon
46 allée d'Italie
69364 Lyon cedex 07, France

LIG, Université de Grenoble
220, rue de la Chimie
38400 Saint Martin d'Hères, France

pablo.arrighi@imag.fr

Alejandro Díaz-Caro

LIG, Université de Grenoble
220, rue de la Chimie
38400 Saint Martin d'Hères, France

LIPN, Université Paris 13, Sorbonne Paris Cité
99 av. J-B Clément,
93430 Villetaneuse, France

alejandro@diaz-caro.info

Benoît Valiron

LIPN, Université Paris 13, Sorbonne Paris Cité
99 av. J-B Clément,
93430 Villetaneuse, France

University of Pennsylvania
CIS Department
Philadelphia, PA 19104, USA

benoit.valiron@monoidal.net

We describe a type system for the linear-algebraic lambda-calculus. The type system accounts for the part of the language emulating linear operators and vectors, i.e. it is able to statically describe the linear combinations of terms resulting from the reduction of programs. This gives rise to an original type theory where types, in the same way as terms, can be superposed into linear combinations. We show that the resulting typed lambda-calculus is strongly normalising and features a weak subject-reduction.

1 Introduction

A number of recent works seek to endow the λ-calculus with a structure of vector space; this agenda has emerged simultaneously in two different contexts (albeit related [5]). A first line of work forked from the study of relational models of linear logic. In [7, 11, 17], various algebraic lambda-calculi, that is, languages with vectorial structures, are considered. These languages are based on an interpretation of intuitionistic logic by linear logic. A second line of work [2, 3, 6] considers linear combinations of terms as a sort of "quantum superposition". This paper stems from this second approach.

In quantum computation, data is encoded on normalised vectors in Hilbert spaces. For our purpose, it is enough to say that a Hilbert space is a vector space over the field of complex numbers. The smallest space usually considered is the space of *qubits*. This space is the two-dimensional vector space $\mathbb{C}^2$, and comes with a chosen orthonormal basis denoted by $\{|0\rangle, |1\rangle\}$. A general quantum bit (or qubit) is a normalised vector $\alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. The operations on qubits that we consider are the *quantum gates*, *i.e.* unitary operations. For our purpose, their interesting property is to be *linear*.

The language we consider in this paper will be called the *vectorial lambda-calculus*, denoted by λ^{vec}. It is inspired from *Lineal* [3]. This language admits the regular constructs of lambda-calculus: variables $x, y, \ldots$, lambda-abstractions $\lambda x.\mathbf{s}$ and application $(\mathbf{s})\mathbf{t}$. It also admits linear combinations of terms: $\mathbf{0}$, $\mathbf{s} + \mathbf{t}$ and $\alpha \cdot \mathbf{s}$ are terms. The scalar α ranges over the ring of complex numbers. As in [3], it behaves in

a call-by-value oriented manner, in the sense that $(\lambda x.\mathbf{r})(\mathbf{s}+\mathbf{t})$ first reduces to $(\lambda x.\mathbf{r})\mathbf{s}+(\lambda x.\mathbf{r})\mathbf{t}$ until *basis terms* are reached, at which point beta-reduction applies. The lambda-binder is not linear with respect to the vectorial structure: $\lambda x.(\mathbf{s}+\mathbf{t})$ is not the same thing as $\lambda x.\mathbf{s}+\lambda x.\mathbf{t}$; in fact abstractions and variables are exactly what is meant by basis terms.

The set of the normal forms of the terms can then be interpreted as a vector space and the term $(\lambda x.\mathbf{r})\mathbf{s}$ can be seen as the application of the linear operator $(\lambda x.\mathbf{r})$ to the vector $\mathbf{s}$. The goal of this paper is to give a formal description of this intuition at the level of the type system.

Related works and contribution.   This paper is part of a general research framework aiming at understanding the relationship between quantum computation and algebraic lambda-calculi [1, 3, 14, 15]. The ultimate goal of this research path is to design a typed language whose terms can be interpreted both as quantum data and descriptions of quantum algorithms. The type system would then provide a "quantum theoretical logic" and the language a Curry-Howard isomorphism for quantum computation.

The central question this paper is concerned with is the nature of the type system to be used. The solution we are proposing is an extension of two languages designed in [2] and [6].

The first paper [2] is uniquely concerned with the addition of scalars in the type system. If α is a scalar and $\Gamma \vdash \mathbf{t}\colon T$ is a sequent, $\alpha \cdot \mathbf{t}$ is of type $\alpha \cdot T$. The developed language actually provides a static analysis tool for *probabilistic* computation, when the scalars are taken to be *positive real numbers*. It however fails to address the issue in this paper: without sums but with negative numbers, the term $\lambda x.\lambda y.x - \lambda x.\lambda y.y$ is typed with $0 \cdot (X \rightarrow (X \rightarrow X))$, a type which fails to exhibits the fact that we have a superposition of terms.

The second paper [6] is concerned with the addition of sums to a regular type system. In this case, if $\Gamma \vdash \mathbf{s}\colon S$ and $\Gamma \vdash \mathbf{t}\colon T$ are two valid sequents, $\mathbf{s}+\mathbf{t}$ is of type $S+T$. However, the language considered is only the *additive* fragment of *Lineal*, it leaves scalars out of the picture.

The paper we present here builds on these two approaches. Its goal is to characterise the notion of vectors in the vectorial lambda-calculus. Because of the possible negative or complex coefficients, this requires to keep track of the 'direction' as well as the 'amplitude' of a term. We propose a type system with both sums and scalars, reflecting the vectorial structure of the vectorial lambda-calculus. Interestingly enough, combining the two separate features of [2, 6] raises subtle novel issues. In the end we achieve a type system which is such that if $\mathbf{t}$ has type $\sum_i \alpha_i \cdot U_i$, then it must reduce to a $\mathbf{t}'$ of the form $\sum_i \alpha_i \cdot \mathbf{b}_i$, where the $\mathbf{b}_i$'s are basis terms. The resulting language is strongly normalising, confluent, and features a weak-subject reduction.

Plan of the paper.   In Section 2, we present the language. We discuss the differences with the original language *Lineal* [3]. In Section 3, we expose the type system and the problem arising from the possibility of having linear combinations of types. Section 4 is devoted to subject reduction. We first say why the usual result is not valid, then we provide a solution and a candidate subject reduction theorem; the rest of the section is concerned with the proof of the result. In Section 5, we prove confluence and strong normalisation for this setting. Finally we close the paper with some examples in Section 6 and conclusions in Section 7.

2 The Terms

We consider the untyped language λ^{vec} described in Figure 1. It is based on *Lineal* [3]: terms come in two flavours, basis terms which are the only ones that will substitute a variable in a β-reduction step, and

general terms.

Terms are considered modulo associativity and commutativity of the operator $+$, making the reduction into an *AC-rewrite system* [9]. Scalars (notation $\alpha, \beta, \gamma, \dots$) form a ring $(\mathsf{S}, +, \times)$. The typical ring we consider in the examples is the ring of complex numbers. In particular, we shall use the shortcut notation $\mathbf{s} - \mathbf{t}$ in place of $\mathbf{s} + (-1) \cdot \mathbf{t}$. The set of free variables of a term is defined as usual: the only operator binding variables is the λ-abstraction. The operation of substitution on terms (notation $\mathbf{t}[\mathbf{b}/x]$) is defined in the usual way for the regular lambda-term constructs, by taking care of variable renaming to avoid capture. For a linear combination, the substitution is defined as follows: $(\alpha \cdot \mathbf{t} + \beta \cdot \mathbf{r})[\mathbf{b}/x] = \alpha \cdot \mathbf{t}[\mathbf{b}/x] + \beta \cdot \mathbf{r}[\mathbf{b}/x]$.

In addition to β-reduction, there are fifteen rules stemming from the oriented axioms of vector spaces [3], specifying the behaviour of sums and products. A general term $\mathbf{t}$ is thought of as a linear combination of terms $\alpha \cdot \mathbf{r} + \beta \cdot \mathbf{r}'$. When we apply $\mathbf{s}$ to this superposition, $(\mathbf{s})\,\mathbf{t}$ reduces to $\alpha \cdot (\mathbf{s})\,\mathbf{r} + \beta \cdot (\mathbf{s})\,\mathbf{r}'$.

Note that we need to choose a reduction strategy: we cannot reduce the term $(\lambda x.(x)\,x)\,(y + z)$ both to $(\lambda x.(x)\,x)\,y + (\lambda x.(x)\,x)\,z$ and to $(y + z)\,(y + z)$. Indeed, the former reduces to $(y)\,y + (z)\,z$ whereas the latter reduces to $(y)\,z + (y)\,y + (z)\,y + (z)\,z$. Since this calculus inherits from [2, 3, 6], we consider the beta-reduction acting in a call-by-value oriented way (in fact, "call-by-base" is a more accurate name).

$$
\begin{array}{lll}
& \textit{Terms:} & \mathbf{t}, \mathbf{r}, \mathbf{u} ::= \quad \mathbf{b} \mid (\mathbf{t})\,\mathbf{r} \mid \mathbf{0} \mid \alpha \cdot \mathbf{t} \mid \mathbf{t} + \mathbf{r} \\
& \textit{Basis terms:} & \mathbf{b} ::= \quad x \mid \lambda x.\mathbf{t} \\
\textit{Group E:} & \textit{Group F:} & \textit{Group A:} \\
0 \cdot \mathbf{t} \to \mathbf{0} & \alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t} \to (\alpha + \beta) \cdot \mathbf{t} & (\mathbf{t} + \mathbf{r})\,\mathbf{u} \to (\mathbf{t})\,\mathbf{u} + (\mathbf{r})\,\mathbf{u} \\
1 \cdot \mathbf{t} \to \mathbf{t} & \alpha \cdot \mathbf{t} + \mathbf{t} \to (\alpha + 1) \cdot \mathbf{t} & (\mathbf{t})\,(\mathbf{r} + \mathbf{u}) \to (\mathbf{t})\,\mathbf{r} + (\mathbf{t})\,\mathbf{u} \\
\alpha \cdot \mathbf{0} \to \mathbf{0} & \mathbf{t} + \mathbf{t} \to (1 + 1) \cdot \mathbf{t} & (\alpha \cdot \mathbf{t})\,\mathbf{r} \to \alpha \cdot (\mathbf{t})\,\mathbf{r} \\
\alpha \cdot (\beta \cdot \mathbf{t}) \to (\alpha \times \beta) \cdot \mathbf{t} & \mathbf{t} + \mathbf{0} \to \mathbf{t} & (\mathbf{t})\,(\alpha \cdot \mathbf{r}) \to \alpha \cdot (\mathbf{t})\,\mathbf{r} \\
\alpha \cdot (\mathbf{t} + \mathbf{r}) \to \alpha \cdot \mathbf{t} + \alpha \cdot \mathbf{r} & & (\mathbf{0})\,\mathbf{t} \to \mathbf{0} \\
& \textit{Group B:} & (\mathbf{t})\,\mathbf{0} \to \mathbf{0} \\
& (\lambda x.\mathbf{t})\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x] & \\
\end{array}
$$

$$
\dfrac{\mathbf{t} \to \mathbf{s}}{\alpha \cdot \mathbf{t} \to \alpha \cdot \mathbf{s}} \qquad \dfrac{\mathbf{t} \to \mathbf{s}}{\mathbf{r} + \mathbf{t} \to \mathbf{r} + \mathbf{s}} \qquad \dfrac{\mathbf{t} \to \mathbf{s}}{(\mathbf{r})\,\mathbf{t} \to (\mathbf{r})\,\mathbf{s}} \qquad \dfrac{\mathbf{t} \to \mathbf{s}}{(\mathbf{t})\,\mathbf{r} \to (\mathbf{s})\,\mathbf{r}} \qquad \dfrac{\mathbf{t} \to \mathbf{s}}{\lambda x.\mathbf{t} \to \lambda x.\mathbf{s}}
$$

Figure 1: Syntax, reduction rules and context rules of λ^{vec}.

Relation to other algebraic lambda-calculi. Although it is inspired from *Lineal*, the language λ^{vec} is closer to [2, 5, 6]. Indeed, *Lineal* considers some restrictions on the reduction rules, for example $\alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t} \to (\alpha + \beta) \cdot \mathbf{t}$ is only allowed when $\mathbf{t}$ is a closed normal term. These restrictions are enforced to ensure confluence in an untyped calculus. Indeed, consider the following example. Let $\mathbf{Y_b} = (\lambda x.(\mathbf{b} + (x)\,x))\,\lambda x.(\mathbf{b} + (x)\,x)$. Then $\mathbf{Y_b}$ reduces to $\mathbf{b} + \mathbf{Y_b}$. So the term $\mathbf{Y_b} - \mathbf{Y_b}$ reduces to $\mathbf{0}$, but also reduces to $\mathbf{b} + \mathbf{Y_b} - \mathbf{Y_b}$ and hence to $\mathbf{b}$, breaking confluence. The above restriction will forbid the first reduction, bringing back confluence.

A series of works [2, 5, 6] has shown that if one considers a typed language enforcing strong normalisation, one can wave many of the restrictions and consider a more canonical set of rewrite rules. Working with a type system enforcing strong normalisation (as shown in Section 5), we follow this approach.

Booleans in the vectorial lambda-calculus. We claimed in the introduction that this language was a candidate language for quantum computation. In this paragraph we show how quantum gates and matrices can be encoded.

First, both in λ^{vec} and in quantum computation one can interpret the notion of booleans. In the former we can consider the usual booleans $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$ whereas in the latter we consider the regular quantum bits $|0\rangle$ and $|1\rangle$.

In λ^{vec}, a representation of *if* **r** *then* **s** *else* **t** needs to take into account the special relation between sums and applications. We cannot directly encode this test as the usual $((\mathbf{r})\,\mathbf{s})\,\mathbf{t}$. Indeed, if **r**, **s** and **t** were respectively the terms **true**, $\mathbf{s}_1 + \mathbf{s}_2$ and $\mathbf{t}_1 + \mathbf{t}_2$, the term $((\mathbf{r})\,\mathbf{s})\,\mathbf{t}$ would reduce to $((\mathbf{true})\,\mathbf{s}_1)\,\mathbf{t}_1 + ((\mathbf{true})\,\mathbf{s}_1)\,\mathbf{t}_2 + ((\mathbf{true})\,\mathbf{s}_2)\,\mathbf{t}_1 + ((\mathbf{true})\,\mathbf{s}_2)\,\mathbf{t}_2$, then to $2\cdot\mathbf{s}_1 + 2\cdot\mathbf{s}_2$ instead of $\mathbf{s}_1 + \mathbf{s}_2$. We need to "freeze" the computations in each branch of the test so that the sum does not distribute over the application. For that purpose we use the well-known notion of *thunks*: we encode the test as $\{((\mathbf{r})\,[\mathbf{s}])\,[\mathbf{t}]\}$, where $[-]$ is the term $\lambda f.-$ with f a fresh, unused term variable and where $\{-\}$ is the term $(-)\lambda x.x$. The former "freezes" the computation while the latter "releases" it. Then the term *if* **true** *then* $(\mathbf{s}_1 + \mathbf{s}_2)$ *else* $(\mathbf{t}_1 + \mathbf{t}_2)$ reduces to the term $\mathbf{s}_1 + \mathbf{s}_2$ as one could expect. Note that this test is linear, in the sense that the term *if* $(\alpha\cdot\mathbf{true} + \beta\cdot\mathbf{false})$ *then* **s** *else* **t** reduces to $\alpha\cdot\mathbf{s} + \beta\cdot\mathbf{t}$.

This has a striking similarity with the *quantum test* that can be found *e.g.* in [1, 3, 14]. For example, consider the Hadamard gate **H** sending $|0\rangle$ to $\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$ and $|1\rangle$ to $\frac{\sqrt{2}}{2}(|0\rangle - |1\rangle)$. If x is a quantum bit, the value $(\mathbf{H})x$ can be represented as the quantum test *if* x *then* $\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$ *else* $\frac{\sqrt{2}}{2}(|0\rangle - |1\rangle)$. As developed in [3], one can simulate this operation in λ^{vec} using the test construction we just described: $\{(\mathbf{H})x\} = \{((x)\,[\frac{\sqrt{2}}{2}\cdot\mathbf{true} + \frac{\sqrt{2}}{2}\cdot\mathbf{false}])\,[\frac{\sqrt{2}}{2}\cdot\mathbf{true} - \frac{\sqrt{2}}{2}\cdot\mathbf{false}]\}$. Note that the thunks are necessary: the term $((x)\,(\frac{\sqrt{2}}{2}\cdot\mathbf{true} + \frac{\sqrt{2}}{2}\cdot\mathbf{false}))\,(\frac{\sqrt{2}}{2}\cdot\mathbf{true} - \frac{\sqrt{2}}{2}\cdot\mathbf{false})$ would reduce to the term $\frac{1}{2}(((x)\,\mathbf{true})\,\mathbf{true} + ((x)\,\mathbf{true})\,\mathbf{false} + ((x)\,\mathbf{false})\,\mathbf{true} + ((x)\,\mathbf{false})\,\mathbf{false})$, which is fundamentally different from the term **H** we are trying to emulate.

Of course, with this procedure we can "encode" any matrix. If the space is of some general dimension n, instead of the basis elements **true** and **false** we can choose the terms $\lambda x_1.\cdots.\lambda x_n.x_i$'s for $i = 1$ to n to encode the basis of the space.

3 The Type System

Building the type system.   Since we are considering a lambda-calculus, we need at least an arrow type $A \to B$. The terms **true** and **false** can therefore be typed in the usual way with $\mathbb{B} = X \to (X \to X)$, for a fixed type X. Since the sum $\frac{\sqrt{2}}{2}\cdot\mathbf{true} + \frac{\sqrt{2}}{2}\cdot\mathbf{false}$ is a superposition of terms of type $\mathbb{B}$, one could decide to also type it with the type $\mathbb{B}$; in general, a linear combination of terms of type A would be of type A. But then the terms $\lambda x.(1\cdot x)$ and $\lambda x.(2\cdot x)$ would both be of the same type $A \to A$, failing to address the fact that the former respects the norm whereas the latter does not.

To address this problem, we incorporate the notion of scalars in the type system: If A is a valid type, the construction $\alpha\cdot A$ is also a valid type and if the terms **s** and **t** are of type A, the term $\alpha\cdot\mathbf{s} + \beta\cdot\mathbf{t}$ is of type $(\alpha + \beta)\cdot A$. This was achieved in [2] and it allows us to distinguish between the two functions $\lambda x.(1\cdot x)$ and $\lambda x.(2\cdot x)$: the former is of type $A \to A$ whereas the latter is of type $A \to (2\cdot A)$.

Let us now consider the term $\frac{\sqrt{2}}{2}\cdot(\mathbf{true} - \mathbf{false})$. Using the above addition to the type system, this term should be of type $0\cdot\mathbb{B}$, which is odd in the light of the use we want to make of it. Indeed, applying the Hadamard gate to this term produces the term **false** of type $\mathbb{B}$: the "amplitude" of the type (the sum of the squares of the absolute values of the scalars) jumps from 0 to 1.

This time, the problem comes from the fact that the type system does not keep track of the "direction" of a term. We therefore propose to go one step further, and to allow sums in types. Provided that $\mathbb{T} = X \to (Y \to X)$ and $\mathbb{F} = X \to (Y \to Y)$ (with Y another fixed type), we can type the term $\frac{\sqrt{2}}{2}\cdot(\mathbf{true} - \mathbf{false})$

with $\frac{\sqrt{2}}{2} \cdot (\mathbb{T} - \mathbb{F})$, which has "amplitude" 1, in the same way that the type of **false** has "amplitude" 1.

This type system is also able to type the term $\mathbf{H} = \lambda x.((x)[\frac{\sqrt{2}}{2} \cdot \mathbf{true} + \frac{\sqrt{2}}{2} \cdot \mathbf{false}])[\frac{\sqrt{2}}{2} \cdot \mathbf{true} - \frac{\sqrt{2}}{2} \cdot \mathbf{false}]$, with $((\mathbf{I} \to \frac{\sqrt{2}}{2}.(\mathbb{T} + \mathbb{F})) \to (\mathbf{I} \to \frac{\sqrt{2}}{2}.(\mathbb{T} - \mathbb{F})) \to T) \to T$ provided that $\mathbf{I}$ is an identity type of the form $Z \to Z$ and that T and Z are any fixed types.

Let us try to type the term $\{(\mathbf{H})\mathbf{true}\}$. This is possible provided that the fixed type T is equal to $\mathbf{I} \to \frac{\sqrt{2}}{2}.(\mathbb{T} + \mathbb{F})$. If we now want to type the term $\{(\mathbf{H})\mathbf{false}\}$, the fixed type T needs to be equal to $\mathbf{I} \to \frac{\sqrt{2}}{2}.(\mathbb{T} - \mathbb{F})$: we cannot type the term $\{(\mathbf{H})(\frac{2}{\sqrt{2}} \cdot \mathbf{true} + \frac{2}{\sqrt{2}} \cdot \mathbf{false})\}$ since there is no possibility to conciliate the two constraints on T.

To solve this last problem, we introduce the forall construction in the type system, making it *System F* alike. The term $\mathbf{H}$ can now be typed with $\forall T.((\mathbf{I} \to \frac{\sqrt{2}}{2}.(\mathbb{T} + \mathbb{F})) \to (\mathbf{I} \to \frac{\sqrt{2}}{2}.(\mathbb{T} - \mathbb{F})) \to T) \to T$ and the types $\mathbb{T}$ and $\mathbb{F}$ are updated to be respectively $\forall XY.X \to (Y \to X)$ and $\forall XY.X \to (Y \to Y)$. The terms $\{(\mathbf{H})\mathbf{true}\}$ and $\{(\mathbf{H})\mathbf{false}\}$ can both be well-typed with respective types $\frac{\sqrt{2}}{2}.(\mathbb{T} + \mathbb{F})$ and $\frac{\sqrt{2}}{2}.(\mathbb{T} - \mathbb{F})$, as expected.

The term 0. Let us try to type the term $\mathbf{0}$. Analogously to what was done for terms, a natural possibility is to add a special type $\overline{0}$ to type it. This is a reasonable solution that has been used for example in [2]. In this naive interpretation, we would have $0 \cdot S$ equal to $\overline{0}$ and $\overline{0}$ would be the unit for the addition on types.

However, consider the following example. Let $\lambda x.x$ be of type $U \to U$ and let $\mathbf{r}$ be of type R. The term $\lambda x.x + \mathbf{r} - \mathbf{r}$ is of type $(U \to U) + 0 \cdot R$, that is, $(U \to U)$. Now choose $\mathbf{b}$ of type U: we are allowed to say that $(\lambda x.x + \mathbf{r} - \mathbf{r})\mathbf{b}$ is of type U. This term reduces to $\mathbf{b} + (\mathbf{r})\mathbf{b} - (\mathbf{r})\mathbf{b}$. If the type system is reasonable enough, we should at least be able to type $(\mathbf{r})\mathbf{b}$. However, since there is no constraints on the type R, this is difficult to enforce.

The problem comes from the fact that along the typing of $\mathbf{r} - \mathbf{r}$, the type of $\mathbf{r}$ is lost in the equivalence $\overline{0} \equiv 0 \cdot R$. The only solution is to distinguish $\overline{0}$ from $0 \cdot R$. We can also remove $\overline{0}$ altogether, and this is the choice we make for λ^{vec}: without type $\overline{0}$, we do not equate $T + 0 \cdot R$ and T.

The term $\mathbf{0}$ can be typed with any type $0 \cdot T$, so long as T is inhabited (*i.e.* $\mathbf{0}$ can come from a reduction of $\mathbf{r} - \mathbf{r}$ for some term $\mathbf{r}$ of type T).

3.1 Types

We now give a formal account of the type system. Types are defined in Figure 2. They come in two flavours: *unit types* and general types, that is, linear combinations of types. Unit types include all types of *System F* [8, Chapter 11] and intuitively they are used to type basis terms. The arrow type admits only a unit type in its domain. This is due to the fact that the argument of a lambda-abstraction can only be substituted by a basis term. For the same reason, type variables, denoted by $X, Y \ldots$ can only be substituted by unit types. The substitution of X by U in T is defined as usual and is written $T[U/X]$. For a linear combination, the substitution is defined as follows: $(\alpha \cdot T + \beta \cdot R)[U/X] = \alpha \cdot T[U/X] + \beta \cdot R[U/X]$. We also use the vectorial notation $T[\vec{U}/\vec{X}]$ for $T[U_1/X_1] \cdots [U_n/X_n]$ if $\vec{X} = X_1, \ldots, X_n$ and $\vec{U} = U_1, \ldots, U_n$, and also $\forall \vec{X}$ for $\forall X_1 \ldots X_n = \forall X_1. \ldots. \forall X_n$.

We define an equivalence relation $\equiv$ on types as the least congruence such that $1 \cdot T \equiv T$, $\alpha \cdot T + \beta \cdot T \equiv (\alpha + \beta) \cdot T$, $\alpha \cdot (\beta \cdot T) \equiv (\alpha \times \beta) \cdot T$, $T + R \equiv R + T$, $\alpha \cdot T + \alpha \cdot R \equiv \alpha \cdot (T + R)$, $T + (R + S) \equiv (T + R) + S$.

This makes the types into a weak module over the scalars: they almost form a module apart from the fact that there is no neutral element for the addition. Note that although we do not have any special type

$\overline{0}$ (as discussed at the beginning of the section), we do have $0 \cdot T$; however $0 \cdot T$ is not the neutral element of the addition on types.

We may use the summation (Σ) notation without ambiguity, due to the associativity and commutativity equivalences of $+$.

$$
\begin{array}{ll}
\textit{Types:} & T,R,S ::= \quad U \mid \alpha \cdot T \mid T + R \\
\textit{Unit types:} & U,V,W ::= \quad X \mid U \rightarrow T \mid \forall X.U
\end{array}
$$

$$
\frac{}{\Gamma, x : U \vdash x : U} \, ax
\qquad
\frac{\Gamma \vdash \mathbf{t} : T}{\Gamma \vdash \mathbf{0} : 0 \cdot T} \, 0_I
\qquad
\frac{\Gamma, x : U \vdash \mathbf{t} : T}{\Gamma \vdash \lambda x.\mathbf{t} : U \rightarrow T} \, \rightarrow_I
$$

$$
\frac{\Gamma \vdash \mathbf{t} : \sum_{i=1}^{n} \alpha_i \cdot \forall \vec{X}.(U \rightarrow T_i)
\qquad
\Gamma \vdash \mathbf{r} : \sum_{j=1}^{m} \beta_j \cdot V_j
\qquad
\forall V_j, \exists \vec{W}_j, U[\vec{W}_j / \vec{X}] = V_j}
{\Gamma \vdash (\mathbf{t}) \, \mathbf{r} : \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_j \cdot T_i[\vec{W}_j / \vec{X}]} \, \rightarrow_E
$$

$$
\frac{\Gamma \vdash \mathbf{t} : \sum_{i=1}^{n} \alpha_i \cdot U_i \quad X \notin FV(\Gamma)}
{\Gamma \vdash \mathbf{t} : \sum_{i=1}^{n} \alpha_i \cdot \forall X.U_i} \, \forall_I
\qquad
\frac{\Gamma \vdash \mathbf{t} : \sum_{i=1}^{n} \alpha_i \cdot \forall X.U_i}
{\Gamma \vdash \mathbf{t} : \sum_{i=1}^{n} \alpha_i \cdot U_i[V/X]} \, \forall_E
$$

$$
\frac{\Gamma \vdash \mathbf{t} : T}{\Gamma \vdash \alpha \cdot \mathbf{t} : \alpha \cdot T} \, \alpha_I
\qquad
\frac{\Gamma \vdash \mathbf{t} : T \quad \Gamma \vdash \mathbf{r} : R}{\Gamma \vdash \mathbf{t} + \mathbf{r} : T + R} \, +_I
\qquad
\frac{\Gamma \vdash \mathbf{t} : T \quad T \equiv R}{\Gamma \vdash \mathbf{t} : R} \, \equiv
$$

Figure 2: Types and typing rules of λ^{vec}.

The following lemmas give some properties of the equivalence relation. Types are linear combinations of unit types (Lemma 3.1). Finally, the equivalence is well-behaved with respect to type constructs (Lemma 3.2).

Lemma 3.1 (Types characterisation). *For any type T, there exist $n \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_n \in S$ and unit types $U_1, \ldots, U_n$ such that $T \equiv \sum_{i=1}^{n} \alpha_i \cdot U_i$.*

Proof. Structural induction on T. If T is a unit type, take $\alpha = n = 1$ and so $T \equiv \sum_{i=1}^{1} 1 \cdot U = 1 \cdot U$. If $T = \alpha \cdot T'$, then by the induction hypothesis $T' \equiv \sum_{i=1}^{n} \alpha_i \cdot U_i$, so $T = \alpha \cdot T' \equiv \alpha \cdot \sum_{i=1}^{n} \alpha_i \cdot U_i \equiv \sum_{i=1}^{n} (\alpha \times \alpha_i) \cdot U_i$. If $T = R + S$, then by the induction hypothesis $R \equiv \sum_{i=1}^{n} \alpha_i \cdot U_i$ and $S \equiv \sum_{j=1}^{m} \beta_j \cdot V_j$, so $T = R + S \equiv \sum_{i=1}^{n} \alpha_i \cdot U_i + \sum_{j=1}^{m} \beta_j \cdot V_j$. $\square$

Lemma 3.2 (Equivalence $\forall_I$).

1. $\sum_{i=1}^{n} \alpha_i \cdot U_i \equiv \sum_{j=1}^{m} \beta_j \cdot V_j \Leftrightarrow \sum_{i=1}^{n} \alpha_i \cdot \forall X.U_i \equiv \sum_{j=1}^{m} \beta_j \cdot \forall X.V_j$.

2. $\sum_{i=1}^{n} \alpha_i \cdot \forall X.U_i \equiv \sum_{j=1}^{m} \beta_j \cdot V_j \Rightarrow \forall V_j, \exists W_j, V_j \equiv \forall X.W_j$.

3. $T \equiv R \Rightarrow T[U/X] \equiv R[U/X]$.

Proof. Straightforward case by case analysis over the equivalence rules. $\square$

3.2 Typing Rules

The typing rules are described in Figure 2. Contexts are denoted by Γ, Δ, etc. and are defined as sets $\{x : U, \ldots\}$, where x is a term variable appearing only once in the set, and U is a unit type. The axiom

(*ax*) and the arrow introduction rule ($\rightarrow_I$) are the usual ones. The rule (0_I) to type the term **0** takes into account the discussion at the beginning of Section 3. This rule also ensures that the type of **0** is inhabited, discarding problematic types like $0 \cdot \forall X.X$. Any sum of typed terms can be typed using Rule ($+_I$). Similarly, any scaled typed term can be typed with (α_I). Rule ($\equiv$) ensures that equivalent types can be used to type the same terms. Finally, the particular form of the arrow-elimination rule ($\rightarrow_E$) is due to the rewrite rules in group A that distribute sums and scalars over application.

The need and use of this complicated arrow elimination can be illustrated by three examples.

Example 3.1. Rule ($\rightarrow_E$) is easier to read for trivial linear combinations. It states that provided that $\Gamma \vdash \mathbf{s} \colon \forall X.U \rightarrow S$ and $\Gamma \vdash \mathbf{t} \colon V$, if there exists some type W such that $V = U[W/X]$, then since the sequent $\Gamma \vdash \mathbf{s} \colon V \rightarrow S[W/X]$ is valid, we also have $\Gamma \vdash (\mathbf{s})\mathbf{t} \colon S[W/X]$.

Example 3.2. Consider the terms $\mathbf{b}_1$ and $\mathbf{b}_2$, of respective types U_1 and U_2. The term $\mathbf{b}_1 + \mathbf{b}_2$ is of type $U_1 + U_2$. We would reasonably expect the term $(\lambda x.x)(\mathbf{b}_1 + \mathbf{b}_2)$ to be also of type $U_1 + U_2$. This is the case thanks to Rule ($\rightarrow_E$). Indeed, type the term $\lambda x.x$ with the type $\forall X.X \rightarrow X$ and we can now apply the rule.

Example 3.3. A slightly more evolved example is the projection of a pair of elements. It is possible to encode in *System F* the notion of pairs and projections: $\langle \mathbf{b}, \mathbf{c} \rangle = \lambda x.((x)\,\mathbf{b})\,\mathbf{c}$, $\langle \mathbf{b}', \mathbf{c}' \rangle = \lambda x.((x)\,\mathbf{b}')\,\mathbf{c}'$, $\pi_1 = \lambda x.(x)(\lambda y.\lambda z.y)$ and $\pi_2 = \lambda x.(x)(\lambda y.\lambda z.z)$. Provided that $\mathbf{b}$, $\mathbf{b}'$, $\mathbf{c}$ and $\mathbf{c}'$ have respective types U, U', V and V', the type of $\langle \mathbf{b}, \mathbf{c} \rangle$ is $\forall X.(U \rightarrow V \rightarrow X) \rightarrow X$ and the type of $\langle \mathbf{b}', \mathbf{c}' \rangle$ is $\forall X.(U' \rightarrow V' \rightarrow X) \rightarrow X$. The term π_1 and π_2 can be typed respectively with $\forall XYZ.((X \rightarrow Y \rightarrow X) \rightarrow Z) \rightarrow Z$ and $\forall XYZ.((X \rightarrow Y \rightarrow Y) \rightarrow Z) \rightarrow Z$. The term $(\pi_1 + \pi_2)(\langle \mathbf{b}, \mathbf{c} \rangle + \langle \mathbf{b}', \mathbf{c}' \rangle)$ is then typable of type $U + U' + V + V'$, thanks to Rule ($\rightarrow_E$). Note that this is consistent with the rewrite system, since it reduces to $\mathbf{b} + \mathbf{c} + \mathbf{b}' + \mathbf{c}'$.

4 Subject Reduction

Since the terms of λ^{vec} are not explicitly typed, we are bound to have sequents such as $\Gamma \vdash \mathbf{t} \colon T_1$ and $\Gamma \vdash \mathbf{t} \colon T_2$ with distinct types T_1 and T_2 for the same term $\mathbf{t}$. Using Rules ($+_I$) and (α_I) we get the valid typing judgement $\Gamma \vdash \alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t} \colon \alpha \cdot T_1 + \beta \cdot T_2$. Given that $\alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t}$ reduces to $(\alpha + \beta) \cdot \mathbf{t}$, a regular subject reduction would ask for the valid sequent $\Gamma \vdash (\alpha + \beta) \cdot \mathbf{t} \colon \alpha \cdot T_1 + \beta \cdot T_2$. Since in general we do not have $\alpha \cdot T_1 + \beta \cdot T_2 \equiv (\alpha + \beta) \cdot T_1 \equiv (\alpha + \beta) \cdot T_2$, we need to find a way around this.

A first natural solution could be by using the notion of principal types. However, since our type system can be seen as an extension of *System F*, the usual examples for the absence of principal types apply to our settings: we cannot rely on that.

A second potentially natural solution could be to ask for the sequent $\Gamma \vdash (\alpha + \beta) \cdot \mathbf{t} \colon \alpha \cdot T_1 + \beta \cdot T_2$ to be valid. If we force this typing rule into the system, it seems to solve the problem but then the type of a term becomes pretty much arbitrary: with typing context Γ, the term $(\alpha + \beta) \cdot \mathbf{t}$ could be typed with any combination $\gamma \cdot T_1 + \delta \cdot T_2$, when $\alpha + \beta = \gamma + \delta$.

The approach we favour in this paper is by using a notion of order on types. The order, denoted with $\sqsubseteq$, will be chosen so that the factorisation rules make the types of terms smaller according to the order. We will ask in particular that $(\alpha + \beta) \cdot T_1 \sqsubseteq \alpha \cdot T_1 + \beta \cdot T_2$ and $(\alpha + \beta) \cdot T_2 \sqsubseteq \alpha \cdot T_1 + \beta \cdot T_2$ whenever T_1 and T_2 are types for the same term. This approach can also be extended to solve a second pitfall coming the rule $\mathbf{t} + \mathbf{0} \rightarrow \mathbf{t}$. Indeed, although $x : X \vdash x + \mathbf{0} : X + 0 \cdot T$ is well-typed for any inhabited T, the sequent $x : X \vdash x : X + 0 \cdot T$ is not valid in general. We therefore extend the ordering to also allow $X \sqsubseteq X + 0 \cdot T$.

4.1   An Ordering Relation on Types.

We start with another relation $\prec$ inspired from [4]. This relation can be deduced from Rules $(\forall_I)$ and $(\forall_E)$ as follows: write $T \prec R$ if either $T \equiv \sum_{i=1}^{n} \alpha_i \cdot U_i$ and $R \equiv \sum_{i=1}^{n} \alpha_i \cdot \forall X.U_i$ or $T \equiv \sum_{i=1}^{n} \alpha_i \cdot \forall X.U_i$ and $R \equiv \sum_{i=1}^{n} \alpha_i \cdot U_i[V/X]$. We denote the reflexive (with respect to $\equiv$) and transitive closure of $\prec$ with $\preceq$. The relation $\preceq$ admits a subsumption lemma.

Lemma 4.1 ($\preceq$-subsumption). *For any context Γ, any term t and any types T,R such that $T \preceq R$ and no free type variable in T occurs in Γ. Then $\Gamma \vdash t : T$ implies $\Gamma \vdash t : R$.*

Proof. One can assume $\exists S_1, \ldots, S_n \ / \ T \equiv S_1 \prec S_2 \prec \cdots \prec S_n \equiv R$ (if not, there must be an equivalence instead, so the lemma would hold due to the $\equiv$-rule). So for all i one has $S_i \equiv \sum_{j=1}^{n} \alpha_j \cdot U_j^i$, thus $\Gamma \vdash t : \sum_{j=1}^{n} \alpha_j \cdot U_j^i$ and using $(\forall_E)$ or $(\forall_I)$, we get $\Gamma \vdash t : \sum_{j=1}^{n} \alpha_j \cdot U_j^{i+1}$. Since $\sum_{j=1}^{n} \alpha_j \cdot U_j^{i+1} \equiv S_{i+1}$ we finally get $\Gamma \vdash t : S_{i+1}$. Repeating the process we eventually reach $\Gamma \vdash t : S_n \equiv R$. $\qquad\square$

We can now define the ordering relation $\sqsubseteq$ on types discussed above as the smallest reflexive transitive relation satisfying the rules:

1. $(\alpha + \beta) \cdot T \sqsubseteq \alpha \cdot T + \beta \cdot T'$ if there are Γ, t such that $\Gamma \vdash \alpha \cdot t : \alpha \cdot T$ and $\Gamma \vdash \beta \cdot t : \beta \cdot T'$.

2. $T \sqsubseteq T + 0.R$ for any type R.

3. If $T \preceq R$, then $T \sqsubseteq R$.

4. If $T \sqsubseteq R$ and $U \sqsubseteq V$, then $T + S \sqsubseteq R + S$, $\alpha \cdot T \sqsubseteq \alpha \cdot R$, $U \to T \sqsubseteq U \to R$ and $\forall X.U \sqsubseteq \forall X.V$.

Note that the fact that $\Gamma \vdash t : T$ and $\Gamma \vdash t : T'$ does not imply that $\beta \cdot T \sqsubseteq \beta \cdot T'$. Indeed, although $\beta \cdot T \sqsubseteq 0 \cdot T + \beta \cdot T'$, we do not have $0 \cdot T + \beta \cdot T' \equiv \beta \cdot T'$. Note also that this ordering is not a subtyping relation. Indeed, although $\vdash (\alpha + \beta) \cdot \lambda x.\lambda y.x : (\alpha + \beta) \cdot \forall X.X \to (X \to X)$ is valid and $(\alpha + \beta) \cdot \forall X.X \to (X \to X) \sqsubseteq \alpha \cdot \forall X.X \to (X \to X) + \beta \cdot \forall XY.X \to (Y \to Y)$, the sequent $\vdash (\alpha + \beta) \cdot \lambda x.\lambda y.x : \alpha \cdot \forall X.X \to (X \to X) + \beta \cdot \forall XY.X \to (Y \to Y)$ is not valid.

4.2   Weak Subject Reduction

Let R be any reduction rule from Figure 1. We denote $\to_R$ a one-step reduction by rule R. A weak version of the subject reduction theorem can be stated as follows.

Theorem 4.2 (Weak subject reduction). *For any terms t, t', any context Γ and any type T, if $t \to_R t'$ and $\Gamma \vdash t : T$, then:*

1. *if $R \notin$ Group F, then $\Gamma \vdash t' : T$;*

2. *if $R \in$ Group F, then $\exists S \sqsubseteq T$ such that $\Gamma \vdash t' : S$ and $\Gamma \vdash t : S$.*

How weak is this *weak* subject reduction? First, note that the usual subject reduction result holds for most of the rules. Second, Theorem 4.2 ensures that a term t of a given type, when reduced, can be typed with a type that is also valid for the term t. Third, we can characterise the order relation as follows.

Lemma 4.3 (Order characterisation). *For any type R, unit types $V_1, \ldots, V_m$ and scalars $\beta_1, \ldots, \beta_m$, if $R \sqsubseteq \sum_{j=1}^{m} \beta_j \cdot V_j$, then there exist a scalar δ, a natural number k, a set $N \subseteq \{1, \ldots, m\}$ and a unit type $W \preceq V_k$ such that $R \equiv \delta \cdot W + \sum_{j \in N} \beta_j \cdot V_j$ and $\sum_{j=1}^{m} \beta_j = \delta + \sum_{j \in N} \beta_j$.*

Proof. Structural induction on R. $\qquad\square$

How informative is the type judgement? The following three lemmas express formal relations between the types and their terms.

Lemma 4.4 (Scalars, scaling). *For any context Γ, term t, type T and scalar α, if $\Gamma \vdash \alpha \cdot t : T$, then there exists a type R such that $T \equiv \alpha \cdot R$ and if $\alpha \neq 0$, $\Gamma \vdash t : R$. Moreover, if $\Gamma \vdash \alpha \cdot t : \alpha \cdot T$, then $\Gamma \vdash t : T$.*

Proof. The first part of the Lemma follows by induction on the typing derivation. The second part of the Lemma, $\Gamma \vdash \alpha \cdot t : \alpha \cdot T \Rightarrow \Gamma \vdash t : T$, follows as corollary. If $\Gamma \vdash \alpha \cdot t : \alpha \cdot T$, we have just proved that there exists R such that $\alpha \cdot T \equiv \alpha \cdot R$ and $\Gamma \vdash t : R$. It is easy to check that $\alpha \cdot T \equiv \alpha \cdot R \Rightarrow T \equiv R$, so using rule $\equiv$, $\Gamma \vdash t : T$. □

Lemma 4.4 is precursor of the generation lemma for scalars (Lemma 4.13). However it is more specific since it assumes a specific type and therefore more accurate in the sense that it gives a specific type for the inverted rule which is not possible in the actual generation lemma.

Lemma 4.4 excludes the case of scaling by 0. It is covered by the following (whose proof is done by induction on the typing derivation).

Lemma 4.5 (Zeros). *For any context Γ, term t, unit types $U_1, \ldots, U_n$ and scalars $\alpha_1, \ldots, \alpha_n$, if $\Gamma \vdash 0.t : \sum_{i=1}^{n} \alpha_i \cdot U_i$, then $\forall i$, $\alpha_i = 0$ and there are scalars $\delta_1, \ldots, \delta_n$ such that $\Gamma \vdash t : \sum_{i=1}^{n} \delta_i \cdot U_i$.* □

A basis term can always be given a unit type (the proof is also done by induction on the typing derivation).

Lemma 4.6 (Basis terms). *For any context Γ, type T and basis term b, if $\Gamma \vdash b : T$ then there exists a unit type U such that $T \equiv U$.* □

In the remainder of this section we provide a few definitions and lemmas that are required in order to prove Theorem 4.2.

In the same way that we can change a type in a sequent by an equivalent one using rule $\equiv$, we can prove that this can also be done in the context (proof by induction on the typing derivation).

Lemma 4.7 (Context equivalence). *For any term t, any context $\Gamma = (x_i : U_i)_i$ and any type T, if $\Gamma \vdash t : T$ and $\Gamma' = (x_i : V_i)_i$ where $U_i \equiv V_i$, then $\Gamma' \vdash t : T$.* □

The following lemma is standard in proofs of subject reduction for *System F*-like systems, and can be found, *e.g.* in [4, Ch. 4]. It ensures that by substituting type variables for type or term variables in an adequate manner, the derived type is still valid.

Lemma 4.8 (Substitution lemma). *For any term t, basis term b, term variable x, context Γ, types T, U, $\vec{W}$ and type variables $\vec{X}$,*

1. *if $\Gamma \vdash t : T$, then $\Gamma[U/X] \vdash t : T[U/X]$;*
2. *if $\Gamma, x : U \vdash t : T$, $\Gamma \vdash b : U[\vec{W}/\vec{X}]$ and $\vec{X} \notin FV(\Gamma)$, then $\Gamma \vdash t[b/x] : T[\vec{W}/\vec{X}]$.*

Proof. Both results follow by induction on the typing derivation. □

Proving subject reduction requires the proof that each reduction rule preserves types. Thus three generation lemmas are required: two classical ones, for applications (Lemma 4.9) and for abstractions (Lemma 4.10 and Corollary 4.12) and one for linear combinations: sums, scalars and zero (Lemma 4.13). The first two lemmas follow by induction on the typing derivation.

Lemma 4.9 (Generation lemma (application)). *For any terms t, r, any context Γ and any type T, if $\Gamma \vdash (t)\,r : T$, then there exist natural numbers n, m, unit types $U, V_1, \ldots, V_m$, types $T_1, \ldots, T_n$ and scalars $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_m$, such that $\Gamma \vdash t : \sum_{i=1}^{n} \alpha_i \cdot \forall \vec{X}.(U \rightarrow T_i)$, $\Gamma \vdash r : \sum_{j=1}^{m} \beta_j \cdot V_j$, for all V_j, there exists $\vec{W}_j$ such that $U[\vec{W}_j/\vec{X}] = V_j$ and $\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_j \cdot T_i[\vec{W}_j/\vec{X}] \preceq T$.* □

Lemma 4.10 (Generation lemma (abstraction)). *For any term variable x, term **t**, context Γ and type T, if $\Gamma \vdash \lambda x.\mathbf{t}: R$, there exist types U and T such that $U \to T \preceq R$ and $\Gamma, x: U \vdash \mathbf{t}: T$.* $\square$

The following lemma is needed for the proof of Corollary 4.12.

Lemma 4.11 (Arrows comparison). *For any types T, R and any unit types U, V, if $V \to R \preceq U \to T$, then there exist $\vec{W}, \vec{X}$ such that $U \to T \equiv (V \to R)[\vec{W}/\vec{X}]$.*

Proof. A map $(\cdot)^\circ$ from types to types is defined by $X^\circ = X$, $(\alpha \cdot T)^\circ = \alpha \cdot T^\circ$, $(U \to T)^\circ = U \to T$, $(T + R)^\circ = T^\circ + R^\circ$ and $(\forall X.U)^\circ = U^\circ$.

We need two intermediate results (the first one follows from a structural induction on T and the second one is a case by case analysis on $T \prec R$ using the first result).

1. For any type T and unit type U, there exists a unit type V such that $(T[U/X])^\circ \equiv T^\circ[V/X]$

2. For any types T, R, if $T \preceq R$ then $\exists \vec{U}, \vec{X} / R^\circ \equiv T^\circ[\vec{U}/\vec{X}]$

Proof of the lemma: by definition $U \to T = (U \to T)^\circ$ which by 2 is equivalent to $(V \to R)^\circ[\vec{W}/\vec{X}] = (V \to R)[\vec{W}/\vec{X}]$. $\square$

Corollary 4.12 (of Lemma 4.10). *For any context Γ, term variable x, term **t**, type variables $\vec{X}$ and types U and T, if $\Gamma \vdash \lambda x.\mathbf{t}: \forall \vec{X}.(U \to T)$ then the typing judgement $\Gamma, x: U \vdash \mathbf{t}: T$ is valid.*

Proof. By Lemma 4.10, there exist V, R such that $V \to R \preceq \forall \vec{X}.(U \to T)$ and $\Gamma, x: V \vdash \mathbf{t}: R$. Note that $V \to R \preceq \forall \vec{X}.(U \to T) \preceq U \to T$, so by Lemma 4.11, there are $\vec{W}, \vec{Y}$ such that $U \to T \equiv (V \to R)[\vec{W}/\vec{Y}] \equiv V[\vec{W}/\vec{Y}] \to R[\vec{W}/\vec{Y}]$ so $U \equiv V[\vec{W}/\vec{Y}]$ and $T \equiv R[\vec{W}/\vec{Y}]$. Also by Lemma 4.8, $\Gamma[\vec{W}/\vec{Y}], x: V[\vec{W}/\vec{Y}] \vdash \mathbf{t}: R[\vec{W}/\vec{Y}]$. By Lemma 4.7 and Rule $(\equiv)$, $\Gamma[\vec{W}/\vec{Y}], x: U \vdash \mathbf{t}: T$. If $\Gamma[\vec{W}/\vec{Y}] \equiv \Gamma$, we are done. Otherwise, $\vec{Y}$ appears free in Γ. Since $V \to R \preceq U \to T$ and $\Gamma \vdash \lambda x.\mathbf{t}: V \to R$, according to Lemma 4.1, $U \to T$ can be obtained from $V \to R$ as a type for $\lambda x.\mathbf{t}$: we would need to use Rule $(\forall_I)$; thus $\vec{Y}$ cannot appear free in Γ, which constitutes a contradiction. So, $\Gamma, x: U \vdash \mathbf{t}: T$. $\square$

Lemma 4.13 (Generation lemma (linear combinations)). *For any context Γ, scalar α, terms **t** and **r** and types S and T:*

1. *if $\Gamma \vdash \mathbf{t} + \mathbf{r}: S$ then there exist types R and R' such that $\Gamma \vdash \mathbf{t}: R$, $\Gamma \vdash \mathbf{r}: R'$ and $R + R' \preceq S$;*

2. *if $\Gamma \vdash \alpha \cdot \mathbf{t}: T$, then there exists a type R such that $\alpha \cdot R \preceq T$ and $\Gamma \vdash \alpha \cdot \mathbf{t}: \alpha \cdot R$;*

3. *if $\Gamma \vdash \mathbf{0}: T$, then there exists a type R such that $T \equiv 0 \cdot R$.*

Proof. All the cases follow by structural induction on the typing derivation. $\square$

4.3   Proof of Theorem 4.2

We are now ready to prove Theorem 4.2.

Proof. Let $\mathbf{t} \to_R \mathbf{t}'$ and $\Gamma \vdash \mathbf{t}: T$. We proceed by induction. We only give two interesting cases.

$R = \alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t} \to (\alpha + \beta) \cdot \mathbf{t}$. Let $\Gamma \vdash \alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t}: T$. Then by Lemma 4.13, there are types R, S such that $\Gamma \vdash \alpha \cdot \mathbf{t}: R$ and $\Gamma \vdash \beta \cdot \mathbf{t}: S$ with $R + S \preceq T$. By Lemma 4.13, there exists a type R' such that $\alpha.R' \preceq R$ and $\Gamma \vdash \alpha \cdot \mathbf{t}: \alpha \cdot R'$, and there exists a type S' such that $\beta \cdot S' \preceq S$ and $\Gamma \vdash \beta \cdot \mathbf{t}: \beta \cdot S'$.

- If $\alpha \neq 0$ (or analogously $\beta \neq 0$), then by Lemma 4.4, $\Gamma \vdash \mathbf{t}: R'$ and so by (α_I) we conclude $\Gamma \vdash (\alpha + \beta) \cdot \mathbf{t}: (\alpha + \beta) \cdot R'$. Notice that $(\alpha + \beta) \cdot R' \sqsubseteq \alpha \cdot R' + \beta \cdot S' \sqsubseteq R + S \sqsubseteq T$. Also using Rules $(+_I)$ and $(\equiv)$ we conclude $\Gamma \vdash \alpha \cdot \mathbf{t} + \beta \cdot \mathbf{t}: (\alpha + \beta) \cdot R'$.

- If $\alpha = \beta = 0$, then notice that $\Gamma \vdash 0 \cdot \mathbf{t} : 0 \cdot R'$ and $0 \cdot R' \sqsubseteq 0 \cdot R' + 0 \cdot S' \sqsubseteq R + S \sqsubseteq T$. Using again Rules $(+_I)$ and $(\equiv)$, we conclude $\Gamma \vdash 0 \cdot \mathbf{t} + 0 \cdot \mathbf{t} : 0 \cdot R'$.

$R = (\lambda x.\mathbf{t}) \, \mathbf{b} \to \mathbf{t}[\mathbf{b}/x]$. Let $\Gamma \vdash (\lambda x.\mathbf{t}) \, \mathbf{b} : T$. Then by Lemma 4.9, there exist numbers n, m, scalars $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m$, a unit type U, and general types $T_1, \ldots, T_n$ such that $\Gamma \vdash \lambda x.\mathbf{t} : \sum_{i=1}^{n} \alpha_i \cdot \forall \vec{X}.(U \to T_i)$ and $\Gamma \vdash \mathbf{b} : \sum_{j=1}^{m} \beta_j \cdot V_j$ with $\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_j \cdot T_i[\vec{W}_j/\vec{X}] \preceq T$ and where for all j, $\vec{W}_j$ is such that $U[\vec{W}_j/\vec{X}] \equiv V_j$.

By Lemma 4.6, $\sum_{i=1}^{n} \alpha_i \cdot \forall \vec{X}.(U \to T_i) \equiv \forall \vec{X}.(U \to T_i)$ and for all i, k, $T_i = T_k$. Analogously $\sum_{j=1}^{m} \beta_j \cdot V_j \equiv V_j$ where for all j, h, $V_j = V_h$. So $\sum_{i=1}^{n} \alpha_i = 1$ and $\sum_{j=1}^{m} \beta_i = 1$. Then by Rule $(\equiv)$, $\Gamma \vdash \lambda x.\mathbf{t} : \forall \vec{X}.(U \to T_i)$, and $\Gamma \vdash \mathbf{b} : V_i$.

Thus, by Corollary 4.12, $\Gamma, x : U \vdash \mathbf{t} : T_i$. Notice that $V_i \equiv U[\vec{W}_i/\vec{X}]$. By Lemma 4.8, we have $\Gamma \vdash \mathbf{t}[\mathbf{b}/x] : T_i[\vec{W}_i/\vec{X}]$. Since $T_i[\vec{W}_j/\vec{X}] \equiv (1 \times 1) \cdot T_i[\vec{W}_j/\vec{X}] = (\sum_{i=1}^{n} \alpha_i) \times (\sum_{j=1}^{m} \beta_j) \cdot T_i[\vec{W}_j/\vec{X}] = (\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_j) \cdot T_i[\vec{W}_j/\vec{X}]$, and since all the T_i are equivalents between them, this type is equivalent to $\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \times \beta_j \cdot T_i[\vec{W}_j/\vec{X}] \preceq T$. By Lemma 4.1, we conclude $\Gamma \vdash \mathbf{t}[\mathbf{b}/x] : T$. $\square$

5 Confluence and Strong Normalisation

The language has the usual properties for a typed lambda-calculus: the reduction is locally confluent and the type system enforces strong normalisation. From these two results, we infer the confluence of the rewrite system.

Theorem 5.1 (Local confluence). *For any terms t, r_1 and r_2, if $t \to r_1$ and $t \to r_2$, then there exists a term u such that $r_1 \to^* u$ and $r_2 \to^* u$.*

Proof. First, one proves the local confluence of the algebraic fragment of the rewrite system (that is, all the rules minus the beta-reduction). This has been automatised [16] using COQ [13]. The proof of confluence of the beta-reduction alone is a straightforward extension of the proof of confluence of the usual untyped λ-calculus which can be found in many textbooks, *e.g.* [10, Sec. 1.3]. Finally, a straightforward induction entails that the two fragments commute: this entails the local confluence of the whole rewrite system. $\square$

For proving strong normalisation of well-typed terms, we use reducibility candidates, a well-known method described for example in [8, Ch. 14] The technique is adapted to linear combinations of terms.

A *neutral term* is a term that is not a lambda-abstraction and that does reduce to anything. The set of *closed neutral terms* is denoted with $\mathcal{N}$. We write Λ_0 for the set of closed terms and SN_0 for the set of closed, strongly normalising terms. If $\mathbf{t}$ is any term, $\mathrm{Red}(\mathbf{t})$ is the set of all terms $\mathbf{t}'$ such that $\mathbf{t} \to \mathbf{t}'$. It is naturally extended to sets of terms. We say that a set S of closed terms is a reducibility candidate, denoted with $S \in \mathsf{RC}$ if the following conditions are verified:

RC$_1$ Strong normalisation: $S \subseteq SN_0$.

RC$_2$ Stability under reduction: $\mathbf{t} \in S$ implies $\mathrm{Red}(\mathbf{t}) \subseteq S$.

RC$_3$ Stability under neutral expansion: If $\mathbf{t} \in \mathcal{N}$ and $\mathrm{Red}(\mathbf{t}) \subseteq S$ then $\mathbf{t} \in S$.

RC$_4$ The common inhabitant: $\mathbf{0} \in S$.

We define the following operations on reducibility candidates. Let A and B be in RC. $\mathsf{A} \to \mathsf{B}$ is the closure of $\{\mathbf{t} \in \Lambda_0 \mid \forall \mathbf{b} \in \mathsf{A}, (\mathbf{t}) \, \mathbf{b} \in \mathsf{B}\}$ under **RC$_3$** and **RC$_4$**, where $\mathbf{b}$ is a base term. If $\{\mathsf{A}_i\}_i$ is a family of reducibility candidates, $\sum_i \mathsf{A}_i$ is the closure of $\{\sum_i \alpha_i \cdot \mathbf{t}_i \mid \mathbf{t}_i \in \mathsf{A}_i\}$ under **RC$_2$** and **RC$_3$**. If there is only one type A in the sum, we write $1 \cdot A$ instead.

Lemma 5.2. *If* A, B *and all the* A_i*'s are in* RC, *then so are* A $\to$ B, $\sum_i A_i$ *and* $\cap_i A_i$.

Proof. We need an intermediate result first, showing that

> if $\{\mathbf{t}_i\}_i$ is a family of strongly normalising term, then so is any linear combination of term made of the $\mathbf{t}_i$

Proof of this result: Let $\vec{\mathbf{t}} = \mathbf{t}_1, \ldots, \mathbf{t}_n$. We define the *algebraic context* $F(\cdot)$ by the following grammar: $F(\vec{\mathbf{t}}) ::= \mathbf{t}_i \,|\, F(\vec{\mathbf{t}}) + F(\vec{\mathbf{t}}) \,|\, \alpha \cdot F(\vec{\mathbf{t}}) \,|\, \mathbf{0}$. We claim that for all algebraic contexts $F(\cdot)$ and all strongly normalising terms $\mathbf{t}_i$ that are not linear combinations (that is, of the form x, $\lambda x.\mathbf{r}$ or (s) r), the term $F(\vec{\mathbf{t}})$ is also strongly normalising. The claim is proven by induction on $s(\vec{\mathbf{t}})$, the sum over i of the sum of the lengths of all the possible rewrite sequences starting with $\mathbf{t}_i$.
Proof of the lemma:

A $\to$ B **RC$_1$**: Assume that $\mathbf{t} \in$ A $\to$ B is not in SN_0. Then there is an infinite sequence of reductions $(\mathbf{t}_n)_n$ with $\mathbf{t}_0 = \mathbf{t}$. So there is an infinite sequence of reduction $((\mathbf{t}_n)\,\mathbf{b})_n$ starting with $(\mathbf{t})\mathbf{b}$, for all base terms $\mathbf{b}$. This contradicts the definition of A $\to$ B. **RC$_2$**: We must show that if $\mathbf{t} \to \mathbf{t}'$ and $\mathbf{t} \in$ A $\to$ B, then $\mathbf{t}' \in$ A $\to$ B. Let $\mathbf{t}$ such that for all $\mathbf{b} \in$ A, $(\mathbf{t})\,\mathbf{b} \in$ B. Then by **RC$_2$** in B, $(\mathbf{t}')\,\mathbf{b} \in$ B, and so $\mathbf{t}' \in$ A $\to$ B. If $\mathbf{t}$ is neutral and Red$(\mathbf{t}) \subseteq$ A $\to$ B, then $\mathbf{t}' \in$ A $\to$ B since $\mathbf{t}' \in$ Red$(\mathbf{s})$. If $\mathbf{t} = \mathbf{0}$, it does not reduce. **RC$_3$** and **RC$_4$**: Trivially true by definition.

$\sum_i A_i$ **RC$_1$**: If $\mathbf{t} \in \{\sum_i \alpha_i.\mathbf{t}_i \,|\, \mathbf{t}_i \in A_i\}$, the result is trivial by condition **RC$_1$** on the A_i and the previous result about linear combination of strongly normalising terms. If $\mathbf{t}$ is neutral and Red$(\mathbf{t}) \subseteq$ A + B, then $\mathbf{t}$ is strongly normalising since all elements of Red$(\mathbf{t})$ are strongly normalising. **RC$_2$** and **RC$_3$**: Trivially true by definition. **RC$_4$**: Since $\sum_i 0 \cdot \mathbf{t}_i \in \sum_i A_i$, by **RC$_2$**, $\mathbf{0}$ is also in the set.

$\cap_i A_i$ **RC$_1$**: Trivial since for all i, $A_i \subseteq SN$. **RC$_2$**: Let $\mathbf{t} \in \cap_i A_i$, then for all i, $\mathbf{t} \in A_i$ and so by **RC$_2$** in A_i, Red$(\mathbf{t}) \subseteq A_i$. Thus Red$(\mathbf{t}) \subseteq \cap_i A_i$. **RC$_3$**: Let $\mathbf{t} \in \mathcal{N}$ and Red$(\mathbf{t}) \subseteq \cap_i A$. Then $\forall i$, Red$(\mathbf{t}) \subseteq A_i$, and thus, by **RC$_3$** in A_i, $\mathbf{t} \in A_i$, which implies $\mathbf{t} \in \cap_i A_i$. **RC$_4$**: By **RC$_4$**, $\forall i$, $\mathbf{0} \in A_i$, then $\mathbf{0} \in \cap_i A_i$. $\square$

A *single type valuation* is a partial function from type variables to reducibility candidates, that we define as a sequence of comma-separated mappings, with $\emptyset$ denoting the empty valuation: $\rho :=$ $\emptyset \,|\, \rho, X \mapsto A$. Type variables are interpreted using pairs of single type valuations, that we simply call *valuations*, with common domain: $\rho = (\rho_+, \rho_-)$ with $|\rho_+| = |\rho_-|$. Given a valuation $\rho = (\rho_+, \rho_-)$, the *complementary valuation* $\bar{\rho}$ is the pair (ρ_-, ρ_+). We write $(X_+, X_-) \mapsto (A_+, A_-)$ for the valuation $(X_+ \mapsto A_+, X_- \mapsto A_-)$. A valuation is called *valid* if for all X, $\rho_-(X) \subseteq \rho_+(X)$.

To define the interpretation of a type T, we use the following result.

Corollary 5.3 (of Lemma 3.1). *Any type T has a unique canonical decomposition $T \equiv \sum_{i=1}^n \alpha_i \cdot U_i$ such that for all j, k, $U_j \not\equiv U_k$.*

Proof. By Lemma 3.1, $T \equiv \sum_{i=1}^n \alpha_i \cdot U_i$. Suppose that there exist j, k such that $U_j \equiv U_k$. Then notice that $T \equiv (\alpha_j + \alpha_k) \cdot U_j + \sum_{i \neq j,k} \alpha_i \cdot U_i$. Repeat the process until there is no more j, k such that $U_j \not\equiv U_k$. $\square$

The interpretation $[\![T]\!]_\rho$ of a type T in a valuation $\rho = (\rho_+, \rho_-)$ defined for each free type variable of T is given by: $[\![X]\!]_\rho = \rho_+(X)$, $[\![U \to T]\!]_\rho = [\![U]\!]_{\bar{\rho}} \to [\![T]\!]_\rho$, $[\![\forall X.U]\!]_\rho = \cap_{A \subseteq B \in RC} [\![U]\!]_{\rho,(X_+,X_-) \mapsto (A,B)}$, and if $T \equiv \sum_i \alpha_i \cdot U_i$ is the canonical decomposition of T then $[\![T]\!]_\rho = \sum_i [\![U_i]\!]_\rho$. From Lemma 5.2, the interpretation of any type is a reducibility candidate.

Reducibility candidates deal with closed terms, whereas proving the adequacy lemma by induction requires the use of open terms with some assumptions on their free variables, that will be guaranteed

by a context. Therefore we use *substitutions* σ to close terms: $\sigma := \emptyset \mid (x \mapsto \mathbf{b}; \sigma)$, then $\mathbf{t}_\emptyset = \mathbf{t}$ and $\mathbf{t}_{x \mapsto \mathbf{b}; \sigma} = \mathbf{t}[\mathbf{b}/x]_\sigma$.

Given a context Γ, we say that a substitution σ *satisfies* Γ for the valuation ρ (notation: $\sigma \in \llbracket \Gamma \rrbracket_\rho$) when $(x : U) \in \Gamma$ implies $x_\sigma \in \llbracket U \rrbracket_\rho$ (Note the change in polarity). Let $T \equiv \sum_{i=1}^n \alpha_i \cdot U_i$, such that for all i, j, $U_i \not\equiv U_j$, which always exists by Corollary 5.3. A typing judgement $\Gamma \vdash \mathbf{t} : T$, is said to be *valid* (notation $\Gamma \models \mathbf{t} : T$) if for every valuation ρ, and set of valuations $\{\rho_i\}_n$, where ρ_i acts on $FV(U_i) \setminus FV(\Gamma)$, and for every substitution $\sigma \in \llbracket \Gamma \rrbracket_\rho$, we have $\mathbf{t}_\sigma \in \sum_{i=1}^n \llbracket U_i \rrbracket_{\rho, \rho_i}$.

Lemma 5.4 (Adequacy Lemma). *Every derivable typing judgement is valid: For every valid sequent* $\Gamma \vdash \mathbf{t} : T$, *we have* $\Gamma \models \mathbf{t} : T$. □

Proof. The proof uses a few auxiliary lemmas.

1. Given a (valid) valuation $\rho = (\rho_+, \rho_-)$, for all types T we have $\llbracket T \rrbracket_{\bar\rho} \subseteq \llbracket T \rrbracket_\rho$. *Proof:* Structural induction on T

2. Let $\rho = (\rho_+, \rho_-)$ and $\rho' = (\rho'_+, \rho'_-)$ be two valuations such that $\forall X$, $\rho'_-(X) \subseteq \rho_-(X)$ and $\rho_+(X) \subseteq \rho'_+(X)$. Then for any type T we have $\llbracket T \rrbracket_\rho \subseteq \llbracket T \rrbracket_{\rho'}$ and $\llbracket T \rrbracket_{\bar\rho'} \subseteq \llbracket T \rrbracket_{\bar\rho}$. *Proof:* Structural induction on T.

3. For all reducibility candidates A, $A \subseteq 1 \cdot A$. Moreover, if $\mathbf{b} \in 1 \cdot A$ is a base term, then $\mathbf{b} \in A$. *Proof:* For all $\mathbf{t} \in A$, the term $1 \cdot \mathbf{t} \in 1 \cdot A$. Since $1 \cdot \mathbf{t} \to \mathbf{t}$, we conclude using $\mathbf{RC_2}$. Now, consider $\mathbf{b} \in 1 \cdot A$. We proceed by structural induction on $1 \cdot A$.

4. For all reducibility candidates $\{A_{i,1}\}_{i=1\cdots n_1}$, $\{A_{i,2}\}_{i=1\cdots n_2}$, if $\mathbf{s} \in \sum_{i=1}^{n_1} A_{i,k}$ and $\mathbf{t} \in \sum_{i=1}^{n_2} A_{i,2}$, then $\mathbf{s} + \mathbf{t} \in \sum_{k=1,2, i=1 \cdots n_k} A_{i,k}$. *Proof:* By structural induction on $\sum_{i=1}^{n_1} A_{i,1}$ and $\sum_{i=1}^{n_2} A_{i,2}$.

5. Suppose that $\mathbf{s} \in A \to B$ and $\mathbf{b} \in A$, then $(\mathbf{s}) \mathbf{b} \in B$. *Proof:* Induction on the definition of $A \to B$.

The proof of the adequacy lemma is done by induction on the size of the typing derivation of $\Gamma \vdash \mathbf{t} : T$, relying on these results. □

Theorem 5.5 (Strong normalisation). *If* $\Gamma \vdash \mathbf{t} : T$ *is a valid sequent, then* $\mathbf{t}$ *is strongly normalising.*

Proof. If Γ is the list $(x_i : U_i)_i$, the sequent $\vdash \lambda x_1 \ldots x_n.\mathbf{t} : U_1 \to (\cdots \to (U_n \to T) \cdots)$ is valid. Using Lemma 5.4, we deduce that for any valuation ρ and any substitution $\sigma \in \llbracket \emptyset \rrbracket_\rho$, we have $\sigma(\mathbf{t}) \in \llbracket T \rrbracket_\rho$. By construction, σ does nothing on $\mathbf{t}$: $\sigma(\mathbf{t}) = \mathbf{t}$. Since $\llbracket T \rrbracket_\rho$ is a reducibility candidate, $\lambda x_1 \ldots x_n.\mathbf{t}$ is strongly normalising. Now suppose that $\mathbf{t}$ were not strongly normalising. There would be an infinite rewrite sequence of terms $(\mathbf{t}_i)_i$ starting with $\mathbf{t}$. But then $(\lambda \vec{x}.\mathbf{t}_i)_i$ would then be an infinite rewrite sequence of terms starting with a strongly normalising term: contradiction. Therefore, $\mathbf{t}$ is strongly normalising. □

Corollary 5.6 (Confluence). *If* $\Gamma \vdash s : S$ *is a valid typing judgement and if* $s \to^* r$ *and* $s \to^* t$, *then there exists* s' *such that* $r \to^* s'$ *and* $t \to^* s'$.

Proof. A rewrite system that is both locally confluent and strongly normalising is confluent [12]. □

6   Expressing Matrices and Vectors

In this section we come back to the motivating example introducing the type system and we show how λ^{vec} handles the Hadamard gate, and how to encode matrices and vectors.

With an empty typing context, the booleans $\mathbf{true} = \lambda x.\lambda y.x$ and $\mathbf{false} = \lambda x.\lambda y.y$ can be respectively typed with the types $\mathbb{T} = \forall XY.Y \to (Y \to X)$ and $\mathbb{F} = \forall XY.X \to (Y \to Y)$. The superposition has the following type $\vdash \alpha \cdot \mathbf{true} + \beta \cdot \mathbf{false} : \alpha \cdot \mathbb{T} + \beta \cdot \mathbb{F}$. (Note that it can also be typed with $(\alpha + \beta) \cdot \forall X.X \to X \to X$).

With an empty typing context, the linear map $\mathbf{U}$ sending $\mathbf{true}$ to $a \cdot \mathbf{true} + b \cdot \mathbf{false}$ and $\mathbf{false}$ to $c \cdot \mathbf{true} + d \cdot \mathbf{false}$ is written as $\mathbf{U} = \lambda x.((x)[a \cdot \mathbf{true} + b \cdot \mathbf{false}])[c \cdot \mathbf{true} + d \cdot \mathbf{false}]$. The following sequent is valid: $\vdash \mathbf{U} : \forall X.((I \to (a \cdot \mathbb{T} + b \cdot \mathbb{F})) \to (I \to (c \cdot \mathbb{T} + d \cdot \mathbb{F})) \to X) \to X$.

This is consistent with the discussion in the introduction: the Hadamard gate is the case $a = b = c = \frac{\sqrt{2}}{2}$ and $d = -\frac{\sqrt{2}}{2}$. One can check that with an empty typing context, $\{(\mathbf{U})\ \mathbf{true}\}$ is well typed of type $a \cdot \mathbb{T} + b \cdot \mathbb{F}$, as expected since it reduces to $a \cdot \mathbf{true} + b \cdot \mathbf{false}$.

The term $\{(\mathbf{H})\ \frac{\sqrt{2}}{2} \cdot (\mathbf{true} + \mathbf{false})\}$ is well-typed of type $\mathbb{T} + 0 \cdot \mathbb{F}$. Since the term reduces to $\mathbf{true}$, this is still consistent with the subject reduction: we indeed have $\mathbb{T} \sqsubseteq \mathbb{T} + 0 \cdot \mathbb{F}$.

7   Conclusion

In this paper we define a strongly normalising, confluent, typed, algebraic λ-calculus satisfying a weak subject reduction. The language allows making arbitrary linear combinations of λ-terms $\alpha \cdot \mathbf{t} + \beta \cdot \mathbf{u}$. Its *vectorial* type system is a fine-grained analysis tool describing the "vectorial" properties of typed terms: First, it keeps track of the 'amplitude of a term', *i.e.* if $\mathbf{t}$ and $\mathbf{u}$ both have the same type U, then $\alpha \cdot \mathbf{t} + \beta \cdot \mathbf{u}$ has type $(\alpha + \beta) \cdot U$. Then it keeps track of the 'direction of a term', *i.e.* if $\mathbf{t}$ and $\mathbf{u}$ have types U and V respectively, then $\alpha \cdot \mathbf{t} + \beta \cdot \mathbf{u}$ has type $\alpha \cdot U + \beta \cdot V$. This type system is expressive enough to be able to type the encoding of matrices and vectors.

The resulting type system has the property that if $\Gamma \vdash \mathbf{t} : \sum_i \alpha_i \cdot U_i$ then there exists $\mathbf{t}'$ such that $\mathbf{t} \to^* \mathbf{t}'$ and $\mathbf{t}' = \sum_i \alpha_i \cdot \mathbf{b}_i$, where each $\mathbf{b}_i$ is a basis term of type U_i. Such a $\mathbf{t}'$ is obtained by normalising $\mathbf{t}$ under all rules but the factorisation rules. Within such a $\mathbf{t}'$ there may be subterms of the form $\alpha_1 \cdot \mathbf{b} + \alpha_2 \cdot \mathbf{b}$ of type $\alpha_1 \cdot V_1 + \alpha_2 \cdot V_2$, which are redexes for the factorisation rules. Under our type system, the reduct $(\alpha_1 + \alpha_2) \cdot \mathbf{b}$ can be given both the types $(\alpha_1 + \alpha_2) \cdot V_1$ and $(\alpha_1 + \alpha_2) \cdot V_2$.

The tool we propose in this paper is a first step towards lifting the "quantumness" of algebraic lambda-calculi to the level of a type based analysis. It is also a step towards a "quantum theoretical logic" coming readily with a Curry-Howard isomorphism. The logic we are sketching merges intuitionistic logic and vectorial structure. It results into a novel and intriguing tool.

The next step in the study of the quantumness of the linear algebraic lambda-calculus is the exploration of the notion of orthogonality between terms, and the validation of this notion by means of a compilation into quantum circuits. The work in [15] shows that it is worthwhile pursuing in this direction.

References

[1] Thorsten Altenkirch & Jonathan J. Grattage (2005): *A functional quantum programming language*. In: *Proceedings of LICS-2005*, IEEE Computer Society, pp. 249–258, doi:10.1109/LICS.2005.1. Available at `http://arxiv.org/abs/quant-ph/0409065`.

[2] Pablo Arrighi & Alejandro Díaz-Caro (2011): *Scalar system F for linear-algebraic lambda-calculus: towards a quantum physical logic*. In Bob Coecke, Prakash Panangaden & Peter Selinger, editors: *Proceedings of QPL-2009, Electronic Notes in Theoretical Computer Science* 270/2, Elsevier, pp. 219–229, doi:10.1016/j.entcs.2011.01.033. Available at `http://arxiv.org/abs/0903.3741`.

[3] Pablo Arrighi & Gilles Dowek (2008): *Linear-algebraic lambda-calculus: higher-order, encodings, and confluence*. In Andrei Voronkov, editor: *Proceedings of RTA-2008, Lecture Notes in Computer Science* 5117, Springer, pp. 17–31, doi:10.1007/978-3-540-70590-1_2. Available at `http://arxiv.org/abs/quant-ph/0612199`.

[4] Henk P. Barendregt (1992): *Lambda-calculi with types*. Handbook of Logic in Computer Science II, Oxford University Press.

[5] Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson & Benoît Valiron (2010): *Equivalence of algebraic lambda-calculi*. In: *Proceedings of the 5th International Workshop on Higher-Order Rewriting, HOR-2010*, Edinburgh, UK, pp. 6–11. Available at `http://arxiv.org/abs/1005.2897`.

[6] Alejandro Díaz-Caro & Barbara Petit (2012): *Linearity in the non-deterministic call-by-value setting*. In Luke Ong & Ruy de Queiroz, editors: *Proceedings of WoLLIC-2012, Lecture Notes in Computer Science* 7456, Springer, pp. 216–231. Available at `http://arxiv.org/abs/1011.3542`. To appear.

[7] Thomas Ehrhard & Laurent Regnier (2003): *The differential lambda-calculus*. Theoretical Computer Science 309(1), pp. 1–41, doi:10.1016/S0304-3975(03)00392-X.

[8] Jean-Yves Girard, Yves Lafont & Paul Taylor (1989): *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press. Available at `http://www.paultaylor.eu/stable/Proofs+Types.html`.

[9] Jean-Pierre Jouannaud & Hélène Kirchner (1986): *Completion of a set of rules modulo a set of equations*. SIAM Journal on Computing 15(4), pp. 1155–1194, doi:10.1145/800017.800519.

[10] Jean-Louis Krivine (1990): *Lambda-calcul: Types et Modèles*. Études et Recherches en Informatique, Masson.

[11] Christine Tasson (2009): *Algebraic totality, towards completeness*. In Pierre-Louis Curien, editor: *Proceedings of TLCA-2009, Lecture Notes in Computer Science* 5608, Springer, pp. 325–340, doi:10.1007/978-3-642-02273-9_24. Available at `http://arxiv.org/abs/0912.2349`.

[12] TeReSe (2003): *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science 55, Cambridge University Press.

[13] The Coq Development Team (2010): *Reference Manual*, 8.3 edition. INRIA. Available at `http://coq.inria.fr/doc`.

[14] André van Tonder (2004): *A Lambda-calculus for quantum computation*. SIAM Journal of Computing 33, pp. 1109–1135, doi:10.1137/S0097539703432165. Available at `http://arxiv.org/abs/quant-ph/0307150`.

[15] Benoît Valiron (2010): *Orthogonality and algebraic lambda-calculus*. In: *Proceedings of the 7th International Workshop on Quantum Physics and Logic, QPL-2010*, Oxford, UK, pp. 169–175. Available at `http://www.cs.ox.ac.uk/people/bob.coecke/QPL_proceedings.html`.

[16] Benoît Valiron (2011): *Local confluence of the algebraic fragment: proof in COQ*. Available at `http://www.monoidal.net/vectorial-lvec-coqproof.tar.bz2`.

[17] Lionel Vaux (2009): *The algebraic lambda-calculus*. Mathematical Structures in Computer Science 19(5), pp. 1029–1059, doi:10.1017/S0960129509990089. Available at `http://hal.archives-ouvertes.fr/hal-00379750`.