Lineal: A linear-algebraic lambda-calculus

Pablo Arrighi, Gilles Dowek

Lineal: A linear-algebraic λ-calculus

Pablo Arrighi[a,1], Gilles Dowek[c]

[a] *Université de Grenoble, Laboratoire LIG, UMR 5217, 220 rue de la Chimie, 38400 Saint-Martin d'Hères, France.*
[b] *Université de Lyon, Laboratoire LIP, UMR 5668, 46 allée d'Italie 69007 Lyon, France.*
[c] *INRIA, 23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France.*

Abstract

We provide a computational definition of the notions of vector space and bilinear functions. We use this result to introduce a minimal language combining higher-order computation and linear algebra. This language extends the λ-calculus with the possibility to make arbitrary linear combinations of terms $\alpha.\mathbf{t} + \beta.\mathbf{u}$. We describe how to "execute" this language in terms of a few rewrite rules, and justify them through the two fundamental requirements that the language be a language of linear operators, and that it be higher-order. We mention the perspectives of this work in the field of quantum computation, whose circuits we show can be easily encoded in the calculus. Finally, we prove the confluence of the entire calculus.

1. Motivations

Knuth and Bendix have described a method to transform an equational theory into a rewrite system [35]. In this paper, we show that this can be achieved for the theory of vector spaces. This yields a computational definition of the notion of vector space. We then use this definition to merge at a fundamental level higher-order computation in its simplest and most general form, the untyped λ-calculus, together with linear algebra. We see this *Linear-algebraic λ-calculus* (also referred to as *Lineal* for short) as a platform for various applications, such as non-deterministic, probabilistic and quantum computation — each of these applications probably requiring their own type systems.

This journal paper integrates the contributions from three already published conference papers [5, 6, 7]. There has been a number of recent works surrounding these papers, whose presentation we postpone till Section 8. The emphasis of the present introduction is on the original motivations behind this calculus; in the same way that the emphasis of the present paper is on providing an integrated, coherent, comprehensive presentation of the calculus without further add-ons.

Email addresses: `pablo.arrighi@imag.fr` (Pablo Arrighi), `gilles.dowek@inria.fr` (Gilles Dowek)

1.1. Quantum programming languages

Over the last two decades, the discovery of several great algorithmic results [22, 52, 33] has raised important expectations in the field of quantum computation. Somewhat surprisingly however these results have been expressed in the primitive model of quantum circuits – a situation which is akin to that of classical computation in the 1950s. Over the last few years a number of researchers have sought to develop quantum programming languages as a consequence. Without aiming to be exhaustive and in order to understand where the perspectives of this work come in, it helps to classify these proposals according to "how classical" versus "how quantum" they are [50]. There are two ways a quantum mechanical system may evolve: according to a unitary transformation or under a measurement. The former is often thought of as "purely quantum": it is deterministic and will typically be used to obtain quantum superpositions of base vectors. The latter is probabilistic in the classical sense, and will typically be used to obtain some classical information about a quantum mechanical system, whilst collapsing the system to a mere base vector.

Note that these are only typical uses: it is well-known that one can simulate any unitary transformation by series of generalized measures on the one hand, and reduce all measures to a mere projection upon the canonical basis at the end of a computation on the other hand. It remains morally true nonetheless that measurement-based models of quantum computation tend to hide quantum superpositions behind a classical interface, whilst the unitary-based models of quantum computation tend to consider quantum superpositions as legitimate expressions of the language, and sometimes even seek to generalize their effects to control flow.

Therefore one may say that measurement-based models of quantum computation – whether reliant upon teleportation [40], state transfer [43] or more astonishingly graph states [46] – lie on one extreme, as they keep the "quantumness" to a minimum.

A more balanced approach is to allow for both unitary transformations and quantum measurements. Such models can be said to formalize the existing algorithm description methods to a strong extent: they exhibit quantum registers upon which quantum circuits may be applied, together with classical registers and programming structures in order to store measurements results and control the computation [48]. For this reason they are the more practical route to quantum programming. Whilst this juxtaposition of "quantum data, classical control" has appeared ad-hoc and heterogeneous at first, functional-style approaches together with linear type systems [50, 4] have ended up producing elegant quantum programming languages.

Finally we may evacuate measures altogether – leaving them till the end of the computation and outside the formalism. This was the case for instance in [55, 56], but here the control structure remained classical.

In our view, such a language becomes even more interesting once we have also overcome the need for any additional classical registers and programming structures, and aim to draw the full consequence of quantum mechanics: "quantum data, quantum control". After all, classical control can be viewed as a particular case of quantum control. Moreover, avoiding this distinction leads to a simpler language, exempt of the separation between classical and quantum expressions. Finally recent results suggest that quantum control may turn out to be more efficient that classical control in the presence of Black-box algorithms [41, 18].

Quantum Turing Machines [13], for instance, lie on this other extreme, since the entire machine can be in a superposition of base vectors. Unfortunately they are a rather oblivious way to describe an algorithm. Functional-style control structure, on the other hand, seem to merge with quantum evolution descriptions in a unifying manner. The functional language we describe may give rise to a "purely quantum" programming language, but only once settled the question of restricting to unitary operators. This language is exempt of classical registers, classical control structure, measurements, and allows arbitrary quantum superpositions of base vectors.

A survey and comparison of these quantum programming languages can be found in [31].

1.2. Current status of the language

In our view, the problem of formulating a language of higher-order computable operators upon infinite dimensional vector spaces was the first challenge that needed to be met, before even aiming to have a physically executable language. In the current state of affairs computability in vector spaces is dealt with matrices and compositions, and hence restricted to finite-dimensional systems – although this limitation is sometimes circumvented by introducing an extra classical control structure e.g. via the notions of uniform circuits or linear types. The language we provide achieves this goal of a minimal calculus for describing higher-order computable linear operators in a wide sense. Therefore this work may serve as a basis for studying wider notions of computability upon abstract vector spaces, whatever the interpretation of the vector space is (probabilities, number of computational paths leading to one result,. . .).

The downside of this generality as far as the previously mentioned application to quantum computation are concerned is that our operators are not restricted to being unitary. A further step towards specializing our language to quantum computation would be to restrict to unitary operators, as required by quantum physics. There may be several ways to do so. A first lead would be to design an a posteriori static analysis that enforces unitarity – exactly like typability is not wired in pure lambda-calculus, but may be enforced a posteriori. A second one would be to require a formal unitarity proof from the programmer. With a term and a unitarity proof, we could derive a more standard representation of the operator, for instance in terms of a universal set of quantum gates [17]. This transformation may be seen as part of a compilation process.

In its current state, our language can be seen as a specification language for quantum programs, as it possesses several desirable features of such a language: it allows a high level description of algorithms without any commitment to a particular architecture, it allows the expression of black-box algorithms through the use of higher-order functionals, its notation remains close to both linear algebra and functional languages.

Besides quantum computing, this work may bring contributions to other fields, which we now develop.

1.3. Logics, calculi

In this article linearity is understood in the sense of linear algebra, which is not to be confused with linearity in the sense of Linear λ-calculus [1]. It may help the reader to draw attention to this distinction: Linear λ-calculus is a calculus whose types are formulae of Linear Logic [32]. In such a λ-calculus, one distinguishes linear resources, which may be neither duplicated nor discarded, from nonlinear ones, whose fate is not subjected to particular restrictions. The Linear-algebraic λ-calculus we describe does have some resemblances with the Linear λ-calculus, as well as some crucial, strongly motivated differences. Duplication of a term $\mathbf{t}$ is again treated cautiously, but in a different way: only terms expressing base vectors can be duplicated, which is compatible with linear algebra. As we shall see, terms of the form $\lambda\mathbf{x}\ \mathbf{u}$ are always base vectors. As a consequence, even when a term $\mathbf{t}$ cannot be duplicated the term $\lambda\mathbf{x}\ \mathbf{t}$ can. Since the term $\lambda\mathbf{x}\ \mathbf{t}$ is a function building the term $\mathbf{t}$, it can be thought of as a description of $\mathbf{t}$. (This suggests some possible connections between the abstraction $\lambda\mathbf{x}$, the ! bang operator of linear lambda-calculus and the $'$ quote operator that transforms a term into a description of it as used for instance in LISP.)
Again in connection with Linear Logic, Vaux has proposed an *Algebraic λ-calculus* [57] independently and simultaneously [5, 6, 7] with ours, and which is similar in the sense that it exhibits linear combinations of terms, and different in both the reduction strategy and the set of scalars considered. We will say more about this in Section 8. His work is both a restriction (less operators) and a generalization (positive reals) of Ehrhard and Regnier's differential λ-calculus [26].
The functional style of programming is based on the λ-calculus together with a number of extensions, so as to make everyday programming more accessible. Hence, since the birth of functional programming there has been several theoretical studies of extensions of the λ-calculus in order to account for basic arithmetic (see for instance Dougherty's algebraic extension [25] for normalising terms of the λ-calculus). *Lineal* could again be viewed as just an extension of the λ-calculus in order to handle operations over vector spaces, and make everyday programming more accessible upon them. The main difference in approach is that here the λ-calculus is not seen as a control structure which sits on top of the vector space data structure, controlling which operations to apply and when. Rather, the λ-calculus terms themselves can be summed and weighted, hence they actually are the basis of the vector space... upon which they can also act. This intertwining of concepts is essential if seeking to represent parallel or probabilistic computation as it is the computation itself which must be endowed with a vector space structure. The ability to superpose λ-calculus terms in that sense takes us back to Boudol's parallel λ-calculus [15], and may also be viewed as taking part of a wave of probabilistic extensions of calculi, e.g.[16, 34, 30].

1.4. Confluence techniques

A standard way to describe how a program is executed is to give a small step operational semantic for it, in the form of a finite set rewrite rules which gradually transform a program into a value. The main theorem proved in this paper is the confluence of our language. What this means is that the order in which those transformations are applied does not affect the end result of the computation. Confluence results are milestones in the study of programming languages and more generally in the theory of rewriting. Our proof uses many of the theoretical tools that have been developed for confluence proofs in a variety of fields (local confluence and Newman's lemma; strong confluence and the Hindley-Rosen lemma) as well as the avatar lemma for parametric rewriting as introduced in [5]. These are fitted together in an elaborate architecture which may have its own interest whenever one seeks to merge a non-terminating conditional confluent rewrite system together with a terminating conditional confluent rewrite system.

1.5. Outline

Section 2 develops a computational definition of vector spaces and bilinear functions. This is achieved by taking the axioms of vector spaces and orienting them. Section 3 explains how to have a rewrite system for scalars that are enough to account for quantum computation. Section 4 presents the designing principles of the language, Section 5 formally describes the Linear-algebraic λ-calculus and its semantics. Section 6 shows that the language is expressive enough for classical and quantum computations. These are the more qualitative sections of the paper. We chose to postpone till Section 7 the various proofs of confluence, as they are more technical. Section 8 will be the opportunity to provide an overview of the most recent contributions surrounding this work. Section 9 provides a summary and some perspectives.

2. Computational vector spaces and bilinear functions

One way to prove the equality of two vectors expressed by terms such as $2.\mathbf{x} + \mathbf{y} + 3.\mathbf{x}$ and $5.(\mathbf{x} + \mathbf{y}) + (-4).\mathbf{y}$ is to transform these terms into linear combinations of the unknowns and check that the terms obtained this way are the same. This algorithm transforming a term expressing a vector into a linear combination of the unknowns is also useful to express the operational semantic of programming languages for quantum computing, because in such languages a program and its input value form a term expressing a vector whose value, the output, is a linear combination of constants. More generally, several algorithms used in linear algebra, such as matrix multiplication algorithms, transform a term expressing a vector with various constructs into a linear combination of constants.

The algorithm transforming a term expressing a vector into a linear combination of the unknowns is valid in all vector spaces. The goal of this Section is to show that, moreover, it completely defines the notion of vector space. This computational definition of the notion of vector space can be extended to define other algebraic notions such as bilinearity.

2.1. Algorithms and models

In this paper rewriting systems play a double role: they serve to provide an oriented version of the notion of vector space, and to provide a small step operational semantics for the introduced language. We now provide the standard definitions about them.

Definition 1. (Rewriting) *Let $\mathcal{L}$ be a first-order language. A rewrite system X on $\mathcal{L}$ is given by a finite set of rules of the form $l \longrightarrow r$. We define the relation $\longrightarrow_X$ as follows: $t \longrightarrow_X u$ if and only if there is an occurrence α in the term t, a rewrite rule $l \longrightarrow r$ in X, and a substitution σ such that $t_{|\alpha} = \sigma l$ and $u = t[\sigma r]_\alpha$, where $t_{|\alpha}$ is the subterm of t at occurrence α, and $t[v]_\alpha$ is the graft of v in t occurrence α.*

Definition 2. (AC-Rewriting) *Let $\mathcal{L}$ be a first-order language. A AC-rewrite system X on $\mathcal{L}$ is given by binary function symbols $f_1, \dots, f_n$ of the language and a finite set of rules of the form $l \longrightarrow r$. We define the relation $=_{AC}$ as the congruence generated by the associativity and commutativity axioms of the symbols $f_1, \dots, f_n$. We define the relation $\longrightarrow_X$ as follows: $t \longrightarrow_X u$ if and only if there exists a term t' such that $t =_{AC} t'$, an occurrence α in t', a rewrite rule $l \longrightarrow r$ in X and a substitution σ such that $t'_{|\alpha} = \sigma l$ and $u =_{AC} t'[\sigma r]_\alpha$.*

Definition 3. (Algebra) *Let $\mathcal{L}$ be a first-order language. An* $\mathcal{L}$-algebra *is a family formed by a set M and for each symbol f of $\mathcal{L}$ of arity n, a function $\hat{f}$ from M^n to M. The denotation $[\![t]\!]_\phi$ of a term t for an assignment ϕ is defined as usual: $[\![x]\!]_\phi = \phi(x)$ and $[\![f(t_1, \dots, t_n)]\!]_\phi = \hat{f}([\![t_1, \dots, t_n]\!])$.*

Definition 4. (Model of a rewrite system) *Let $\mathcal{L}$ be a first-order language and X an algorithm defined by a rewrite system on terms of the language $\mathcal{L}$. An* $\mathcal{L}$-algebra *$\mathcal{M}$ is a* model *of the algorithm X, or the algorithm X is* valid *in the model $\mathcal{M}$, ($\mathcal{M} \models X$) if for all rewrite rules $l \longrightarrow r$ of the rewrite system and for all valuations ϕ, $[\![l]\!]_\phi = [\![r]\!]_\phi$.*

Example 1. *Consider the language $\mathcal{L}$ formed by two binary symbols $+$ and $\times$ and the algorithm X defined by the distributivity rules*

$$(x + y) \times z \longrightarrow (x \times z) + (y \times z)$$

$$x \times (y + z) \longrightarrow (x \times y) + (x \times z)$$

transforming for instance, the term $(a + a) \times a$ to the term $a \times a + a \times a$. The algebra $\langle \{0, 1\}, \min, \max \rangle$ is a model of this algorithm.

Remark 1. *This definition of the validity of an algorithm in a model is strongly related with denotational semantics, as rewriting systems could also be seen as programs, and the algebraic structure as a denotational semantics.*

Definition 5. (Model of an AC-rewrite system) *Let $\mathcal{L}$ be a first-order language. Let X be a AC-rewrite system. An $\mathcal{L}$-*algebra *$\mathcal{M}$ is a* model *of the AC-rewrite system X ($\mathcal{M} \models R$) if*

- *for all rewrite rules $l \longrightarrow r$ of X and for all valuations ϕ, $[\![l]\!]_\phi = [\![r]\!]_\phi$,*
- *for all AC-symbol f of X and for all valuations ϕ and indices i*

$$[\![f(x, f(y, z))]\!]_\phi = [\![f(f(x, y), z)]\!]_\phi$$

$$[\![f(x, y)]\!]_\phi = [\![f(y, x)]\!]_\phi$$

As a consequence if $t \longrightarrow_X^* u$ then for all ϕ, $[\![t]\!]_\phi = [\![u]\!]_\phi$.

2.2. Vector spaces: an algorithm

Let $\mathcal{L}$ be a 2-sorted language with a sort K for scalars and a sort E for vectors containing two binary symbols $+$ and $\times$ of rank $\langle K, K, K\rangle$, two constants 0 and 1 of sort K, a binary symbol, also written $+$, of rank $\langle E, E, E\rangle$, a binary symbol . of rank $\langle K, E, E\rangle$ and a constant $\mathbf{0}$ of sort E.

To transform a term of sort E into a linear combination of the unknowns, we want to develop sums of vectors: $\alpha.(\mathbf{u} + \mathbf{v}) \longrightarrow \alpha.\mathbf{u} + \alpha.\mathbf{v}$, but factor sums of scalars and nested products: $\alpha.\mathbf{u} + \beta.\mathbf{u} \longrightarrow (\alpha + \beta).\mathbf{u}$, $\alpha.(\beta.\mathbf{u}) \longrightarrow (\alpha \times \beta).\mathbf{u}$. We also need the trivial rules $\mathbf{u} + \mathbf{0} \longrightarrow \mathbf{u}$, $0.\mathbf{u} \longrightarrow \mathbf{0}$ and $1.\mathbf{u} \longrightarrow \mathbf{u}$. Finally, we need three more rules for confluence $\alpha.\mathbf{0} \longrightarrow \mathbf{0}$, $\alpha.\mathbf{u} + \mathbf{u} \longrightarrow (\alpha + 1).\mathbf{u}$, $\mathbf{u} + \mathbf{u} \longrightarrow (1 + 1).\mathbf{u}$. As we want to be able to apply the factorization rule to a term of the form $(3.\mathbf{x} + 4.\mathbf{y}) + 2.\mathbf{x}$, reductions in the above rewrite system must be defined modulo the associativity and commutativity of $+$. This leads to the following definition.

Definition 6. (The rewrite system V) *The rewrite system V is the AC-rewrite system where the only AC-symbol is $+$ and the rules are*

$$\mathbf{u} + \mathbf{0} \longrightarrow \mathbf{u}$$

$$0.\mathbf{u} \longrightarrow \mathbf{0}$$

$$1.\mathbf{u} \longrightarrow \mathbf{u}$$

$$\alpha.\mathbf{0} \longrightarrow \mathbf{0}$$

$$\alpha.(\beta.\mathbf{u}) \longrightarrow (\alpha.\beta).\mathbf{u}$$

$$\alpha.\mathbf{u} + \beta.\mathbf{u} \longrightarrow (\alpha + \beta).\mathbf{u}$$

$$\alpha.\mathbf{u} + \mathbf{u} \longrightarrow (\alpha + 1).\mathbf{u}$$

$$\mathbf{u} + \mathbf{u} \longrightarrow (1 + 1).\mathbf{u}$$

$$\alpha.(\mathbf{u} + \mathbf{v}) \longrightarrow \alpha.\mathbf{u} + \alpha.\mathbf{v}$$

To be complete, we should also transform the axioms of the theory of fields into a rewrite system, which is known to be impossible as there is no equational description of fields as a consequence of Birkhoff's HSP theorem and the fact that the class of fields is not closed under direct product [14].

We could switch to the theory of modules and use the fact that the axioms of the theory of rings can be transformed into a rewrite system.

An alternative is to provide term rewrite systems for specific rings or fields such as $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Q}(i, \sqrt{2})$, etc. as we shall do in Section 3. Notice that these rewrite system are in general richer than that of the theory of rings. For instance in the language of the rewrite system of $\mathbb{Q}(i, \sqrt{2})$, we have terms expressing the numbers $1/2$ or $\sqrt{2}$ that are not in the generic language of rings.

Thus we shall introduce a general notion of "scalar rewrite system" and consider an arbitrary such system. Basically the notion of a scalar rewrite systems lists the few basic properties that scalars are usually expected to have: neutral elements, associativity of $+$, etc.

Definition 7. (Scalar rewrite system) *A* scalar rewrite system *is a rewrite system on a language containing at least the symbols $+$, $\times$, 0 and 1 such that:*

- *S is terminating and ground confluent,*
- *for all closed terms α, β and γ, the pair of terms*
  - *$0 + \alpha$ and α,*
  - *$0 \times \alpha$ and 0,*
  - *$1 \times \alpha$ and α,*
  - *$\alpha \times (\beta + \gamma)$ and $(\alpha \times \beta) + (\alpha \times \gamma)$,*
  - *$(\alpha + \beta) + \gamma$ and $\alpha + (\beta + \gamma)$,*
  - *$\alpha + \beta$ and $\beta + \alpha$,*
  - *$(\alpha \times \beta) \times \gamma$ and $\alpha \times (\beta \times \gamma)$,*
  - *$\alpha \times \beta$ and $\beta \times \alpha$*

  have the same normal forms,
- *0 and 1 are normal terms.*

Later in Subsection 7.4 we shall prove that for any such scalar rewrite system S, $S \cup V$ is terminating and confluent.

Proposition 1. *Let $\mathbf{t}$ be a normal term whose variables are among $\mathbf{x}_1, ..., \mathbf{x}_n$. The term $\mathbf{t}$ is $\mathbf{0}$ or a term of the form $\alpha_1.\mathbf{x}_{i_1} + ... + \alpha_k.\mathbf{x}_{i_k} + \mathbf{x}_{i_{k+1}} + ... + \mathbf{x}_{i_{k+l}}$ where the indices $i_1, ..., i_{k+l}$ are distinct and $\alpha_1, ..., \alpha_k$ are neither 0 nor 1.*

Proof. The term $\mathbf{t}$ is a sum $\mathbf{u}_1 + ... + \mathbf{u}_n$ of normal terms that are not sums (we take $n = 1$ if $\mathbf{t}$ is not a sum).

A normal term that is not a sum is either $\mathbf{0}$, a variable, or a term of the form $\alpha.\mathbf{v}$. In this case, α is neither 0 nor 1 and $\mathbf{v}$ is neither $\mathbf{0}$, nor a sum of two vectors nor a product of a scalar by a vector, thus it is a variable.

As the term $\mathbf{t}$ is normal, if $n > 1$ then none of the $\mathbf{u}_i$ is $\mathbf{0}$. Hence, the term $\mathbf{t}$ is either $\mathbf{0}$ or a term of the form

$$\alpha_1.\mathbf{x}_{i_1} + ... + \alpha_k.\mathbf{x}_{i_k} + \mathbf{x}_{i_{k+1}} + ... + \mathbf{x}_{i_{k+l}}$$

where $\alpha_1, ..., \alpha_k$ are neither 0 nor 1. As the term $\mathbf{t}$ is normal, the indices $i_1, ..., i_{k+l}$ are distinct.

2.3. Vector spaces: a computational characterization

With respect to the notion of model, algorithms play the same role as sets of axioms: an algorithm may or may not be valid in a model, exactly like a set of axioms may or may not be valid in a model.

The notion of validity may be used to study sets of axioms, typically building a model is a way to prove that some proposition is not provable from a set of axioms. But validity can also be used in the other direction: to define algebraic structures as models of some theories. For instance, given a field $\mathcal{K} = \langle K, +, \times, 0, 1\rangle$ the class of $\mathcal{K}$-vector spaces can be defined as follows.

Definition 8. (Vector space) *The algebra $\langle E, +, ., \mathbf{0}\rangle$ is a $\mathcal{K}$-vector space if and only if $\mathcal{K} = \langle K, +, \times, 0, 1\rangle$ is a field and the algebra $\langle K, +, \times, 0, 1, E, +, ., \mathbf{0}\rangle$ is a model of the 2-sorted set of axioms*

$$\forall \mathbf{u} \forall \mathbf{v} \forall \mathbf{w}\ ((\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w}))$$

$$\forall \mathbf{u} \forall \mathbf{v}\ (\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u})$$

$$\forall \mathbf{u}\ (\mathbf{u} + \mathbf{0} = \mathbf{u})$$

$$\forall \mathbf{u}\ \exists \mathbf{u}'\ (\mathbf{u} + \mathbf{u}' = \mathbf{0})$$

$$\forall \mathbf{u}\ (1.\mathbf{u} = \mathbf{u})$$

$$\forall \alpha \forall \beta \forall \mathbf{u}\ (\alpha.(\beta.\mathbf{u}) = (\alpha.\beta).\mathbf{u})$$

$$\forall \alpha \forall \beta \forall \mathbf{u}\ ((\alpha + \beta).\mathbf{u} = \alpha.\mathbf{u} + \beta.\mathbf{u})$$

$$\forall \alpha \forall \mathbf{u} \forall \mathbf{v}\ (\alpha.(\mathbf{u} + \mathbf{v}) = \alpha.\mathbf{u} + \alpha.\mathbf{v})$$

We now prove that, the class of $\mathcal{K}$-vector spaces can be defined as the class of models of the rewrite system V.

Proposition 2. *Let $\mathcal{K} = \langle K, +, \times, 0, 1\rangle$ be a field. The algebra $\langle E, +, ., \mathbf{0}\rangle$ is a $\mathcal{K}$-vector space if and only if the algebra $\langle K, +, \times, 0, 1, E, +, ., \mathbf{0}\rangle$ is a model of the rewrite system V.*

Proof. We first check that all the rules of V and the associativity and commutativity of addition are valid in all vector spaces. All of them are trivial except $\alpha.\mathbf{u}+\mathbf{u}=(\alpha+1).\mathbf{u}$, $\mathbf{u}+\mathbf{u}=(1+1).\mathbf{u}$, $0.\mathbf{u}=\mathbf{0}$ and $\alpha.\mathbf{0}=\mathbf{0}$. The first and second are consequence of $1.\mathbf{u}=\mathbf{u}$ and $\alpha.\mathbf{u}+\beta.\mathbf{u}=(\alpha+\beta).\mathbf{u}$. To prove the third let $\mathbf{u}'$ be such that $\mathbf{u}+\mathbf{u}'=\mathbf{0}$. We have $0.\mathbf{u}=0.\mathbf{u}+\mathbf{0}=0.\mathbf{u}+\mathbf{u}+\mathbf{u}'=0.\mathbf{u}+1.\mathbf{u}+\mathbf{u}'=1.\mathbf{u}+\mathbf{u}'=\mathbf{u}+\mathbf{u}'=\mathbf{0}$. The last one is a consequence of $0.\mathbf{u}=\mathbf{0}$ and $\alpha.(\beta.\mathbf{u})=(\alpha.\beta).\mathbf{u}$.

Conversely, we prove that all axioms of vector spaces are valid in all models of V. The validity of each of them is a consequence of the validity of a rewrite rule, except $\forall\mathbf{u}\exists\mathbf{u}'\ (\mathbf{u}+\mathbf{u}'=\mathbf{0})$ that is a consequence of $\mathbf{u}+(-1).\mathbf{u}=\mathbf{0}$ itself being a consequence of $\alpha.\mathbf{u}+\beta.\mathbf{u}=(\alpha+\beta).\mathbf{u}$ and $0.\mathbf{u}=\mathbf{0}$.

2.4. Vector spaces: decidability

We now show that the rewrite system V (Definition 6) permits to prove the decidability of the word problem (i.e., whether two terms express the same vector or not) for vector spaces.

Definition 9. *The* decomposition *of* $\mathbf{t}$ *along* $\mathbf{x}_1, ..., \mathbf{x}_n$ *is the sequence* $\alpha_1, ..., \alpha_n$ *such that if there is a subterm of the form* $\alpha.\mathbf{x}_i$ *in* $\mathbf{t}$*, then* $\alpha_i=\alpha$*, if there is a subterm of the form* $\mathbf{x}_i$ *in* $\mathbf{t}$*, then* $\alpha_i=1$*, and* $\alpha_i=0$ *otherwise.*

Proposition 3. *Let* $\mathbf{t}$ *and* $\mathbf{u}$ *be two terms whose variables are among* $\mathbf{x}_1, ..., \mathbf{x}_n$*. The following propositions are equivalent:*

(i) *the normal forms of* $\mathbf{t}$ *and* $\mathbf{u}$ *are identical modulo AC,*
(ii) *the equation* $\mathbf{t}=\mathbf{u}$ *is valid in all* $\mathcal{K}$*-vector spaces,*
(iii) *and the denotation of* $\mathbf{t}$ *and* $\mathbf{u}$ *in* K^n *for the assignment* $\phi=\mathbf{e}_1/\mathbf{x}_1, ..., \mathbf{e}_n/\mathbf{x_n}$*, where* $\mathbf{e}_1, ..., \mathbf{e}_n$ *is the canonical base of* K^n*, are identical.*

Proof. Proposition (i) implies proposition (ii) and proposition (ii) implies proposition (iii). Let us prove that proposition (iii) implies proposition (i).

Let $\mathbf{t}$ be a normal term whose variables are among $\mathbf{x}_1, ..., \mathbf{x}_n$. Assume $[\![\mathbf{t}]\!]_\phi=[\![\mathbf{u}]\!]_\phi$. Let $\mathbf{e}_1, ..., \mathbf{e}_n$ be the canonical base of K^n and $\phi=\mathbf{e}_1/\mathbf{x}_1, ..., \mathbf{e}_n/\mathbf{x}_n$. Call $\alpha_1, ..., \alpha_n$ the coordinates of $[\![\mathbf{t}]\!]_\phi$ in $\mathbf{e}_1, ..., \mathbf{e}_n$. Then the decompositions of the normal forms of $\mathbf{t}$ and $\mathbf{u}$ are both $\alpha_1, ..., \alpha_n$ and thus they are identical modulo AC.

2.5. Summary

We usually define an algebraic structure as an algebra $\langle M, \hat{f}_1, \dots, \hat{f}_n\rangle$ that validates some propositions. For instance $\mathcal{K}$-vector spaces are defined as the algebras $\langle E, \mathbf{0}, +, .\rangle$ that validate the equations of Definition 8.

We can, in a more computation-oriented way, define an algebraic structure as an algebra that validates an algorithm on terms constructed upon these operations. For instance $\mathcal{K}$-vector spaces are are defined as the algebras $\langle E, \mathbf{0}, +, .\rangle$ that validate the algorithm V of Definition 6.

This algorithm is a well-known algorithm in linear algebra: it is the algorithm that transforms any linear expression into a linear combination of the unknowns.

If we chose a base, as will be the case in section 5, this algorithm may be used to transforms any linear expression into a linear combination of base vectors. Still the algorithm itself is not linked to any particular base and it may even be used if the unknowns represent a linearly dependent family.

This algorithm is, at a first look, only one among the many algorithms used in linear algebra, but it completely defines the notion of vector space: a vector space is any algebra where this algorithm is valid, it is any algebra where linear expressions can be transformed this way into linear combinations of the unknowns.

2.6. Bilinearity

Another important notion about vector spaces is that of bilinear functions. For instance the tensor product, matrix multiplication, the inner product and as we shall see the application in Lineal are all bilinear operations. The method we developed for a computational characterization of vector spaces extends to this notion:

Definition 10. (Bilinear function) *Let E, F, and G be three vector spaces on the same field. A function $\otimes$ from $E \times F$ to G is said to be* bilinear *if*

$$(\mathbf{u} + \mathbf{v}) \otimes \mathbf{w} = (\mathbf{u} \otimes \mathbf{w}) + (\mathbf{v} \otimes \mathbf{w})$$
$$(\alpha.\mathbf{u}) \otimes \mathbf{v} = \alpha.(\mathbf{u} \otimes \mathbf{v})$$
$$\mathbf{u} \otimes (\mathbf{v} + \mathbf{w}) = (\mathbf{u} \otimes \mathbf{v}) + (\mathbf{u} \otimes \mathbf{w})$$
$$\mathbf{u} \otimes (\alpha.\mathbf{v}) = \alpha.(\mathbf{u} \otimes \mathbf{v})$$

Definition 11. (Tensor product) *Let E and F be two vector spaces, the pair formed by the vector space G and the bilinear function from $E \times F$ to G is a* tensor product *of E and F if for all bases $(\mathbf{e}_i)_{i \in I}$ of E and $(\mathbf{e'}_j)_{j \in J}$ of F the family $(\mathbf{e}_i \otimes \mathbf{e'}_j)_{\langle i,j \rangle}$ is a base of G.*

The corresponding algorithm is as follows:

Definition 12. (The rewrite system V') *Consider a language with four sorts: K for scalars and E, F, and G for the vectors of three vector spaces, the symbols $+$, $\times$, 0, 1 for scalars, three copies of the symbols $+$, $.$ and $\mathbf{0}$ for each sort E, F, and G and a symbol $\otimes$ of rank $\langle E, F, G \rangle$.*

The system V' is the rewrite system formed by three copies of the rules of the system V and the rules

$$(\mathbf{u} + \mathbf{v}) \otimes \mathbf{w} \longrightarrow (\mathbf{u} \otimes \mathbf{w}) + (\mathbf{v} \otimes \mathbf{w})$$
$$(\alpha.\mathbf{u}) \otimes \mathbf{v} \longrightarrow \alpha.(\mathbf{u} \otimes \mathbf{v})$$
$$\mathbf{u} \otimes (\mathbf{v} + \mathbf{w}) \longrightarrow (\mathbf{u} \otimes \mathbf{v}) + (\mathbf{u} \otimes \mathbf{w})$$
$$\mathbf{u} \otimes (\alpha.\mathbf{v}) \longrightarrow \alpha.(\mathbf{u} \otimes \mathbf{v})$$
$$\mathbf{0} \otimes \mathbf{u} \longrightarrow \mathbf{0}$$
$$\mathbf{u} \otimes \mathbf{0} \longrightarrow \mathbf{0}$$

Later in Subsection 7.4 we shall prove that for any such scalar rewrite system S, $S \cup V'$ is terminating and confluent.

Propositions 1-3 generalize easily.

Proposition 4. *Let* $\mathbf{t}$ *be a normal term whose variables of sort* E *are among* $\mathbf{x}_1, ..., \mathbf{x}_n$*, whose variables of sort* F *are among* $\mathbf{y}_1, ..., \mathbf{y}_p$*, and that has no variables of sort* G *and* K*. If* $\mathbf{t}$ *has sort* E *or* F*, then it has the same form as in Proposition 1. If it has sort* G*, then it has the form*

$$\alpha_1.(\mathbf{x}_{i_1} \otimes \mathbf{y}_{j_1}) + ... + \alpha_k.(\mathbf{x}_{i_k} \otimes \mathbf{y}_{j_k}) + (\mathbf{x}_{i_{k+1}} \otimes \mathbf{y}_{j_{k+1}}) + ... + (\mathbf{x}_{i_{k+l}} \otimes \mathbf{y}_{j_{k+l}})$$

where the pairs of indices $\langle i_1, j_1\rangle, ..., \langle i_{k+l}, j_{k+l}\rangle$ *are distinct and* $\alpha_1, ..., \alpha_k$ *are neither* 0 *nor* 1*.*

Proposition 5. *Let* $\mathcal{K} = \langle K, +, \times, 0, 1\rangle$ *be a field. The structures* $\langle E, +, ., \mathbf{0}\rangle$*,* $\langle F, +, ., \mathbf{0}\rangle$*,* $\langle G, +, ., \mathbf{0}\rangle$ *are* $\mathcal{K}$*-vector spaces and* $\otimes$ *is a bilinear function from* $E \times F$ *to* G *if and only if* $\langle K, +, \times, 0, 1, E, +, ., \mathbf{0}, F, +, ., \mathbf{0}, G, +, ., \mathbf{0}, \otimes\rangle$ *is a model of the system* V'*.*

Proposition 6. *Let* $\mathbf{t}$ *and* $\mathbf{u}$ *be two terms whose variables of sort* E *are among* $\mathbf{x}_1, ..., \mathbf{x}_n$*, whose variables of sort* F *are among* $\mathbf{y}_1, ..., \mathbf{y}_p$*, and that have no variables of sort* G *and* K*. The following propositions are equivalent:*

(i) *the normal forms of* $\mathbf{t}$ *and* $\mathbf{u}$ *are identical modulo AC,*

(ii) *the equation* $\mathbf{t} = \mathbf{u}$ *is valid in all structures formed by three vector spaces and a bilinear function,*

(iii) *the equation* $\mathbf{t} = \mathbf{u}$ *is valid in all structures formed by two vector spaces and their tensor product,*

(iv) *and the denotation of* $\mathbf{t}$ *and* $\mathbf{u}$ *in* K^{np} *for the assignment*

$$\phi = \mathbf{e}_1/\mathbf{x}_1, ..., \mathbf{e}_n/\mathbf{x_n}, \mathbf{e}'_1/\mathbf{y}_1, ..., \mathbf{e}'_p/\mathbf{y_p}$$

where $\mathbf{e}_1, ..., \mathbf{e}_n$ *is the canonical base of* K^n*,* $\mathbf{e}'_1, ..., \mathbf{e}'_p$ *that of* K^p *and* $\otimes$ *is the unique bilinear function such that* $\mathbf{e}_i \otimes \mathbf{e}'_j = \mathbf{e}''_{p(i-1)+j}$ *where* $\mathbf{e}''_1, ..., \mathbf{e}''_{np}$ *is the canonical base of* K^{np}*.*

3. The field of quantum computing

As explained in Section 2.2, fields are not easily implemented as term rewrite systems. In the previous section such problems were avoided by simply assuming the provision of *some* scalar rewrite system, i.e., some term rewrite system for scalars having a certain number of properties (Definition 7). However if the objective is to provide a formal operational semantics for a quantum programming language, up to the point that it provides a full description of a classical simulator for the language, then we must give such a term rewrite system explicitly. The present section briefly outlines how this can be achieved.

3.1. A rewrite system for the field $\mathbb{Q}(i,\sqrt{2})$

In the circuit model of quantum computation the emphasis was placed on the ability to *approximate* any unitary transform from a finite set of gates, where approximation is defined in terms of the distance induced by the supremum norm. This line of research (cf. [53, 37] to cite a few) has culminated with [17], where the following set

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1}$$
$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \quad P = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

was proven to be universal, in the sense that its closure under composition, tensor product and tracing out forms a dense set relative to the set of unitary matrices — with respect to the supremum norm. Thus, the field $\mathbb{Q}(i,\sqrt{2})$ is enough for quantum computation.

To define a scalar rewrite system for this field we can proceed in three steps.

- We define a scalar rewrite system for $\mathbb{Q}$,
- We use Section 2.2 to define a rewrite system for the vector space of linear combination of the form:
$$\alpha.\mathbf{1} + \beta.\frac{\mathbf{1}}{\sqrt{\mathbf{2}}} + \gamma.\mathbf{i} + \delta.\frac{\mathbf{i}}{\sqrt{\mathbf{2}}}.$$
- We use Section 2.6 to define multiplication as a bilinear AC-operation and with the rules
$$\begin{array}{rr} \mathbf{1} \times \mathbf{v} \longrightarrow \mathbf{v} & \frac{\mathbf{1}}{\sqrt{\mathbf{2}}} \times \frac{\mathbf{1}}{\sqrt{\mathbf{2}}} \longrightarrow (1/2).\mathbf{1} \\ \frac{\mathbf{1}}{\sqrt{\mathbf{2}}} \times \mathbf{i} \longrightarrow \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} & \frac{\mathbf{1}}{\sqrt{\mathbf{2}}} \times \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} \longrightarrow (1/2).\mathbf{i} \\ \mathbf{i} \times \mathbf{i} \longrightarrow (-1/2).\mathbf{1} & \mathbf{i} \times \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} \longrightarrow (-1/2).\frac{\mathbf{1}}{\sqrt{\mathbf{2}}} \\ \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} \times \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} \longrightarrow (-1/2).\mathbf{1} & \end{array}$$

3.2. Restricting to the ring $\mathbb{D}(i,1/\sqrt{2})$

For the sake of implementation we can make further simplifications. As division does not appear when composing unitary operators, the only scalars that appear in the closure of gates (1) are those obtained by additive and multiplicative closure from the elements 1, i and $\frac{1}{\sqrt{2}}$, i.e., elements of the ring $\mathbb{D}(i,1/\sqrt{2})$

where $\mathbb{D}$ is the ring of numbers that have a finite dyadic development. Thus this ring is enough for quantum computing.

Notice that, as noticed above, switching to the theory of ring would not be sufficient as, in order to express the gates of quantum computing, we need terms expressing the scalars i and $1/\sqrt{2}$. An implementation of this ring, along these lines, can be found in [8] that builds upon Section 3.1 and [20, 58] for implementing binary numbers.

4. Towards a higher-order language

We introduce a language combining higher-order computation and linear algebra. The syntax of this language is minimal, in the sense that it just contains the syntax of λ-calculus and the possibility to make linear combinations of terms $\alpha.\mathbf{t} + \beta.\mathbf{u}$. This language is called Linear-algebraic λ-calculus, or just *Lineal*.

To start with, we take as the operational semantics, the rewrite rules of vector spaces of Section 2.2, the bilinearity of application of Section 2.6, and the β-reduction rule. The goal of this section is to present the issues arising. This will result in the fine-tuned system presented in Definition 13.

4.1. Higher-order λ-calculus

In quantum computing, many algorithms fall into the category of *black-box* algorithms. I.e., some mysterious implementation of a function f is provided to us which we call "oracle" – and we wish to evaluate some property of f, after a limited number of queries to its oracle. For instance in the Deutsch-Josza quantum algorithm, f is a function $f : \{\mathbf{false}, \mathbf{true}\}^n \longrightarrow \{\mathbf{false}, \mathbf{true}\}$ which is either constant (i.e., $\exists c \forall x[f(x) = c]$) or balanced (i.e., $|\{x \,\text{such that}\, f(x) = \mathbf{false}\}| = |\{x \,\text{such that}\, f(x) = \mathbf{true}\}|$), and whose corresponding oracle is a unitary transformation $U_f : \mathcal{H}_2^{n+1} \longrightarrow \mathcal{H}_2^{n+1}$ such that $U_f : \mathbf{x} \otimes \mathbf{b} \mapsto \mathbf{x} \otimes (\mathbf{b} \oplus f(\mathbf{x}))$, where $\mathcal{H}_2^{n+1}$ stands for a tensor product of $n+1$ two-dimensional vector spaces, $\otimes$ is the tensor product and $\oplus$ just the addition modulo two. The aim is to determine whether f is constant or balanced, and it turns out that this can be done in one single query to its oracle. The algorithm works by applying $H^{\otimes^{n+1}}$ upon $(\mathbf{false}^{\otimes^n} \otimes \mathbf{true})$, then U_f, and then $H^{\otimes^{n+1}}$ again, where $H^{\otimes^{n+1}}$ means applying the Hadamard gate on each of the $n+1$ qubits. It is clear that a desirable feature for a linear-algebraic functional language is to be able to express algorithms as a function of an oracle. E.g. we may want to define

$$\mathbf{Dj}_1 \equiv \lambda \mathbf{x}\,((H \otimes H)\,(\mathbf{x}\,((H \otimes H)\,(\mathbf{false} \otimes \mathbf{true})))$$

so that $\mathbf{Dj}_1\, U_f$ reduces to $(H \otimes H)\,(U_f\,((H \otimes H)\,(\mathbf{false} \otimes \mathbf{true})))$. More importantly even, one must be able to express algorithms, whether they are black-box or not, independent of the size of their input. This is what differentiates programs from fixed-size circuits acting upon finite dimensional vector spaces, and demonstrates the ability to have control flow. The way to achieve this in functional languages involves duplicating basic components of the algorithm an appropriate number of times. E.g. we may want to define some $\mathbf{Dj}$ operator so that $(\mathbf{Dj}\ \mathbf{n})\ U_f$ reduces to the appropriate $(\mathbf{Dj}_n)\ U_f$, where $\mathbf{n}$ is a natural number.

Clearly the language of uniform circuits does not offer an elegant presentation for this issue. Higher-order appears to be a desirable feature to have for black-box computing, but also for expressing recursion and for high-level programming.

4.2. Reduction strategies: copying and cloning

We seek to design a λ-calculus, i.e., have the possibility to introduce and abstract upon variables, as a mean to express functions of these variables. In doing so, we must allow functions such as $\lambda\mathbf{x}\,(\mathbf{x}\otimes\mathbf{x})$, which duplicate their argument. This is necessary for expressiveness, for instance in order to obtain the fixed point operator or any other form of iteration/recursion.

Now problems come up when functions such as $\lambda\mathbf{x}\,(\mathbf{x}\otimes\mathbf{x})$ are applied to superpositions (i.e., sums of vectors). Linear-algebra brings a strong constraint: we know that cloning is not allowed, i.e., that the operator which maps any vector ψ onto the vector $\psi\otimes\psi$ is not linear. In quantum computation this impossibility is referred to as the *no-cloning* theorem [59]. Most quantum programming language proposals so far consist in some quantum registers undergoing unitary transforms and measures on the one hand, together with classical registers and programming structures ensuring control flow on the other, precisely in order to avoid such problems. But as we seek to reach beyond this duality and obtain a purely quantum programming language, we need to face it in a different manner.

This problem may be seen as a confluence problem. Faced with the term $(\lambda\mathbf{x}\,(\mathbf{x}\otimes\mathbf{x}))\,(\mathbf{false}+\mathbf{true})$, one could either start by substituting $\mathbf{false}+\mathbf{true}$ for $\mathbf{x}$ and get the normal form $(\mathbf{false}+\mathbf{true})\otimes(\mathbf{false}+\mathbf{true})$, or start by using the fact that all the functions defined in our language must be linear and get $((\lambda\mathbf{x}\,(\mathbf{x}\otimes\mathbf{x}))\,\mathbf{false})+((\lambda\mathbf{x}\,(\mathbf{x}\otimes\mathbf{x}))\,\mathbf{true})$ and finally the normal form $(\mathbf{false}\otimes\mathbf{false})+(\mathbf{true}\otimes\mathbf{true})$, leading to two different results. More generally, faced with a term of the form $(\lambda\mathbf{x}\,\mathbf{t})\,(\mathbf{u}+\mathbf{v})$, one could either start by substituting $\mathbf{u}+\mathbf{v}$ for $\mathbf{x}$, or start by applying the right-hand-side linearity of the application, breaking the confluence of the calculus. So that operations remain linear, it is clear that we must start by developing over the $+$ first, until we reach a base vector and then apply β-reduction. By base vector we mean a term which does not reduce to a superposition. Therefore, we define a reduction strategy where the β-reduction rule is restricted to cases where the argument is a base vector, as formalized later.

With this restriction, we say that our language allows *copying* but not *cloning* [6, 4]. It is clear that copying has all the expressiveness required in order to express control flow, since it behaves exactly like the standard β-reduction as long as the argument passed is not in a superposition. This is the appropriate linear extension of the β-reduction, philosophically it comprehends classical computation as a (non-superposed) sub-case of linear-algebraic/quantum computation.

The same applies to erasing: the term $\lambda\mathbf{x}\lambda\mathbf{y}\ \mathbf{x}$ expresses the linear operator mapping the base vector $\mathbf{b}_i\otimes\mathbf{b}_j$ to $\mathbf{b}_i$. Again this is in contrast with other programming languages where erasing is treated in a particular fashion whether for the purpose of linearity of bound variables or the introduction of quantum measurement.

4.3. Base (in)dependence

The main conceptual difficulty when seeking to let our calculus be higher-order is to understand how it combines with this idea of *copying*, i.e., duplicating only base vectors. Terms of the form $(\lambda\mathbf{x}\,(\mathbf{x}\,\mathbf{x}))\,(\lambda\mathbf{x}\,\mathbf{v})$ raise the important question of whether the λ-term $\lambda\mathbf{x}\,\mathbf{v}$ must be considered to be a base vector or not:

- we need to restrict $(\lambda\mathbf{x}\,\mathbf{t})\,\mathbf{u} \longrightarrow \mathbf{t}[\mathbf{u}/\mathbf{x}]$ to "base vectors";
- we want higher-order in the traditional sense $(\lambda\mathbf{x}\,\mathbf{t})\,(\lambda\mathbf{y}\,\mathbf{u}) \longrightarrow \mathbf{t}[\lambda\mathbf{y}\,\mathbf{u}/\mathbf{x}]$;
- therefore abstractions must be the base vectors;
- since variables will only ever be substituted by base vectors, they also are base vectors.

It is clear that there is a notion of privileged basis arising in the calculus, but without us having to a priori choose a canonical basis (e.g. we do not introduce some arbitrary orthonormal basis $\{|i\rangle\}$ all of a sudden – i.e., we have nowhere specified a basis at the first-order level). The eventual algebraic consequences of this notion of a privileged basis arising only because of the higher-order level are left as a topic for further investigations. An important intuition is that $(\lambda\mathbf{x}\,\mathbf{v})$ is not the vector itself, but its classical description, i.e., the machine constructing it – hence it is acceptable to be able to copy $(\lambda\mathbf{x}\,\mathbf{v})$ whereas we cannot clone $\mathbf{v}$. The calculus does exactly this distinction.

4.4. Infinities & confluence

It is possible, in our calculus, to define fixed point operators. For instance for each term $\mathbf{b}$ we can define the term

$$\mathbf{Y_b} = \big((\lambda\mathbf{x}\,(\mathbf{b} + (\mathbf{x}\,\mathbf{x})))\big(\lambda\mathbf{x}\,(\mathbf{b} + (\mathbf{x}\,\mathbf{x}))\big)\big)$$

Then the term $\mathbf{Y_b}$ reduces to $\mathbf{b} + \mathbf{Y_b}$, i.e., the term reductions generate a computable series of vectors $(n.\mathbf{b} + \mathbf{Y_b})_n$ whose "norm" grows towards infinity. This was expected in the presence of both fixed points and linear algebra, but the appearance of such infinities entails the appearance of indefinite forms, which we must handle with great caution. Marrying the full power of untyped λ-calculus, including fixed point operators etc., with linear-algebra therefore jeopardizes the confluence of the calculus, unless we introduce some further restrictions.

Example 2. *If we took an unrestricted factorization rule* $\alpha.\mathbf{t}+\beta.\mathbf{t} \longrightarrow (\alpha+\beta).\mathbf{t}$*, then the term* $\mathbf{Y_b} - \mathbf{Y_b}$ *would reduce to* $(1 + (-1)).\mathbf{Y_b}$ *and then* $\mathbf{0}$*. It would also be reduce to* $\mathbf{b} + \mathbf{Y_b} - \mathbf{Y_b}$ *and then to* $\mathbf{b}$*, breaking the confluence.*

Thus, exactly like in elementary calculus $\infty - \infty$ cannot be simplified to 0, we need to introduce a restriction to the rule allowing to factor $\alpha.\mathbf{t} + \beta.\mathbf{t}$ into $(\alpha+\beta).\mathbf{t}$ to the cases where $\mathbf{t}$ is finite. But what do we mean by finite? Notions of norm in the usual mathematical sense seem difficult to import here. In order to avoid infinities we would like to ask that $\mathbf{t}$ is normalizable, but this is impossible to test in general. Hence, we restrict further the rule $\alpha.\mathbf{t} + \beta.\mathbf{t} \longrightarrow (\alpha + \beta).\mathbf{t}$ to the case where the term $\mathbf{t}$ is normal. It is quite striking to see how this restriction equates the algebraic notion of "being normalized" with the rewriting notion of "being normal". The next three examples show that this indefinite form may pop up in some other, more hidden, ways.

Example 3. *Consider the term* $(\lambda\mathbf{x}\,((\mathbf{x}\,\_) - (\mathbf{x}\,\_)))\,(\lambda\mathbf{y}\,\mathbf{Y_b})$ *where* $\_$ *is any base vector, for instance* **false**. *If the term* $(\mathbf{x}\,\_) - (\mathbf{x}\,\_)$ *reduced to* $\mathbf{0}$ *then this term would both reduce to* $\mathbf{0}$ *and also to* $\mathbf{Y_b} - \mathbf{Y_b}$*, breaking confluence.*

Thus, the term $\mathbf{t}$ we wish to factor must also be closed, so that it does not contain any hidden infinity.

Example 4. *If we took an unrestricted rule* $(\mathbf{t}+\mathbf{u})\,\mathbf{v} \longrightarrow (\mathbf{t}\,\mathbf{v})+(\mathbf{u}\,\mathbf{v})$ *the term* $(\lambda\mathbf{x}\,(\mathbf{x}\,\_) - \lambda\mathbf{x}\,(\mathbf{x}\,\_))\,(\lambda\mathbf{y}\,\mathbf{Y_b})$ *would reduce to* $\mathbf{Y_b} - \mathbf{Y_b}$ *and also to* $\mathbf{0}$*, breaking confluence.*

Thus we have to restrict the rule $(\mathbf{t} + \mathbf{u})\,\mathbf{v} \longrightarrow (\mathbf{t}\,\mathbf{v}) + (\mathbf{u}\,\mathbf{v})$ to the case where $\mathbf{t} + \mathbf{u}$ is normal and closed.

Example 5. *If we took an unrestricted rule* $(\alpha.\mathbf{u})\,\mathbf{v} \longrightarrow \alpha.(\mathbf{u}\,\mathbf{v})$ *then the term* $(\alpha.(\mathbf{x} + \mathbf{y}))\,\mathbf{Y_b}$ *would reduce both to* $(\alpha.\mathbf{x} + \alpha.\mathbf{y})\,\mathbf{Y_b}$ *and to* $\alpha.((\mathbf{x} + \mathbf{y})\,\mathbf{Y_b})$*, breaking confluence due to the previous restriction.*

Thus we have to restrict the rule $(\alpha.\mathbf{u})\,\mathbf{v} \longrightarrow \alpha.(\mathbf{u}\,\mathbf{v})$ to the case where $\mathbf{u}$ is normal and closed.

This discussion motivates each of the restrictions $(*) - (****)$ in the rules below. These restrictions are not just a fix: they are a way to formalize vector spaces in the presence of limits/infinities. It may come as a surprise, moreover, that we are able to tame these infinities with this small added set of restrictions, and without any need for context-sensitive conditions, as we shall prove in Section 7.

5. A higher-order language

We consider a first-order language, called *the language of scalars*, containing at least constants 0 and 1 and binary function symbols + and $\times$. The *language of vectors* is a two-sorted language, with a sort for vectors and a sort for scalars, described by the following term grammar:

$$\mathbf{t} ::= \quad \mathbf{x} \quad | \quad \lambda\mathbf{x}\,\mathbf{t} \quad | \quad \mathbf{t}\ \mathbf{t} \quad | \quad \mathbf{0} \quad | \quad \alpha.\mathbf{t} \quad | \quad \mathbf{t} + \mathbf{t}$$

where α is a term in the language of scalars.

In this paper we consider only semi-open terms, i.e., terms containing vector variables but no scalar variables. In particular all scalar terms will be closed.

As usual we write $\mathbf{t}\,\mathbf{u}_1 \dots \mathbf{u}_n$ for $\dots(\mathbf{t}\,\mathbf{u}_1)\dots \mathbf{u}_n$.

Definition 13 (The system L). *Our small-step semantics is defined by the relation $\longrightarrow_L$ where L is the AC-rewrite system where the only AC-symbol is $+$ and the rules are those of S, a scalar rewrite system (see Definition 7) and the union of four groups of rules E, F, A and B:*
- Group E – elementary rules

$$\begin{aligned}
\mathbf{u}+\mathbf{0} &\longrightarrow \mathbf{u},\\
0.\mathbf{u} &\longrightarrow \mathbf{0},\\
1.\mathbf{u} &\longrightarrow \mathbf{u},\\
\alpha.\mathbf{0} &\longrightarrow \mathbf{0},\\
\alpha.(\beta.\mathbf{u}) &\longrightarrow (\alpha\times\beta).\mathbf{u},\\
\alpha.(\mathbf{u}+\mathbf{v}) &\longrightarrow \alpha.\mathbf{u}+\alpha.\mathbf{v}
\end{aligned}$$

- Group F – factorisation

$$\begin{aligned}
\alpha.\mathbf{u}+\beta.\mathbf{u} &\longrightarrow (\alpha+\beta).\mathbf{u}, && (*)\\
\alpha.\mathbf{u}+\mathbf{u} &\longrightarrow (\alpha+1).\mathbf{u}, && (*)\\
\mathbf{u}+\mathbf{u} &\longrightarrow (1+1).\mathbf{u}, && (*)
\end{aligned}$$

- Group A – application

$$\begin{aligned}
(\mathbf{u}+\mathbf{v})\ \mathbf{w} &\longrightarrow (\mathbf{u}\ \mathbf{w})+(\mathbf{v}\ \mathbf{w}), && (**)\\
\mathbf{w}\ (\mathbf{u}+\mathbf{v}) &\longrightarrow (\mathbf{w}\ \mathbf{u})+(\mathbf{w}\ \mathbf{v}), && (**)\\
(\alpha.\mathbf{u})\ \mathbf{v} &\longrightarrow \alpha.(\mathbf{u}\ \mathbf{v}), && (***)\\
\mathbf{v}\ (\alpha.\mathbf{u}) &\longrightarrow \alpha.(\mathbf{v}\ \mathbf{u}), && (***)\\
\mathbf{0}\ \mathbf{u} &\longrightarrow \mathbf{0},\\
\mathbf{u}\ \mathbf{0} &\longrightarrow \mathbf{0},
\end{aligned}$$

- Group B – beta reduction

$$(\lambda\mathbf{x}\ \mathbf{t})\ \mathbf{b} \longrightarrow \mathbf{t}[\mathbf{b}/\mathbf{x}] \qquad (****)$$

And:
$(*)$ the three rules apply only if $\mathbf{u}$ is a closed L-normal term.
$(**)$ the two rules apply only if $\mathbf{u}+\mathbf{v}$ is a closed L-normal term.
$(***)$ the two rules apply only if $\mathbf{u}$ is a closed L-normal term.
$(****)$ the rule applies only when $\mathbf{b}$ is a "base vector" term, i.e., an abstraction or a variable.

Notice that the restriction $(*)$, $(**)$ and $(***)$ are well-defined as the terms to which the restrictions apply are smaller than the left-hand side of the rule.
Notice also that the restrictions are stable by substitution. Hence these conditional rules could be replaced by an infinite number of non conditional rules, i.e., by replacing the restricted variables by all the closed normal terms verifying the conditions.
Finally notice how the rewrite system $R = S \cup E \cup F \cup A$, taken without restrictions, is really just the V' (see Definition 12) we have seen in Section 2, i.e., an oriented version of the axioms of vector spaces. Intuitively the restricted system defines a notion of vector space with infinities. Rewrite rules with closedness conditions are not new in the theory of λ-calculus (see, for instance, [29]).

Normal forms. We have explained why abstractions ought to be considered as "base vectors" in our calculus. We have highlighted the presence of non-terminating terms and infinities, which make it impossible to interpret the calculus in your usual vector space structure. The following two results show that terminating closed terms on the other hand can really be viewed as superpositions of abstractions.

Proposition 7. *An L-closed normal form, that is not a sum, a product by a scalar, or the null vector, is an abstraction.*

Proof. By induction over term structure. Let $\mathbf{t}$ be a closed normal term that is not a sum, a product by a scalar, or the null vector. The term $\mathbf{t}$ is not a variable because it is closed, hence it is either an abstraction in which case we are done, or an application. In this case it has the form $\mathbf{u}\,\mathbf{v}_1 \ldots \mathbf{v}_n$ where $\mathbf{u}, \mathbf{v}_1, \ldots \mathbf{v}_n$ are normal and closed and n is different from 0. Neither $\mathbf{u}$ nor $\mathbf{v}_1$ is a sum, a product by a scalar, or the null vector since the term being normal we then could apply rules of group A. Thus by induction hypothesis, both terms are abstractions, thus the rule B applies and the term $\mathbf{t}$ is not normal.

Proposition 8 (Form of closed normal forms). *A L-closed normal form is either the null vector or of the form $\sum_i \alpha_i.\lambda\mathbf{x}\,\mathbf{t}_i + \sum_i \lambda\mathbf{x}\,\mathbf{u}_i$ where the scalars are different from 0 and 1 and the abstractions are all distinct.*

Proof. If the term is not the null vector it can be written as a sum of terms that are neither $\mathbf{0}$ nor sums. We partition these terms in order to group those which are weighted by a scalar and those which are not. Hence we obtain a term of the form

$$\sum \alpha'_i.\mathbf{t'}_i + \sum \mathbf{u'}_i$$

where the terms $\mathbf{u'}_i$ are neither null, nor sums, nor weighted by a scalar. Hence by Proposition 7 they are abstractions. Because the whole term is normal the terms $\mathbf{t'}_i$ are themselves neither null, nor sums, nor weighted by a scalar since we could apply rules of group E. Hence Proposition 7 also applies.

6. Encoding classical and quantum computation

The restrictions we have placed upon our language are still more permissive than those of the call-by-value λ-calculus, hence any classical computation can be expressed in the linear-algebraic λ-calculus just as it can in the call-by-value λ-calculus.

It then suffices to express the three universal quantum gates $\mathbf{H}, \mathbf{Phase}, \mathbf{Cnot}$ [17] which we will do next.

Encoding booleans. We encode the booleans as the first and second projections, as usual in the classical λ-calculus: $\mathbf{true} \equiv \lambda\mathbf{x}\,\lambda\mathbf{y}\,\mathbf{x}$, $\mathbf{false} \equiv \lambda\mathbf{x}\,\lambda\mathbf{y}\,\mathbf{y}$. Again, note that these are conceived as linear functions, the fact we erase the second/first argument does not mean that the term should be interpreted as a trace out or a measurement. Here is a standard example on how to use them:

$$\mathbf{Not} \equiv \lambda\mathbf{y}\,\Big(\mathbf{y}\,\mathbf{false}\,\mathbf{true}\Big).$$

Notice that this term expresses a unitary operator upon the vector space generated by **true** and **false**, even if some subterms express non unitary ones.

Encoding unary quantum gates. For the Phase gate the naive encoding will not work, i.e.,

$$\mathbf{Phase} \not\equiv \lambda\mathbf{y}\,\Big(\mathbf{y}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\mathbf{false}\Big)$$

since by bilinearity this would give $\mathbf{Phase\ false} \longrightarrow_L^* e^{i\frac{\pi}{4}}.\mathbf{false}$, whereas the Phase gate is supposed to place an $e^{i\frac{\pi}{4}}$ only on **true**. The trick is to use abstraction in order to retain the $e^{i\frac{\pi}{4}}$ phase on **true** only (where $\_$ is any base vector, for instance **false**).

$$\mathbf{Phase} \equiv \lambda\mathbf{y}\,\Big(\Big(\mathbf{y}\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_\Big)$$

Now, the term **Phase true** reduces in the following way

$$\lambda\mathbf{y}\,\Big(\Big(\mathbf{y}\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_\Big)\,\mathbf{true} \longrightarrow_L \Big(\mathbf{true}\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_$$
$$= \Big((\lambda\mathbf{x}\,\lambda\mathbf{y}\,\mathbf{x})\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_ \longrightarrow_L^* (\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true}))\,\_ \longrightarrow_L e^{i\frac{\pi}{4}}.\mathbf{true}$$

whereas the term **Phase false** reduces as follows

$$\lambda\mathbf{y}\,\Big(\Big(\mathbf{y}\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_\Big)\,\mathbf{false} \longrightarrow_L \Big(\mathbf{false}\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_$$
$$= \Big((\lambda\mathbf{x}\,\lambda\mathbf{y}\,\mathbf{y})\,\lambda\mathbf{x}\,(e^{i\frac{\pi}{4}}.\mathbf{true})\,\lambda\mathbf{x}\,\mathbf{false}\Big)\,\_ \longrightarrow_L^* (\lambda\mathbf{x}\,\mathbf{false})\,\_ \longrightarrow_L \mathbf{false}$$

This idea of using a dummy abstraction to restrict linearity can be generalized and made more elegant with the following syntactic sugar:

- $[\mathbf{t}] \equiv \lambda \mathbf{x}\, \mathbf{t}$. The effect of this *canon* [.] is to associate to any state $\mathbf{t}$ a base vector $[\mathbf{t}]$.
- $\{\mathbf{t}\} \equiv \mathbf{t}\ \_$ where $\_$ is any closed normal base vector, for instance $\lambda \mathbf{x}\, \mathbf{x}$. The effect of this *uncanon* is to lift the canon, i.e., we have the derived rule $\{[\mathbf{t}]\} \longrightarrow_L \mathbf{t}$.

Note that {.} is a "left-inverse" of [.], but not a "right inverse", just like eval and ′ (quote) in LISP. Again these hooks do not add anymore power to the calculus, in particular they do not enable cloning. We cannot clone a given state $\alpha.\mathbf{t} + \beta.\mathbf{u}$, but we can copy its classical description $[\alpha.\mathbf{t} + \beta.\mathbf{u}]$. For instance the function $\lambda \mathbf{x}\, [\mathbf{x}]$ will never "canonize" anything else than a base vector, because of restriction $(* * **)$. The phase gate can then be written

$$\mathbf{Phase} \equiv \lambda \mathbf{y} \left\{ (\mathbf{y}\, [e^{i\frac{\pi}{4}}.\mathbf{true}])\, [\mathbf{false}] \right\}$$

For the Hadamard gate the game is just the same:

$$\mathbf{H} \equiv \lambda \mathbf{y} \left\{ \mathbf{y}\, [\frac{\sqrt{2}}{2}.(\mathbf{false} + \mathbf{true})]\, [\frac{\sqrt{2}}{2}.(\mathbf{false} - \mathbf{true})] \right\}$$

Encoding tensors. In quantum mechanics, vectors are put together via the bilinear symbol $\otimes$. But because in our calculus application is bilinear, the usual encoding of pairs does just what is needed.

$$\otimes \equiv \lambda \mathbf{x}\, \lambda \mathbf{y}\, \lambda \mathbf{f}\, \big(\mathbf{f}\, \mathbf{x}\, \mathbf{y}\big), \quad \pi_1 \equiv \lambda \mathbf{p}(\mathbf{p}\, \lambda \mathbf{x}\, \lambda \mathbf{y}\, \mathbf{x}), \quad \pi_2 \equiv \lambda \mathbf{p}(\mathbf{p}\, \lambda \mathbf{x}\, \lambda \mathbf{y}\, \mathbf{y}),$$
$$\bigotimes \equiv \lambda \mathbf{f}\, \lambda \mathbf{g}\, \lambda \mathbf{x} \left(\otimes\ \big(\mathbf{f}\, (\pi_1\, \mathbf{x})\big)\, \big(\mathbf{g}\, (\pi_2\, \mathbf{x})\big) \right)$$

E.g. $\mathbf{H}^{\otimes \mathbf{2}} \equiv \big(\bigotimes \mathbf{H}\, \mathbf{H}\big)$. From there on the infix notation for tensors will be used, i.e., $\mathbf{t} \otimes \mathbf{u} \equiv \otimes\, \mathbf{t}\, \mathbf{u}$, $\quad \mathbf{t} \bigotimes \mathbf{u} \equiv \bigotimes\ \mathbf{t}\, \mathbf{u}$.

Encoding the Cnot gate. This binary gate is essentially a classical gate, its encoding is standard.

$$\mathbf{Cnot} \equiv \lambda \mathbf{x} \left((\pi_1\, \mathbf{x}) \otimes \left(\Big((\pi_1\, \mathbf{x})\, \big(\mathbf{Not}\, (\pi_2\, \mathbf{x})\big) \Big)\, (\pi_2\, \mathbf{x}) \right) \right)$$

Expressing the Deutsch-Josza algorithm parametrically.

As discusses in Section 4, an advantage of a higher-order language is that it permits to express black-box algorithms, such as the Deutsch-Josza algorithm, in a parametric way. We show now how to encode this algorithm in Lineal.

Here is the well-known simple example of the Deutsch algorithm, which is the $n = 1$ case of the Deutsch-Josza algorithm

$$\mathbf{Dj}_1 \equiv \lambda \mathbf{x} \left(\mathbf{H}^{\otimes \mathbf{2}} \left(\mathbf{x} \left(\mathbf{H}^{\otimes \mathbf{2}}\, \big(\mathbf{false} \otimes \mathbf{true}\big) \right) \right) \right)$$

But we can also express control structure and use them to express the dependence of the Deutsch-Josza algorithm with respect to the size of the input. Encoding the natural number n as the Church numeral $\mathbf{n} \equiv \lambda\mathbf{x}\,\lambda\mathbf{f}\,(\mathbf{f}^n\ \mathbf{x})$ the term $(\mathbf{n}\ \mathbf{H}\ \lambda\mathbf{y}\,(\mathbf{H} \bigotimes \mathbf{y}))$ reduces to $\mathbf{H}^{\otimes^{n+1}}$ and similarly the term $(\mathbf{n}\ \mathbf{true}\ \lambda\mathbf{y}\,(\mathbf{false} \otimes \mathbf{y}))$ reduces to $\mathbf{false}^{\otimes^n} \otimes \mathbf{true}$. Thus the expression of the Deutsch-Josza algorithm term of the introduction is now straightforward:

$$\mathbf{Dj} \equiv \lambda\mathbf{n}\lambda\mathbf{x}\left((\mathbf{n}\ \mathbf{H}\ \lambda\mathbf{y}\,(\mathbf{H} \bigotimes \mathbf{y}))\left(\mathbf{x}\left((\mathbf{n}\ \mathbf{H}\ \lambda\mathbf{y}\,(\mathbf{H} \bigotimes \mathbf{y}))\,(\mathbf{n}\ \mathbf{true}\ \lambda\mathbf{y}\,(\mathbf{false} \otimes \mathbf{y}))\right)\right)\right).$$

Infinite dimensional operators. Notice that our language enables us to express operators independently of the dimension of the space they apply to, even when this dimension is infinite. For instance the identity operator is not expressed as a sum of projections whose number would depend on the dimension of the space, but as the mere lambda term $\lambda\mathbf{x}\,\mathbf{x}$. In this sense our language is a language of infinite dimensional computable linear operators, in the same way that matrices are a language of computable finite dimensional linear operators.

7. Confluence

The main theorem of this paper is the confluence of the system L. Along the way, we will also prove the confluence of the unrestricted systems V and V' which we introduced for the sake of our computational definition of vector spaces. This section is quite technical. A reader who is not familiar with rewriting techniques may be happy with reading just Definition 16 and Theorem 1, and then skipping to Section 8. A reader with an interest in such techniques may on the other hand find the architecture of the proof quite useful. The rationale is as follows:

- There is only a very limited set of techniques available for proving the confluence of non-terminating rewrite systems (mainly the notions of parallel reductions and strong confluence). Hence we must distinguish the non-terminating rules, which generate the infinities (the B rule), from the others (the R rules) and show that they are confluent on their own;

- We then must show that the terminating rules and the non-terminating rules commute, so that their union is confluent. (The conditions on B, F, A are key to obtaining the commutation. Without them, both subsets are confluent, but not their union.)

- The rules of R being terminating, the critical pairs lemma applies. The critical pairs can be checked automatically for the non-conditional rules, but the conditional ones must be checked by hand. Hence we must distinguish the non-conditional rules (E rules), from the others (the F and A rules).

- In order to handle the parametricity with respect to the scalars rewrite system, we shall introduce a new technique called the avatar's lemma.

The first step is to prove the confluence of the system $R = S \cup E \cup F \cup A$, i.e., the system L of Definition 13, minus the rule B. To prove the confluence of this system we shall prove that of the system $R_0 = S_0 \cup E \cup F \cup A$ where S_0 is a small avatar of S, namely the simplest possible scalar rewrite system. Then we use the avatar lemma to extend the result from S_0 to S, hereby obtaining the confluence of $R = S \cup E \cup F \cup A$.

Definition 14 (The rewrite system S_0). *The system S_0 is formed by the rules*

$$0 + \alpha \longrightarrow \alpha$$
$$0 \times \alpha \longrightarrow 0$$
$$1 \times \alpha \longrightarrow \alpha$$
$$\alpha \times (\beta + \gamma) \longrightarrow (\alpha \times \beta) + (\alpha \times \gamma)$$

where $+$ and $\times$ are AC-symbols.

To be able to use a critical pair lemma in an AC context, we shall use a well-known technique, detailed in the Section 7.1, and introduce addenda S_{0ext}, E_{ext} and F_{ext} to the systems S_0, E and F, with some extra rules called *extension* rules, but with a more restricted form of AC-rewriting and the system $R_{0ext} = S_{0ext} \cup E_{ext} \cup F_{ext} \cup A$.

The second step towards our main goal is show that the $B^{\|}$ rule, the parallel version of the rule B defined in Definition 21, is strongly confluent on the term algebra, and commutes with R^*, hence giving the confluence of L (Section 7.5).

As the system $R = S \cup E \cup F \cup A$ does not deal at all with abstractions and bound variables, we have, throughout this first part of the proof, considered $\lambda\mathbf{x}$ as a unary function symbol and the bound occurrences of $\mathbf{x}$ as constants. This way we can safely apply known theorems about first-order rewriting.

7.1. Reminder on rule extensions and the critical pairs lemma

The term $((\mathbf{a} + \mathbf{2}.\mathbf{a}) + \mathbf{b}) + \mathbf{c}$ reduces to the term $((2 + 1).\mathbf{a} + \mathbf{b}) + \mathbf{c}$, as it contains a subterm $\mathbf{a} + \mathbf{2}.\mathbf{a}$ that is AC-equivalent to an instance of the left hand side of the rule $\alpha.\mathbf{u} + \mathbf{u} \longrightarrow (\alpha + 1).\mathbf{u}$. The term $((2.\mathbf{a} + \mathbf{b}) + \mathbf{a}) + \mathbf{c}$ does not contain such a subterm, yet it can be reduced, because it is itself AC-equivalent to the term $((2.\mathbf{a} + \mathbf{a}) + \mathbf{b}) + \mathbf{c}$ that contain a subterm that is an instance of the left hand side of this rule. This suggests that there is a more local way to define reduction modulo AC where the first reduction is possible but not the second.

Unfortunately, the critical pair lemma for AC-rewrite system gives the confluence of this local version of AC-reduction system and not the global one we are interested in. This problem has been solved by [42, 36] that show that the globally AC reduction relation is confluent if the locally AC reduction relation of an extended rewrite system is confluent.

Definition 15 (The extension rules). *Let X be a AC-rewrite system with AC symbols $f_1, \ldots, f_n$. We define the AC-rewrite system X_{ext} as containing the same AC symbols as X, the same rules as X, plus the rules $f_i(t, x) \longrightarrow f_i(u, x)$ for each rule $t \longrightarrow u$ of X where the head symbol of t is f_i.*

Proposition 9 **(R_{0ext}).** *The system S_{0ext} is formed by the rules of S_0 and the rule*

$$(0+\alpha)+\chi \longrightarrow \alpha+\chi$$

$$(0\times\alpha)\times\chi \longrightarrow 0\times\chi$$

$$(1\times\alpha)\times\chi \longrightarrow \alpha\times\chi$$

$$(\alpha\times(\beta+\gamma))\times\chi \longrightarrow ((\alpha\times\beta)+(\alpha\times\gamma))\times\chi$$

The system E_{ext} is formed by the rules of E and the rule:

$$(\mathbf{u}+\mathbf{0})+\mathbf{x} \longrightarrow \mathbf{u}+\mathbf{x}$$

The system F_{ext} is formed by the rules of F and these three rules:

$$(\alpha.\mathbf{u}+\beta.\mathbf{u})+\mathbf{x} \longrightarrow (\alpha+\beta).\mathbf{u}+\mathbf{x} \quad (*)$$

$$(\alpha.\mathbf{u}+\mathbf{u})+\mathbf{x} \longrightarrow (\alpha+1).\mathbf{u}+\mathbf{x} \quad (*)$$

$$(\mathbf{u}+\mathbf{u})+\mathbf{x} \longrightarrow (1+1).\mathbf{u}+\mathbf{x} \quad (*)$$

where $(*)$ imposes the three rules apply only if **u** *is a closed normal term. The system A_{ext} is A.*
The system R_{0ext} is $S_{0ext}\cup E_{ext}\cup F_{ext}\cup A$.

As usual we write $\mathbf{t}\longrightarrow_X^* \mathbf{u}$ if and only if $\mathbf{t}=\mathbf{u}$ or $\mathbf{t}\longrightarrow_X \ldots \longrightarrow_X \mathbf{u}$. We also write $\mathbf{t}\longrightarrow_X^? \mathbf{u}$ if and only if $\mathbf{t}=\mathbf{u}$ or $\mathbf{t}\longrightarrow_X \mathbf{u}$ $\mathbf{t}\longrightarrow_{X;Y} \mathbf{u}$ if there exist a $\mathbf{v}$ such that $\mathbf{t}\longrightarrow_X \mathbf{v}\longrightarrow_Y \mathbf{u}$, and $\mathbf{t}\longrightarrow_{X\downarrow} \mathbf{u}$ if $\mathbf{t}\longrightarrow_X^* \mathbf{u}$ and $\mathbf{u}$ is normal for the relation X.

7.2. Termination

Proposition 10. *The systems S_{0ext} terminates.*

Proof. Consider the following interpretation (compatible with AC)

$$|x|_s = |0|_s = |1|_s = 2$$

$$|t+u|_s = 1+|t|_s+|u|_s$$

$$|t\times u|_s = |t|_s\times|u|_s$$

Each time a term t rewrites to a term t' we have $|t|_s > |t'|_s$. Hence, the system terminates.

Proposition 11. *The systems R_{0ext}, R_{ext}, $S\cup V$ and $S\cup V'$ terminate.*

Proof. [*The system $E_{ext} \cup F_{ext} \cup A$ terminates*]
Consider the following interpretation (compatible with AC)

$$|(\mathbf{u}\ \mathbf{v})| = (3|\mathbf{u}| + 2)(3|\mathbf{v}| + 2)$$

$$|\mathbf{u} + \mathbf{v}| = 2 + |\mathbf{u}| + |\mathbf{v}|$$

$$|\alpha.\mathbf{u}| = 1 + 2|\mathbf{u}|$$

$$|\mathbf{0}| = 0$$

Each time a term $\mathbf{t}$ rewrites to a term $\mathbf{t}'$ we have $|\mathbf{t}| > |\mathbf{t}'|$. Hence, the system terminates.

[*The system R_{0ext} terminates*]
The system R_{0ext} is $S_{0ext} \cup E_{ext} \cup F_{ext} \cup A$. It is formed of two subsystems S_{0ext} and $E_{ext} \cup F_{ext} \cup A$. By definition of the function $|\ |$, if a term $\mathbf{t}$ S_{0ext}-reduces to a term $\mathbf{t}'$ then $|\mathbf{t}| = |\mathbf{t}'|$. Consider a R-reduction sequence. At each $E_{ext} \cup F_{ext} \cup A$-reduction step, the measure of the term strictly decreases and at each S_{0ext}-reduction step it remains the same. Thus there are only a finite number of $E_{ext} \cup F_{ext} \cup A$-reduction steps in the sequence and, as S_{0ext} terminates, the sequence is finite.

The same argument applies for R_{ext}, $S \cup V$ and $S \cup V'$, with respect to S instead of S_{0ext}.

7.3. Critical pairs

Definition 16 (Confluence and local confluence). *A relation X is said to be confluent if whenever $t \longrightarrow_X^* u$ and $t \longrightarrow_X^* v$, there exists a term w such that $u \longrightarrow_X^* w$ and $v \longrightarrow_X^* w$. A relation X is said to be locally confluent if whenever $t \longrightarrow_X u$ and $t \longrightarrow_X v$, there exists a term w such that $u \longrightarrow_X^* w$ and $v \longrightarrow_X^* w$.*

Definition 17 (Critical pair). *Let $l \longrightarrow r$ and $l' \longrightarrow r'$ be two rewrite rules of an AC-rewrite system X, let p be an occurrence in l such that $l_{|p}$ is not a free variable. Let σ be an AC-unifier for $l_{|p}$ and l', the pair $\big(\sigma r, \sigma(l[p \leftarrow r'])\big)$ is a* critical pair *of the the rewrite system X.*

Proposition 12 (Peterson-Stickel Theorem). *If $\longrightarrow_{X_{ext}}$ terminates and for each critical pair (t, u) of X_{ext} there exists a term w such that $t \longrightarrow_{X_{ext}}^* w$ $u \longrightarrow_{X_{ext}}^* w$. Then the relation $\longrightarrow_X$ is confluent.*

Proof. See [42], Theorems 10.5., 9.3 and 8.9.

Proposition 13. *The system $S_0 \cup E$ is confluent.*
The systems $S_0 \cup V$ and $S_0 \cup V'$ are confluent.

Proof. First notice that $(S_0 \cup E)_{ext} = (S_{0ext} \cup E_{ext})$ and that this system terminates (see Proposition 11). Thus by proposition 12 all we need to do is to check that all the critical pair close. As these rules are not conditional, we have used the system CIME [19] to check this automatically. The same applies to $S_0 \cup V_{ext}$ and $S_0 \cup V'_{ext}$.

Proposition 14. *The system $S_0 \cup E \cup F$ is confluent.*

Proof. First notice that the system $(S_{0ext} \cup E \cup F)_{ext} = S_{0ext} \cup E_{ext} \cup F_{ext}$ and that this system terminates (see Proposition 11). Thus by proposition 12 all we need to do is to check that all the critical pair close. If both rules used are rules of the system $S_{0ext} \cup E_{ext}$, then the critical pair closes by Proposition 13. There are no critical pairs between S_{0ext} and $E_{ext} \cup F_{ext}$. Thus, all we need to check are the critical pairs between one rule of E_{ext} and one of F_{ext} or one rule of F_{ext} and one of F_{ext}.

To find the critical pairs, we used CIME [19]. There are 251 critical pairs, not taking the fact that some rules are conditional and thus may not apply. Indeed, among these critical pairs 81 do not verify the conditions of the rules F_{ext}. For instance, the pair given by CIME

$$\alpha.(\mathbf{0} + \mathbf{u}) + \mathbf{u} \longleftarrow \mathbf{0} + \alpha.(\mathbf{0} + \mathbf{u}) + \mathbf{u} \longrightarrow (1 + \alpha).(\mathbf{0} + \mathbf{u})$$

does not verify the condition because $\mathbf{u} + \mathbf{0}$ is never closed normal: we do not need to close this pair because our conditions forbid that it opens.

We need to check the 170 other pairs by hands. Some close easily, for instance the pair

$$\mathbf{0} + \alpha.\mathbf{u} \longleftarrow 0.\mathbf{u} + \alpha.\mathbf{u} \longrightarrow (0 + \alpha).\mathbf{u}$$

closes on $\alpha.\mathbf{u}$.

Some other require a short analysis of the conditions. For instance, for the pair

$$\alpha.\mathbf{u} + \alpha.\mathbf{v} + \beta.(\mathbf{u} + \mathbf{v}) \longleftarrow \alpha.(\mathbf{u} + \mathbf{v}) + \beta.(\mathbf{u} + \mathbf{v}) \longrightarrow (\alpha + \beta).(\mathbf{u} + \mathbf{v})$$

the fact that we have been able to factor $\alpha.(\mathbf{u}+\mathbf{v})+\beta.(\mathbf{u}+\mathbf{v})$ into $(\alpha+\beta).(\mathbf{u}+\mathbf{v})$ shows that the term $\mathbf{u} + \mathbf{v}$ is closed normal, thus the terms $\mathbf{u}$ and $\mathbf{v}$ are closed normal, which permits to reduce both terms to $(\alpha + \beta).\mathbf{u} + (\alpha + \beta).\mathbf{v}$. All the cases are analyzed in [9].

Proposition 15. *The system $R_0 = S_0 \cup E \cup F \cup A$ is confluent.*

Proof. First notice that $R_{0ext} = S_{0ext} \cup E_{ext} \cup F_{ext} \cup A$ and that this system terminates (see Proposition 11). Thus by proposition 12 all we need to do is to check that all the critical pair close. If both rules used are rules of the system $S_{0ext} \cup E_{ext} \cup F_{ext}$, then the critical pair closes by Proposition 14. It is not possible that the top-level rule is in $S_{0ext} \cup E_{ext} \cup F_{ext}$ and the other in A since the rules of $S_{0ext} \cup E_{ext} \cup F_{ext}$ do not contain any application. Thus the top-level rule must be in A and the $(S_{0ext} \cup E_{ext} \cup F_{ext})$-reduction must be performed in a non-toplevel non-variable subterm of the left-hand-side of a rule of A. By inspection of the left-hand-sides of rules $S_{0ext} \cup E_{ext} \cup F_{ext}$ the subterm must be of the form $\mathbf{u} + \mathbf{v}$, $\alpha.\mathbf{u}$ or $\mathbf{0}$. But this subterm cannot be of the form $\mathbf{u} + \mathbf{v}$, because, by restriction (**), the term itself would not be A-reducible. It cannot be $\mathbf{0}$ since this term is normal. Thus it is of the form $\alpha.\mathbf{u}$. As there are five rules reducing a term of this form, there are 10 critical pairs to check. Because of the conditionality of the rewrite system we check them by hand.

Pair 1 $\mathbf{0}\,\mathbf{v} \longleftarrow (0.\mathbf{u})\,\mathbf{v} \longrightarrow 0.(\mathbf{u}\,\mathbf{v})$: this critical pair closes on $\mathbf{0}$.

Pair 2 $\mathbf{u}\,\mathbf{v} \longleftarrow (1.\mathbf{u})\,\mathbf{v} \longrightarrow 1.(\mathbf{u}\,\mathbf{v})$: this critical pair closes on $\mathbf{u}\,\mathbf{v}$.

Pair 3 $\mathbf{0}\,\mathbf{v} \longleftarrow (\alpha.\mathbf{0})\,\mathbf{v} \longrightarrow \alpha.(\mathbf{0}\,\mathbf{v})$: this critical pair closes on $\mathbf{0}$.

Pair 4 $((\alpha \times \beta).\mathbf{u})\,\mathbf{v} \longleftarrow (\alpha.(\beta.\mathbf{u}))\,\mathbf{v} \longrightarrow \alpha.((\beta.\mathbf{u})\,\mathbf{v})$: the term $\mathbf{u}$ is closed and normal by $(***)$. Hence, the critical pair closes on $(\alpha \times \beta).(\mathbf{u}\,\mathbf{v})$.

Pair 5 $(\alpha.\mathbf{u}+\alpha.\mathbf{v})\,\mathbf{w} \longleftarrow (\alpha.(\mathbf{u}+\mathbf{v}))\,\mathbf{w} \longrightarrow \alpha.((\mathbf{u}+\mathbf{v})\,\mathbf{w})$: the term $\mathbf{u}+\mathbf{v}$ is closed and normal. Hence, by Proposition 8 it is of the form $\sum_i \beta_i.\mathbf{a}_i + \sum_i \mathbf{b}_i$. Therefore the top reduct reduces to $(\sum_i(\alpha \times \beta_i) \downarrow .\mathbf{a}_i + \sum_i \alpha.\mathbf{b}_i)\,\mathbf{w}$, where $\downarrow$ denotes normalization by S_{0ext}. We treat only the case where the terms $(l \times \beta_i) \downarrow$ are neither 0 nor 1, the other cases being similar. Hence, we can apply rules of group A yielding $\sum_i(\alpha \times \beta_i) \downarrow .(\mathbf{a}_i\,\mathbf{w}) + \sum_i \alpha.(\mathbf{b}_i\,\mathbf{w})$. It is routine to check that the bottom reduct also reduces to this term.

The five next critical pairs are the symmetrical cases, permuting the left and right-hand-sides of the application.

Now, when both rules are in the group A, there are 9 critical pairs to check.

Pair 11 $\mathbf{u}(\mathbf{w}+\mathbf{x}) + \mathbf{v}(\mathbf{w}+\mathbf{x}) \longleftarrow (\mathbf{u}+\mathbf{v})(\mathbf{w}+\mathbf{x}) \longrightarrow (\mathbf{u}+\mathbf{v})\mathbf{w} + (\mathbf{u}+\mathbf{v})\mathbf{x}$: as $\mathbf{u}+\mathbf{v}$ and $\mathbf{w}+\mathbf{x}$ are normal and closed, so are $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ and $\mathbf{x}$. Hence the critical pair closes on $\mathbf{uw}+\mathbf{ux}+\mathbf{vw}+\mathbf{vx}$.

Pair 12 $\mathbf{u}(\alpha.\mathbf{w}) + \mathbf{v}(\alpha.\mathbf{w}) \longleftarrow (\mathbf{u}+\mathbf{v})(\alpha.\mathbf{w}) \longrightarrow \alpha.((\mathbf{u}+\mathbf{v}).\mathbf{w})$: as, by $(**)$ and $(***)$, $\mathbf{u}+\mathbf{v}$ and $\mathbf{w}$ are closed normal terms , so are $\mathbf{u}$ and $\mathbf{v}$. Thus the top reduct further reduces to $\alpha.(\mathbf{u}\,\mathbf{w})+\alpha.(\mathbf{v}\,\mathbf{w})$ and the bottom reduct further reduces to $\alpha.((\mathbf{u}\,\mathbf{w})+(\mathbf{v}\,\mathbf{w}))$ and both terms reduce to $\alpha.(\mathbf{u}\,\mathbf{w})+\alpha.(\mathbf{v}\,\mathbf{w})$.

Pair 13 $\mathbf{0} \longleftarrow (\mathbf{u}+\mathbf{v})\mathbf{0} \longrightarrow (\mathbf{u0}) + (\mathbf{v0})$: this critical pair closes on $\mathbf{0}$.

Pair 14 $\alpha.(\mathbf{u}(\mathbf{v}+\mathbf{w})) \longleftarrow (\alpha.\mathbf{u})(\mathbf{v}+\mathbf{w}) \longrightarrow (\alpha.\mathbf{u})\mathbf{v}+(\alpha.\mathbf{u})\mathbf{w}$: the terms $\mathbf{u}$ and $\mathbf{v}+\mathbf{w}$ are closed normal. Thus, the top reduct further reduces to $\alpha.(\mathbf{uv}+\mathbf{uw})$ and the bottom reduct to $\alpha.(\mathbf{uv}) + \alpha.(\mathbf{uw})$. Hence the critical pair closes on $\alpha.(\mathbf{uv}) + \alpha.(\mathbf{uw})$.

Pair 15 $\alpha.(\mathbf{u}(\beta.\mathbf{v})) \longleftarrow (\alpha.\mathbf{u})(\beta.\mathbf{v}) \longrightarrow \beta.((\alpha.\mathbf{u})\mathbf{v})$: as $\mathbf{u}$ and $\mathbf{v}$ are closed normal, the first term reduces to $\alpha.(\beta.(\mathbf{uv}))$ and the second to $\beta.(\alpha.(\mathbf{uv}))$ and both terms reduce to $(\alpha \times \beta).(\mathbf{uv})$.

Pair 16 $\alpha.(\mathbf{u0}) \longleftarrow (\alpha.\mathbf{u})\mathbf{0} \longrightarrow \mathbf{0}$: this critical pair closes on $\mathbf{0}$.

Pair 17 $\mathbf{0} \longleftarrow \mathbf{0}(\mathbf{u}+\mathbf{v}) \longrightarrow (\mathbf{0u}) + (\mathbf{0v})$: this critical pair closes on $\mathbf{0}$.

Pair 18 $\mathbf{0} \longleftarrow \mathbf{0}(\alpha.\mathbf{u}) \longrightarrow \alpha.(\mathbf{0u})$: this critical pair closes on $\mathbf{0}$.

Pair 19 $\mathbf{0} \longleftarrow \mathbf{00} \longrightarrow \mathbf{0}$: this critical pair closes on $\mathbf{0}$.

7.4. The avatar lemma

Definition 18 (Subsumption). *A terminating and confluent relation S* sub-sumes *a relation S_0 if whenever $t \longrightarrow_{S_0} u$, t and u have the same S-normal form.*

Definition 19 (Commuting relations). *Two relations X and Y are said to be commuting if whenever $t \longrightarrow_X u$ and $t \longrightarrow_Y v$, there exists a term w such that $u \longrightarrow_Y w$ and $v \longrightarrow_X w$.*

Proposition 16 (The avatar lemma). *[6] Let X, S and S_0 be three relations defined on a set such that:*

- *S is terminating and confluent;*
- *S subsumes S_0;*
- *$S_0 \cup X$ is locally confluent;*
- *X commutes with S^*.*

Then, the relation $S \cup X$ is locally confluent.

Proof. [*X can be simulated by $X; S\downarrow$*].
If $t \longrightarrow_X u$ and $t \longrightarrow_{S\downarrow} v$, then there exists w such that $u \longrightarrow_{S\downarrow} w$ and $v \longrightarrow_{X;S\downarrow} w$. Indeed by commutation of X and S^* there exists a such that $u \longrightarrow_{S^*} a$ and $v \longrightarrow_X a$. Normalizing a under S yields the w.

[*$S_0 \cup X$ can be simulated by $(X; S\downarrow)^?$*].

If $t \longrightarrow_{S_0 \cup X} u$ and $t \longrightarrow_{S\downarrow} v$, then there exists w such that $u \longrightarrow_{S\downarrow} w$ and $v \longrightarrow^?_{X;S\downarrow} w$. Indeed if $t \longrightarrow_{S_0} u$ this is just subsumption, else the first point of this proof applies.

[*$S \cup X$ can be simulated by $(X; S\downarrow)^?$*].

If $t \longrightarrow_{S \cup X} u$ and $t \longrightarrow_{S\downarrow} v$, then there exists w such that $u \longrightarrow_{S\downarrow} w$ and $v \longrightarrow^?_{X;S\downarrow} w$. Indeed if $t \longrightarrow_S u$ this is just the normalization of S, else the first point of this proof applies.

[*$X; S\downarrow$ is locally confluent*].

If $t \longrightarrow_{X;S\downarrow} u$ and $t \longrightarrow_{X;S\downarrow} v$, then there exists w such that $u \longrightarrow^*_{X;S\downarrow} w$ and $v \longrightarrow^*_{X;S\downarrow} w$. Indeed if $t \longrightarrow_X a \longrightarrow_{S\downarrow} u$ and $t \longrightarrow_X b \longrightarrow_{S\downarrow} v$ we know from the local confluence of $S_0 \cup X$ that there exists c such that $a \longrightarrow^*_{S_0 \cup X} c$ and $b \longrightarrow^*_{S_0 \cup X} c$. Normalizing c under S yields the w. This is because by the second point of the proof $u \longrightarrow^*_{X;S\downarrow} w$ and $v \longrightarrow^*_{X;S\downarrow} w$.

[*$S \cup X$ is locally confluent*].

If $t \longrightarrow_{S \cup X} u$ and $t \longrightarrow_{S \cup X} v$, then there exists w such that $u \longrightarrow^*_{S \cup X} w$ and $v \longrightarrow^*_{S \cup X} w$. Indeed call $t^\downarrow$, $u^\downarrow$, $v^\downarrow$ the S normalized version of t, u, v. By the third point of our proof we have $t^\downarrow \longrightarrow^?_{X;S\downarrow} u^\downarrow$ and $t^\downarrow \longrightarrow^?_{X;S\downarrow} v^\downarrow$. By the fourth point of our proof there exists w such that $u^\downarrow \longrightarrow^*_{X;S\downarrow} w$ and $v^\downarrow \longrightarrow^*_{X;S\downarrow} w$.

Proposition 17. *For any scalar rewrite system S the systems $R = S \cup E \cup F \cup A$, $S \cup V$ and $S \cup V'$ are confluent.*

Proof. The system S is confluent and terminating because it is a scalar rewrite system. The system S subsumes S_0 because S is a scalar rewrite system. From Proposition 15, the system $R_0 = S_0 \cup E \cup F \cup A$ is confluent. Finally, the system $E \cup F \cup A$ commutes with S^*. Indeed, each rule of $E \cup F \cup A$ commutes with S^* as each subterm of sort scalar in the left member of a rule is either a variable or 0 or 1, which are normal forms. We conclude with Proposition 16 that $R = S \cup E \cup F \cup A$ is locally confluent. Hence as it is terminating it is confluent by Newman's lemma [39].The same argument applies for $S \cup V$ and $S \cup V'$.

7.5. The system L

We now want to prove that the system L is confluent. With the introduction of the rule B, we lose termination, hence we cannot use Newman's lemma [39] anymore. Thus we shall use for this last part techniques coming from the proof of confluence of the λ-calculus and prove that the parallel version of the B rule is strongly confluent. In our case as we have to mix the rule B with R we shall also prove that it commutes with $\longrightarrow_R^*$.

Definition 20 (Strong confluence). *A relation X is said to be strongly confluent if whenever $t \longrightarrow_X u$ and $t \longrightarrow_X v$, there exists a term w such that $u \longrightarrow_X w$ and $v \longrightarrow_X w$.*

Definition 21 (The relation $\longrightarrow_B^{\|}$). *The relation $\longrightarrow_B^{\|}$ is the smallest reflexive congruence such that if $\mathbf{u}$ is a base vector, $\mathbf{t} \longrightarrow_B^{\|} \mathbf{t}'$ and $\mathbf{u} \longrightarrow_B^{\|} \mathbf{u}'$ then*

$$(\lambda \mathbf{x}\ \mathbf{t})\ \mathbf{u} \longrightarrow_B^{\|} \mathbf{t}'[\mathbf{u}'/\mathbf{x}]$$

Proposition 18. *If $\mathbf{v}_1 \longrightarrow_R^* \mathbf{w}_1$ then $\mathbf{v}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_R^* \mathbf{w}_1[\mathbf{b}/\mathbf{x}]$, where $\mathbf{b}$ is a base vector.*

Proof. If the reduction of $\mathbf{v}_1$ to $\mathbf{w}_1$ involves an application of a conditional rule, then the condition is preserved on $\mathbf{v}_1[\mathbf{v}_2/\mathbf{x}]$. Indeed, substituting some term in a closed normal term yields the same term.

Proposition 19. *If $\mathbf{v}_2 \longrightarrow_R^* \mathbf{w}_2$ then $\mathbf{v}_1[\mathbf{v}_2/\mathbf{x}] \longrightarrow_R^* \mathbf{v}_1[\mathbf{w}_2/\mathbf{x}]$.*

Proof. The reduction is a congruence.

Proposition 20. *If $\mathbf{t} = \mathbf{u}$ or $\mathbf{t} \longrightarrow_R \mathbf{u}$ and if $\mathbf{t} \longrightarrow_B^{\|} \mathbf{v}$ then there exists $\mathbf{w}$ such that $\mathbf{u} \longrightarrow_B^{\|} \mathbf{w}$ and $\mathbf{v} \longrightarrow_R^* \mathbf{w}$.*

Proof. By induction on the structure of $\mathbf{t}$. If $\mathbf{t} = \mathbf{u}$ we just take $\mathbf{w} = \mathbf{v}$. Thus we focus in the rest of the proof to the case where $\mathbf{t} \longrightarrow_R \mathbf{u}$.

If the $B^{\|}$-reduction takes place at the toplevel, then $\mathbf{t} = (\lambda\mathbf{x}\,\mathbf{t}_1)\ \mathbf{t}_2$, $\mathbf{t}_2$ is a base vector and there exists terms $\mathbf{v}_1$ and $\mathbf{v}_2$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$ and $\mathbf{v} = \mathbf{v}_1[\mathbf{v}_2/\mathbf{x}]$. Neither $\lambda\mathbf{x}\,\mathbf{t}_1$ nor $\mathbf{t}_2$ is a sum, a product by a scalar or the null vector, hence the R-reduction is just an application of the congruence thus there exists terms $\mathbf{u}_1$ and $\mathbf{u}_2$ such that and $\mathbf{t}_1 \longrightarrow_R^{?} \mathbf{u}_1$ and $\mathbf{t}_2 \longrightarrow_R^{?} \mathbf{u}_2$. Since $\mathbf{t}_2$ is a base vector, $\mathbf{u}_2$ is also a base vector. By induction hypothesis, there exist terms $\mathbf{w}_1$ and $\mathbf{w}_2$ such that $\mathbf{u}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{v}_1 \longrightarrow_R^{*} \mathbf{w}_1$, $\mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_2$ and $\mathbf{v}_2 \longrightarrow_R^{*} \mathbf{w}_2$. We take $\mathbf{w} = \mathbf{w}_1[\mathbf{w}_2/\mathbf{x}]$. We have $(\lambda\mathbf{x}\,\mathbf{u}_1)\ \mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}$ and by Proposition 18 and 19 we also have $\mathbf{v}_1[\mathbf{v}_2/\mathbf{x}] \longrightarrow_R^{*} \mathbf{w}$.

If the R-reduction takes place at the toplevel, we have to distinguish several cases according to the rule used for this reduction.

- If $\mathbf{t} = \mathbf{t}_1 + \mathbf{0}$ and $\mathbf{u} = \mathbf{t}_1$, then there exists a term $\mathbf{v}_1$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$ and $\mathbf{v} = \mathbf{v}_1 + \mathbf{0}$. We take $\mathbf{w} = \mathbf{v}_1$.
- If $\mathbf{t} = 0.\mathbf{t}_1$ and $\mathbf{u} = \mathbf{0}$, then there exists a term $\mathbf{v}_1$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$ and $\mathbf{v} = 0.\mathbf{v}_1$. We take $\mathbf{w} = \mathbf{0}$.
- If $\mathbf{t} = 1.\mathbf{t}_1$ and $\mathbf{u} = \mathbf{t}_1$, then there exists a term $\mathbf{v}_1$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$ and $\mathbf{v} = 1.\mathbf{v}_1$. We take $\mathbf{w} = \mathbf{v}_1$.
- If $\mathbf{t} = \alpha.\mathbf{0}$ and $\mathbf{u} = \mathbf{0}$, then $\mathbf{v} = \mathbf{t}$. We take $\mathbf{w} = \mathbf{0}$.
- If $\mathbf{t} = \alpha.(\beta.\mathbf{t}_1)$ and $\mathbf{u} = (\alpha \times \beta).\mathbf{t}_1$, then there exists a term $\mathbf{v}_1$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$ and $\mathbf{v} = \alpha.(\beta.\mathbf{v}_1)$. We take $\mathbf{w} = (\alpha \times \beta).\mathbf{v}_1$.
- If $\mathbf{t} = \alpha.(\mathbf{t}_1 + \mathbf{t}_2)$ and $\mathbf{u} = \alpha.\mathbf{t}_1 + \alpha.\mathbf{t}_2$, then there exist terms $\mathbf{v}_1$ and $\mathbf{v}_2$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$ and $\mathbf{v} = \alpha.(\mathbf{v}_1 + \mathbf{v}_2)$. We take $\mathbf{w} = \alpha.\mathbf{v}_1 + \alpha.\mathbf{v}_2$.
- If $\mathbf{t} = \alpha.\mathbf{t}_1 + \beta.\mathbf{t}_1$ and $\mathbf{u} = (\alpha + \beta).\mathbf{t}_1$, then by $(*)$ $\mathbf{t}_1$ is L-normal, thus $\mathbf{v} = \mathbf{t}$. We take $\mathbf{w} = \mathbf{u}$.

  The cases of the two other factorisation rules are similar.
- If $\mathbf{t} = (\mathbf{t}_1 + \mathbf{t}_2)\ \mathbf{t}_3$ and $\mathbf{u} = \mathbf{t}_1\ \mathbf{t}_3 + \mathbf{t}_2\ \mathbf{t}_3$, then by $(**)$ the term $\mathbf{t}_1 + \mathbf{t}_2$ is L-normal. There exists a term $\mathbf{v}_3$ such that $\mathbf{t}_3 \longrightarrow_B^{\|} \mathbf{v}_3$ and $\mathbf{v} = (\mathbf{t}_1 + \mathbf{t}_2)\ \mathbf{v}_3$. We take $\mathbf{w} = \mathbf{t}_1\ \mathbf{v}_3 + \mathbf{t}_2\ \mathbf{v}_3$.
- If $\mathbf{t} = (\alpha.\mathbf{t}_1)\ \mathbf{t}_2$ and $\mathbf{u} = \alpha.(\mathbf{t}_1\ \mathbf{t}_2)$, then by $(***)$ $\mathbf{t}_1$ is L-normal. There exists a term $\mathbf{v}_2$ such that $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$ and $\mathbf{v} = (\alpha.\mathbf{t}_1)\ \mathbf{v}_2$. We take $\mathbf{w} = \alpha.(\mathbf{t}_1\ \mathbf{v}_2)$.
- If $\mathbf{t} = \mathbf{0}\ \mathbf{t}_2$ and $\mathbf{u} = \mathbf{0}$, then there exists a term $\mathbf{v}_2$ such that $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$ and $\mathbf{v} = \mathbf{0}\ \mathbf{v}_2$. We take $\mathbf{w} = \mathbf{0}$.

  The three other cases where a rule of group A is applied are symmetric.

Finally if both reductions are just applications of the congruence we apply the induction hypothesis to the subterms.

Proposition 21 ($\longrightarrow_R^*$ commutes with $\longrightarrow_B^{\|}$).
If $\mathbf{t} \longrightarrow_R^* \mathbf{u}$ *and* $\mathbf{t} \longrightarrow_B^{\|} \mathbf{v}$ *then there exists* $\mathbf{w}$ *such that* $\mathbf{u} \longrightarrow_B^{\|} \mathbf{w}$ *and* $\mathbf{v} \longrightarrow_R^* \mathbf{w}$.

Proof. By induction on the length of the $\longrightarrow_R^*$ derivation. If $\mathbf{t} = \mathbf{u}$ then we take $\mathbf{w} = \mathbf{v}$. Otherwise there exists a term $\mathbf{t}_1$ such that $\mathbf{t} \longrightarrow_R \mathbf{t}_1 \longrightarrow_R^* \mathbf{u}$ with a shorter reduction from $\mathbf{t}_1$ to $\mathbf{u}$. Using Proposition 20, there exists a term $\mathbf{w}_1$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{w}_1$ and $\mathbf{v} \longrightarrow_R^* \mathbf{w}_1$. By induction hypothesis, there exists a term $\mathbf{w}$ such that $\mathbf{u} \longrightarrow_B^{\|} \mathbf{w}$ and $\mathbf{w}_1 \longrightarrow_R^* \mathbf{w}$. We have $\mathbf{u} \longrightarrow_B^{\|} \mathbf{w}$ and $\mathbf{v} \longrightarrow_R^* \mathbf{w}$.

Proposition 22 (Substitution for $B^{\|}$).
If $\mathbf{t} \longrightarrow_B^{\|} \mathbf{t}'$ *and* $\mathbf{b} \longrightarrow_B^{\|} \mathbf{b}'$ *then* $\mathbf{t}[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{b}'[\mathbf{b}'/\mathbf{x}]$. *Here* $\mathbf{b}$ *denotes a base vector.*

Proof. By induction on the structure of $\mathbf{t}$.

- If $\mathbf{t} = \mathbf{x}$ then $\mathbf{t}' = \mathbf{x}$ and hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = \mathbf{b} \longrightarrow_B^{\|} \mathbf{b}' = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.
- If $\mathbf{t} = \mathbf{y}$ then $\mathbf{t}' = \mathbf{y}$ and hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = \mathbf{y} = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.
- If $\mathbf{t} = \lambda \mathbf{y}\, \mathbf{t}_1$ the $B^{\|}$-reduction is just an application of the congruence. We have $\mathbf{t}' = \lambda \mathbf{y}\, .\mathbf{t}'_1$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{t}'_1$ and the induction hypothesis tells us that $\mathbf{t}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_1[\mathbf{b}'/\mathbf{x}]$. Hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = \lambda \mathbf{y}\, \mathbf{t}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \lambda \mathbf{y}\, \mathbf{t}'_1[\mathbf{b}'/\mathbf{x}] = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.
- If $\mathbf{t} = (\mathbf{t}_1\ \mathbf{t}_2)$ then we consider two cases.
  - We have $\mathbf{t}_1 = \lambda \mathbf{y}\ \mathbf{t}_3$, $\mathbf{t}_2$ a base state, and $\mathbf{t}' = \mathbf{t}'_3[\mathbf{t}'_2/\mathbf{x}]$, i.e., a B-reduction occurs at top-level. By induction hypothesis we know that $\mathbf{t}_3[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_3[\mathbf{b}'/\mathbf{x}]$ and $\mathbf{t}_2[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_2[\mathbf{b}'/\mathbf{x}]$. Because $\mathbf{t}_2$ and $\mathbf{b}$ are base vectors, so is $\mathbf{t}_2[\mathbf{b}/\mathbf{x}]$. Hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = (\lambda \mathbf{y}\, \mathbf{t}_3[\mathbf{b}/\mathbf{x}])\mathbf{t}_2[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_3[\mathbf{b}'/\mathbf{x}][\mathbf{t}'_2[\mathbf{b}'/\mathbf{x}]/\mathbf{y}] = \mathbf{t}'_3[\mathbf{t}'_2/\mathbf{y}][\mathbf{b}'/\mathbf{x}] = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.
  - If $\mathbf{t}' = (\mathbf{t}'_1\ \mathbf{t}'_2)$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{t}'_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{t}'_2$, then by induction hypothesis we know that $\mathbf{t}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_1[\mathbf{b}'/\mathbf{x}]$ and $\mathbf{t}_2[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_2[\mathbf{b}'/\mathbf{x}]$. Hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = (\mathbf{t}_1[\mathbf{b}/\mathbf{x}]\, \mathbf{t}_2[\mathbf{b}/\mathbf{x}]) \longrightarrow_B^{\|} (\mathbf{t}'_1[\mathbf{b}'/\mathbf{x}]\, \mathbf{t}'_2[\mathbf{b}'/\mathbf{x}]) = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.
- If $\mathbf{t} = \mathbf{0}$ then $\mathbf{t}' = \mathbf{0}$ and hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = \mathbf{0} = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.
- If $\mathbf{t}$ is a sum the $B^{\|}$-reduction is just an application of the congruence. Therefore $\mathbf{t}$ is AC-equivalent to $\mathbf{t}_1 + \mathbf{t}_2$ and $\mathbf{t}'$ is AC-equivalent to $\mathbf{t}'_1 + \mathbf{t}'_2$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{t}'_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{t}'_2$. Then by induction hypothesis we know that $\mathbf{t}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_1[\mathbf{b}'/\mathbf{x}]$ and $\mathbf{t}_2[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_2[\mathbf{b}'/\mathbf{x}]$. Hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = \mathbf{t}_1[\mathbf{b}/\mathbf{x}] + \mathbf{t}_2[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}'_1[\mathbf{b}'/\mathbf{x}] + \mathbf{t}'_2[\mathbf{b}'/\mathbf{x}] = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.

- If $\mathbf{t} = \alpha.\mathbf{t}_1$ the $B^{\|}$-reduction is just an application of the congruence. We have $\mathbf{t}' = \alpha.\mathbf{t}_1'$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{t}_1'$ and the induction hypothesis tells us that $\mathbf{t}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{t}_1'[\mathbf{b}'/\mathbf{x}]$. Hence $\mathbf{t}[\mathbf{b}/\mathbf{x}] = \alpha.\mathbf{t}_1[\mathbf{b}/\mathbf{x}] \longrightarrow_B^{\|} \alpha.\mathbf{t}_1'[\mathbf{b}'/\mathbf{x}] = \mathbf{t}'[\mathbf{b}'/\mathbf{x}]$.

Proposition 23 (Strong confluence of $B^{\|}$).
If $\mathbf{t} \longrightarrow_B^{\|} \mathbf{u}$ *and* $\mathbf{t} \longrightarrow_B^{\|} \mathbf{v}$ *then there exists* $\mathbf{w}$ *such that* $\mathbf{u} \longrightarrow_B^{\|} \mathbf{w}$ *and* $\mathbf{v} \longrightarrow_B^{\|} \mathbf{w}$.

Proof. By induction on the structure of $\mathbf{t}$.

- If $\mathbf{t}$ is a variable then $\mathbf{u} = \mathbf{t}$ and $\mathbf{v} = \mathbf{t}$. We take $\mathbf{w} = \mathbf{t}$.
- If $\mathbf{t} = \mathbf{0}$ then $\mathbf{u} = \mathbf{0}$ and $\mathbf{v} = \mathbf{0}$. We take $\mathbf{w} = \mathbf{0}$.
- If $\mathbf{t} = \lambda\mathbf{x}\ \mathbf{t}_1$ then $\mathbf{u} = \lambda\mathbf{x}\ \mathbf{u}_1$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{u}_1$ and $\mathbf{v} = \lambda\mathbf{x}\ \mathbf{v}_1$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$. By induction hypothesis, there exists a $\mathbf{w}_1$ such that $\mathbf{u}_1 \longrightarrow_B^{\|} \mathbf{w}_1$ and $\mathbf{v}_1 \longrightarrow_B^{\|} \mathbf{w}_1$. We take $\mathbf{w} = \lambda\mathbf{x}\ \mathbf{w}_1$.
- If $\mathbf{t} = (\mathbf{t}_1\ \mathbf{t}_2)$ then we consider two cases.
  - If the term $\mathbf{t}_1$ has the form $\lambda\mathbf{x}\ \mathbf{t}_3$ and $\mathbf{t}_2$ is a base vector. We consider three subcases, according to the form of the $B^{\|}$-reductions. Either $\mathbf{v} = (\mathbf{v}_1\ \mathbf{v}_2)$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$, and $\mathbf{u} = (\mathbf{u}_1\ \mathbf{u}_2)$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{u}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{u}_2$. By induction hypothesis, there exists terms $\mathbf{w}_1$ and $\mathbf{w}_2$ such that $\mathbf{u}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{v}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_2$, $\mathbf{v}_2 \longrightarrow_B^{\|} \mathbf{w}_2$. We take $\mathbf{w} = (\mathbf{w}_1\ \mathbf{w}_2)$.

    Or $\mathbf{v} = \mathbf{v}_3[\mathbf{v}_2/\mathbf{x}]$ with $\mathbf{t}_3 \longrightarrow_B^{\|} \mathbf{v}_3$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$, and $\mathbf{u} = ((\lambda\mathbf{x}\ \mathbf{u}_3)\ \mathbf{u}_2)$ with $\mathbf{t}_3 \longrightarrow_B^{\|} \mathbf{u}_3$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{u}_2$. Since $\mathbf{t}_2$ is a base vector, $\mathbf{u}_2$ and $\mathbf{v}_2$ are also base vectors. By induction hypothesis, there exist terms $\mathbf{w}_3$ and $\mathbf{w}_2$ such that $\mathbf{u}_3 \longrightarrow_B^{\|} \mathbf{w}_3$, $\mathbf{v}_3 \longrightarrow_B^{\|} \mathbf{w}_3$, $\mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_2$, $\mathbf{v}_2 \longrightarrow_B^{\|} \mathbf{w}_2$. We take $\mathbf{w} = \mathbf{w}_3[\mathbf{w}_2/\mathbf{x}]$. We have $(\lambda\mathbf{x}\,\mathbf{u}_3)\ \mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_3[\mathbf{w}_2/\mathbf{x}]$ by definition of $B^{\|}$. By Proposition 22 we also have $\mathbf{v}_3[\mathbf{v}_2/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{w}_3[\mathbf{w}_2/\mathbf{x}]$.

    Or $\mathbf{v} = \mathbf{v}_3[\mathbf{v}_2/\mathbf{x}]$ with $\mathbf{t}_3 \longrightarrow_B^{\|} \mathbf{v}_3$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$, and $\mathbf{u} == \mathbf{u}_3[\mathbf{u}_2/\mathbf{x}]$ with $\mathbf{t}_3 \longrightarrow_B^{\|} \mathbf{u}_3$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{u}_2$. Since $\mathbf{t}_2$ is a base vector, $\mathbf{u}_2$ and $\mathbf{v}_2$ are base vectors also. By induction hypothesis, there exist terms $\mathbf{w}_3$ and $\mathbf{w}_2$ such that $\mathbf{u}_3 \longrightarrow_B^{\|} \mathbf{w}_3$, $\mathbf{v}_3 \longrightarrow_B^{\|} \mathbf{w}_3$, $\mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_2$, $\mathbf{v}_2 \longrightarrow_B^{\|} \mathbf{w}_2$. We take $\mathbf{w} = \mathbf{w}_3[\mathbf{w}_2/\mathbf{x}]$. By Proposition 22 we have both $\mathbf{u}_3[\mathbf{u}_2/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{w}_3[\mathbf{w}_2/\mathbf{x}]$ and $\mathbf{v}_3[\mathbf{v}_2/\mathbf{x}] \longrightarrow_B^{\|} \mathbf{w}_3[\mathbf{w}_2/\mathbf{x}]$.
  - Otherwise the $B^{\|}$-reduction is just an application of the congruence, i.e., $\mathbf{v} = (\mathbf{v}_1\ \mathbf{v}_2)$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$, and $\mathbf{u} = (\mathbf{u}_1\ \mathbf{u}_2)$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{u}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{u}_2$. By induction hypothesis, there exists terms $\mathbf{w}_1$ and $\mathbf{w}_2$ such that $\mathbf{u}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{v}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_2$, $\mathbf{v}_2 \longrightarrow_B^{\|} \mathbf{w}_2$. We take $\mathbf{w} = (\mathbf{w}_1\ \mathbf{w}_2)$.

- If $\mathbf{t}$ is a sum then the $B^{\|}$-reduction is just an application of the congruence. The term $\mathbf{t}$ is AC-equivalent to a sum $\mathbf{t}_1+\mathbf{t}_2$, the term $\mathbf{u}$ is AC-equivalent to a sum $\mathbf{u}_1+\mathbf{u}_2$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{u}_1$, $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{u}_2$, and the term $\mathbf{v}$ is AC-equivalent to a sum $\mathbf{v}_1+\mathbf{v}_2$ such that $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$ and $\mathbf{t}_2 \longrightarrow_B^{\|} \mathbf{v}_2$. By induction hypothesis, there exist terms $\mathbf{w}_1$ and $\mathbf{w}_2$ such that $\mathbf{u}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{v}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{u}_2 \longrightarrow_B^{\|} \mathbf{w}_2$, $\mathbf{v}_2 \longrightarrow_B^{\|} \mathbf{w}_2$. We take $\mathbf{w} = \mathbf{w}_1 + \mathbf{w}_2$.
- If finally, $\mathbf{t} = \alpha.\mathbf{t}_1$ then the $B^{\|}$-reduction is just an application of the congruence. We have $\mathbf{u} = \alpha.\mathbf{u}_1$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{u}_1$, and $\mathbf{v} = \alpha\mathbf{v}_1$ with $\mathbf{t}_1 \longrightarrow_B^{\|} \mathbf{v}_1$. By induction hypothesis, there exists a term $\mathbf{w}_1$ such that $\mathbf{u}_1 \longrightarrow_B^{\|} \mathbf{w}_1$, $\mathbf{v}_1 \longrightarrow_B^{\|} \mathbf{w}_1$. We take $\mathbf{w} = \alpha.\mathbf{w}_1$.

Proposition 24 (Hindley-Rosen lemma). *If the relations X and Y are strongly confluent and commute then the relation $X \cup Y$ is confluent.*

Theorem 1. *The system L is confluent.*

Proof. By Proposition 17, the relation $\longrightarrow_R$ is confluent, hence $\longrightarrow_R^*$ is strongly confluent. By Proposition 23, the relation $\longrightarrow_B^{\|}$ is strongly confluent. By Proposition 21, the relations $\longrightarrow_R^*$ and $\longrightarrow_B^{\|}$ commute. Hence, by Proposition 24 the relation $\longrightarrow_R^* \cup \longrightarrow_B^{\|}$ is confluent. Hence, the relation $\longrightarrow_L$ is confluent.

Corollary 1 (No-cloning in the Linear-algebraic λ-calculus). *There is no term* Clone *such that for all term $\mathbf{v}$,* (Clone $\mathbf{v}$) $\longrightarrow_L^*$ $(\mathbf{v}\otimes\mathbf{v})$.

Proof. Note that the $\otimes$, **true** and **false** stand for the term introduced in Section 6. Say (Clone $\mathbf{v}$) $\longrightarrow_L^*$ $(\mathbf{v}\otimes\mathbf{v})$ for all $\mathbf{v}$. Let $\mathbf{v} = \alpha.\mathbf{true} + \beta.\mathbf{false}$ be in closed normal form. Then by the A-rules we have (Clone $(\alpha.\mathbf{true} + \beta.\mathbf{false})$) $\longrightarrow_L^*$ $\alpha.$(Clone **true**) + $\beta.$(Clone **false**). Next according to our supposition on Clone this further reduces to $\alpha.(\mathbf{true}\otimes\mathbf{true})+\beta.(\mathbf{false}\otimes\mathbf{false})$. But our supposition on Clone, also says that (Clone $(\alpha.\mathbf{true}+\beta.\mathbf{false})$) reduces to $(\alpha.\mathbf{true}+\beta.\mathbf{false})\otimes(\alpha.\mathbf{true}+\beta.\mathbf{false})$. Moreover the two cannot be reconciled into a common reduct, because they are normal. Hence our supposition would break the confluence; it cannot hold. Note that $\lambda\mathbf{x}\,\mathbf{v}$ on the other hand can be duplicated, because it is thought as the (plans of) the classical machine for building $\mathbf{v}$ – in other words it stands for potential parallelism rather than actual parallelism. As expected there is no way to transform $\mathbf{v}$ into $\lambda\mathbf{x}\,\mathbf{v}$ in general; confluence ensures that the calculus handles this distinction in a consistent manner.

8. Current works

8.1. Algebraic λ-calculus

As we have mentioned in the introduction the idea of endowing the λ-calculus with a vector space has emerged simultaneously and independently in a different context. Indeed, the exponential-free fragment of Linear Logic is a logic of resources where the propositions themselves stand for those resources – and hence cannot be discarded nor copied. When seeking to find models of this logic, one obtains a particular family of vector spaces and differentiable functions over these. It is by trying to capture back these mathematical structures into a programming language that T. Ehrhard and L. Regnier have defined the *differential λ-calculus* [26], which has an intriguing differential operator as a built-in primitive, and some notion of module of the λ-calculus terms, over the natural numbers. More recently L. Vaux [57] has focused his attention on a "differential λ-calculus without differential operator", extending the module to finitely splitting positive real numbers. He obtained a confluence result in this case, which stands even in the untyped setting. More recent works on this *Algebraic λ-calculus* tend to consider arbitrary scalars [28, 49]. This Algebraic λ-calculus and the Linear-algebraic λ-calculus we presented in this paper are very similar not only in names: they both merge higher-order computation, be it terminating or not, in its simplest and most general form (namely the untyped λ-calculus) together with linear algebra in its simplest and most general form also (the axioms of vector spaces). Skipping over details a closer inspection unravels that:

- the application in the Algebraic λ-calculus is left linear but not right linear;
- the abstraction in the Algebraic λ-calculus is a linear unary operator;
- the rewriting is modulo vector space axioms, and these axioms are not transformed into rewrite rules of the system.

It could be said the last two points are only minor differences; design choices in some sense. Arguably those of *Lineal* are advantageous because they yield a more robust confluence proof, valid for arbitrary scalars. If we lift these two differences, *Lineal* simulates the Algebraic λ-calculus [12]. The first point is a more important difference, with justification right within the origins of the Algebraic λ-calculus and the Differential λ-calculus. Recently, however, it has been shown that the difference really amounts to a choice between call-by-name and call-by-value oriented strategies. The encoding of one strategy into another still works [23] — hence it could be said that the two calculi are essentially equivalent.

8.2. Types

Whilst terms in our calculus seem to form a vector space, the very definition of a norm is difficult in our context: deciding whether a term terminates is undecidable; but these terms produce infinities, hence convergence of a vector space norm is undecidable. Related to this precise topic, L. Vaux has studied simply typed algebraic λ-calculus, ensuring convergence of a vector space norm [57]. Following his work, C. Tasson has studied some model-theoretic properties of the *barycentric* ($\sum \alpha_i = 1$) subset of this simply typed calculus [49]. A recent work by T. Ehrhard proves the convergence of a Taylor series expansion of Algebraic λ-calculus terms, via a System F typing system [28].

Hence, standard type systems ensure the convergence of the vector space norm of a term. And indeed it is not so hard to define a simple extension of System F that fits *Lineal* — just by providing the needed rules to type additions, scalar products and the null vector in some trivial manner, as we did in [10, 11]. As expected one obtains strong normalisation from this type system. An important byproduct of this result is that one can then remove the conditions $(*) - (***)$ that limit the reduction rules of *Lineal* (see Section 5), because their purpose was really to keep indefinite forms from reducing (such as $\mathbf{t} - \mathbf{t}$, with $\mathbf{t}$ not normal and hence potentially infinite). In other words types make *Lineal* into a simpler language.
Yet standard type systems are unable for instance to impose upon the language that any well-typed linear combination of terms $\sum \alpha_i.\mathbf{t}_i$ has $\sum \alpha_i = 1$. That is unless they are provided with a handle upon these scalars. This is the purpose of the *scalar* type system which was recently proposed [10, 11]. This type system which manages to keep track of "the amount of a type" by summing the amplitudes of its contributing terms, and reflects this amount within the type. As an example of its uses, it was demonstrated that this provides a type system which guarantees well-definiteness of probabilistic functions in the sense that it specializes *Lineal* into a probabilistic, higher-order λ-calculus. We are still looking for a type system that would impose that linear combination of terms $\sum \alpha_i.\mathbf{t}_i$ have $\sum |\alpha_i|^2 = 1$, as suited for quantum computing.

8.3. Models

The functions expressed in our language are linear operators upon the space constituted by its terms. It is strongly inspired from the more preliminary [6], where terms clearly formed a vector space. However because the calculus higher-order, we get forms of infinities coming into the game. Thus, the underlying algebraic structure is not as obvious as in [6]. Moreover one can notice already that since the non-trivial models of the untyped λ-calculus are all uncountable, the models of (Linear-)Algebraic λ-calculus are likely to be vector spaces having an uncountable basis. These are fascinating, open questions, but whose difficulty explain why we have not provided a denotational semantics for *Lineal* in this paper. This issue of models of (Linear-)Algebraic λ-calculus is a challenging, active topic of current research. We know of the categorical model of simply typed *Lineal* with fixpoints [51], which establishes a connection between the canon and uncanon construct of Section 6 and monads *à la* Moggi [38]. The finiteness space model of simply typed Algebraic λ-calculus [27, 49] does not easily carry through to *Lineal*, which is call-by-value oriented. Recently, a syntactic finiteness space model of System F algebraic λ-calculus has been developed in [28].

9. Conclusion

9.1. Summary

When merging the untyped λ-calculus with linear algebra one faces two different problems. First of all simple-minded duplication of a vector is a non-linear

operation (cloning) unless it is restricted to base vectors and later extended linearly (copying). Second of all we can express computable but nonetheless infinite series of vectors, hence yielding some infinities and the troublesome indefinite forms. Here again this is fixed by restricting the evaluation of these indefinite forms, this time to normal vectors. Both problems show up when looking at the confluence of the Linear-algebraic λ-calculus (*Lineal*).
The architecture of the proof of confluence seems well-suited to any non-trivial rewrite systems having both some linear algebra and some infinities as its key ingredients. Moreover the proof of confluence entails a no-cloning result for Lineal, in accordance with the linearity of quantum physics.

9.2. Perspectives

Lineal merges higher-order computation with linear algebra in a minimalistic manner. Such a foundational approach is also taking place for instance in [2] via some categorical formulations of quantum theory exhibiting nice composition laws and normal forms, without explicit states, fixed point or the possibility to replicate gate descriptions. As for [2] although we have shown that quantum computation can be encoded in our language, *Lineal* remains some way apart from a model of quantum computation, because it allows evolutions which are not unitary. Establishing formal connections with this categorical approach does not seem an easy matter but is part of our objectives.
These connections might arise through typing. Finding a type system which specializes *Lineal* into a strictly quantum programming language (enforcing the unitary constraint) is not only our next step on the list, it is actually the principal aim and motivation for this work: we wish to extend the Curry-Howard isomorphism between proofs/propositions and programs/types to a linear-algebraic, quantum setting. Having merged higher-order computation with linear-algebra in a minimalistic manner, which does not depend on any particular type systems, grants us a complete liberty to now explore different forms of this isomorphism. For instance we may expect different type systems to have different fields of application, ranging from fine-grained entanglement-analysis for quantum computation [44, 45], to opening connections with linear logic [24] or even giving rise to some novel, quantitative logics [10].

Acknowledgments

The authors would like to thank Alejandro Díaz-Caro, Evelyne Contejean, Philippe Jorrand, Jean-Pierre Jouannaud, Claude Marché, Simon Perdrix, Benoît Valiron and Lionel Vaux for some enlightening discussions.

References

[1] S. Abramsky, *Computational Interpretations of Linear Logic*, Theoretical Computer Science, **111**, 3–57, (1993).

[2] S. Abramsky, B. Coecke, *A categorical semantics of quantum protocols* LICS, IEEE Computer Society, 415-425, (2004).

[3] L. Adleman, J. DeMarrais, M. Huang, *Quantum Computability*, SIAM J. on Comp., **26**, 5, 1524-1540, (1997).

[4] T. Altenkirch, J. Grattage, J.K. Vizzotto, A. Sabry, An Algebra of Pure Quantum Programming, *Third International Workshop on Quantum Programming Languages*, Electronic Notes of Theoretical Computer Science, 170C, 23-47, (2007).

[5] P. Arrighi, G. Dowek, *A computational definition of the notion of vector space*, ENTCS **117**, 249-261, (2005).

[6] P. Arrighi, G. Dowek, *Linear-algebraic lambda-calculus*, in P. Selinger (Ed.), International workshop on quantum programming languages, Turku Centre for Computer Science General Publication, **33**, 21-38, (2004).

[7] P. Arrighi, G. Dowek, *Linear-algebraic lambda-calculus: higher-order, encodings, confluence*, arXiv:quant-ph/0612199.

[8] P. Arrighi, G. Dowek, `www-roc.inria.fr/who/Gilles.Dowek/Prog/lineal.html`.

[9] P. Arrighi, G. Dowek, *On the critical pairs of a rewrite system for vector spaces*, available on the web page of the authors, see `www-roc.inria.fr/who/Gilles.Dowek/Publi/criticalpairs.pdf`, (2012).

[10] P. Arrighi, A. Díaz-Caro, *Scalar System F for Linear-Algebraic λ-Calculus: Towards a Quantum Physical Logic*, Proceedings of the 6th International Workshop on Quantum Physics and Logic, ENTCS 206–215, (2009).

[11] P. Arrighi, A. Díaz-Caro, *A System F accounting for scalars*, Preprint: arXiv:0903.3741, (2009).

[12] P. Arrighi, L. Vaux, *Embedding Algebraic Lambda-calculus into Lineal*, Private communication, (2009).

[13] E. Bernstein, U. Vazirani, *Quantum Complexity Theory*, Annual ACM symposium on Theory of Computing, **25**, (1993).

[14] G. Birkhoff, *On the Structure of Abstract Algebras*, Proc. Cambridge Phil. Soc., **31**, (1935).

[15] G. Boudol, *Lambda-calculi for (strict) parallel functions*, Information and Computation, **108**(1), 51-127, (1994).

[16] O. Bournez, M. Hoyrup, *Rewriting Logic and Probabilities*, Rewriting Techniques and Applications, LNCS **2706**, (2003).

[17] P. Boykin, T. Mor, M. Pulver, V. Roychowdhury, F. Vatan, *On universal and fault-taulerant quantum computing*, arxiv:quant-ph/9906054

[18] G. Chiribella, G. D'Ariano, P. Perinotti, B. Valiron, *Beyond Quantum Computers*, Arxiv preprint arXiv:0912.0195, (2009).

[19] *The CiME Rewrite Tool*, `http://cime.lri.fr/`.

[20] D. Cohen, P. Watson, *An efficient representation of arithmetic for term rewriting*, Proc. of the 4th Conference on Rewrite Techniques and Applications, LNCS

[21] N. Dershowitz, J.-P. Jouannaud, *Rewrite systems*, Handbook of theoretical computer science, Vol. **B**: formal models and semantics, MIT press, (1991).

[22] D. Deutsch, R. Josza, *Rapid solution of problems by quantum computation.* Proc. of the Roy. Soc. of London A, **439**, 553-558, (1992).

[23] A. Díaz-Caro, S. Perdrix, C. Tasson, B. Valiron *Equivalence of Algebraic λ-calculi*, HOR 2010.

[24] A. Díaz-Caro, B. Petit, *From Additive Logic to Linear Logic*, manuscript, (2010).

[25] D. Dougherty, *Adding Algebraic Rewriting to the Untyped Lambda Calculus*, Proc. of the Fourth International Conference on Rewriting Techniques and Applications, 1992.

[26] T. Ehrhard, L. Regnier, *The differential lambda-calculus*, Theoretical Computer Science, **309**, 1–41, (2003).

[27] T. Ehrhard, *Finiteness spaces*, Mathematical Structures in Computer Science, **15**(4), 615–646, (2005).

[28] T. Ehrhard, *A finiteness structure on resource terms*, LICS 2010, to appear.

[29] M. Fernandez and I. Mackie, *Closed Reductions in the λ-calculus*, Computer Science Logic, Lecture Notes in Computer Science 1683, (1999).

[30] A. Di Pierro, C. Hankin, H. Wiklicky, *Probabilistic λ-calculus and quantitative program analysis*, J. of Logic and Computation, **15**(2), 159-179, (2005).

[31] S. J. Gay, *Quantum programming languages: survey and bibliography*, Mathematical Structures in Computer Science, **16**(4), 581–600, (2006).

[32] J.-Y. Girard. *Linear logic.* Theoretical Computer Science, **50**, 1-102, (1987).

[33] L. K. Grover, *Quantum Mechanics Helps in Searching for a Needle in a Haystack*, Phys. Rev. Lett., **79**(2), 325–328, (1997).

[34] O. M. Herescu, C. Palamidessi, *Probabilistic asynchronous pi-calculus*, ETAPS, LNCS **1784**, 146–160, (2000).

[35] G. Huet, *A complete proof of correctness of the Knuth-Bendix completion algorithm*, Journal of Computer and System Sciences, **23**(1), pages 11–21, (1981).

[36] J.-P. Jouannaud, H. Kirchner, *Completion of a Set of Rules Modulo a Set of Equations*, SIAM J. of Computing, **15**(4), 1155–1194, (1986).

[37] A. Kitaev, *Quantum computation, algorithms and error correction*, Russ. Math. Surv., **52**, 6, 1191-1249, (1997).

[38] E. Moggi, *Notions of computation and monads*, Information and Computation, **93**, 55–92, (1991).

[39] M. H. A. Newman, *On theories with a combinatorial definition of "equivalence"*, Annals of Mathematics, **43**2, 223–243, (1942).

[40] M. A. Nielsen, *Universal quantum computation using only projective measurement, quantum memory, and preparation of the 0 state*, Phys. Rev. A, **308**, 96-100, (2003).

[41] , O. Oreshkov, F. Costa, C. Brukner, *Quantum correlations with no causal order*, Arxiv preprint arXiv:1105.4464, (2011).

[42] G. E. Peterson, M. E. Stickel, *Complete Sets of Reductions for Some Equational Theories*, J. ACM, **28**(2), 233-264, (1981).

[43] S. Perdrix, *State transfer instead of teleportation in measurement-based quantum computation*, Int. J. of Quantum Information , **1**(1), 219-223, (2005).

[44] S. Perdrix, *Quantum entanglement analysis based on abstract interpretation*, SAS 2008, LNCS **5079**, (2008).

[45] F. Prost, C. Zerrari, *Reasoning about Entanglement and Separability in Quantum Higher-Order Functions*, UC 2008, Proceedings of the 8th International Conference on Unconventional Computation, 219–235 (2009).

[46] R. Raussendorf, D.E. Browne, H.J. Briegel, *The one-way quantum computer - a non-network model of quantum computation*, Journal of Modern Optics, **49**, p. 1299, (2002).

[47] T. Rudolph, L. Grover, *A two rebit gate universal for quantum computing*, october 2002, arxiv:quant-ph/0210187.

[48] P. Selinger, *Towards a quantum programming language*, Math. Struc. in Computer Science, **14**(4), 527-586, (2004).

[49] C. Tasson, *Algebraic Totality, towards Completeness*, TLCA 2009: Proceedings of the 9th International Conference on Typed Lambda Calculi and Applications, 325–340, (2009).

[50] P. Selinger, B. Valiron, *A lambda calculus for quantum computation with classical control*, Math. Struc. in Computer Science, **16**(3), 527-552, (2006).

[51] B. Valiron, *A Typed, Algebraic, Computational Lambda-Calculus.* Mathematical Structures in Computer Science (to appear).

[52] P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J. on Computing, **26**, 1484-1509, (1997).

[53] R. Solovay, manuscript, (1995).

[54] R. Solovay, A. Yao, *Quantum Circuit Complexity and Universal Quantum Turing Machines*, manuscript, (1996).

[55] A. Van Tonder, *A Lambda Calculus for Quantum Computation*, july 2003, arXiv:quant-ph/0307150.

[56] A. Van Tonder, *Quantum Computation, Categorical Semantics and Linear Logic*, december 2003, arXiv:quant-ph/0312174.

[57] L. Vaux, *On linear combinations of lambda-terms*, Proceedings of RTA 2007, LNCS **4533**, (2007).

[58] H. Walters, H. Zantema, *Rewrite systems for integer arithmetic*, Proc. of Rewriting Techniques and Applications 94, 6th Int. Conf., LNCS **914**, 324-338, (1995).

[59] W. K. Wooters, W. H. Zurek, *A single quantum cannot be cloned*, Nature **299**, 802-803, (1982).