Proc. QPL 2004, pp. 21–38

Operational semantics for formal tensorial calculus

Pablo Arrighi* Gilles Dowek†

Abstract

With a view towards models of quantum computation, we define a functional language where all functions are linear operators by construction. A small step operational semantic (and hence an interpreter/simulator) is provided for this language in the form of a term rewrite systems. The *linear-algebraic λ-calculus* hereby constructed is linear in a different (yet related) sense to that, say, of the linear λ-calculus. These various notions of linearity are discussed in the context of quantum programming languages .

1 Introduction

Quantum computation lacks a convenient model of computation. To this day its algorithms are expressed in terms of quantum circuits, but their descriptions always seem astonishingly remote from the task they do accomplish [12]. Moreover universality is only provided via the notion of uniform family of circuits [29]. Quantum Turing machines solve this latter point, yet they are even less suitable as a programming language [6]. Another approach is to enclose quantum circuits within a classical imperative-style control structure [21] - but we wish to avoid this duality, in an attempt to bring programs closer to their specifications. Functional-style control structure, on the other hand, seem to merge with quantum evolution descriptions in a unifying manner. With a view towards models of quantum computation, we describe a functional language for expressing linear operators, and linear operators only.

We provide a semantic for this language in the form of a *term rewrite system* [11]. These consist in a finite set of rules $l \longrightarrow r$, each interpreted as follows: *"Any term t containing a subterm σl in position p (i.e. $t = t[\sigma l]_p$) should be rewritten into a term t' containing σr in position p, with all the rest unchanged (i.e. $t' = t[\sigma r]_p$)"*. The minimalist interpretation of the rules makes term rewrite systems (TRS) extremely suitable for describing the behavior of a computer languages unambiguously - so long as the order in which the substitution occur does not matter to the end result (a property named confluence). Moreover, because $l \longrightarrow r$ may be seen as an oriented version of equation $l = r$, the TRS provides both an operational semantic (an interpreter/simulator for the language)

*Institut Gaspard Monge, 5 Bd Descartes, Champs-sur-Marne, 77574 Marne-la-Vallée Cedex 2, France, arrighi@univ-mlv.fr.

†École polytechnique and INRIA, LIX, École polytechnique, 91128 Palaiseau Cedex, France, Gilles.Dowek@polytechnique.fr.

and an axiomatic semantic (an equational theory in which to prove properties about the language).

We begin by orienting the equations which axiomatize vectorial spaces and tensor products, to yield an algorithm for reducing vectors to a linear combination of base vectors, e.g.

$$\lambda.(\mu.\mathbf{u}) \longrightarrow (\lambda \times \mu).\mathbf{u}.$$

Completed with few additional rules such as

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

the term rewrite system is shown terminating and confluent. Remarkably this in turn provides a computational definition of vectorial spaces and their tensors, as any mathematical structure validating the algorithm (Section 2).

In order to lay the ground for a rigorous definition of quantum functional languages, the term rewrite system must also cater for the complex scalars upon which its vectors live. This raises the problem of the conditional rewriting required for division, which we can circumvent, basing quantum computation upon the ring of dyadic floats together with $\frac{1}{\sqrt{2}}$ and imaginary number i (Section 3).

Modern days functional languages such as Caml, Haskell etc. are based upon two basic evaluation mechanisms: *matching*, which provides conditional branching by inspection of variables values; and some avatar of the λ-*calculus*, which provides both control flow and data flow.

The first mechanism is obtained as we extend the term rewrite system to handle linear maps - themselves denoted as superpositions of bipartite states, e.g.

$$(\mathbf{true} \rhd \mathbf{false} + \mathbf{true} \rhd \mathbf{false}) * \mathbf{false} \longrightarrow^* \mathbf{false} + \mathbf{true}.$$

Applications are therefore analogous to contractions in tensorial calculus. This approach offers an elegant paradigm to represent quantum operations as quantum states, which we argue may be necessary a feature for quantum higher-order programming (Section 4).

The second mechanism is obtained through an implementation of λ-terms via de Bruijn indices, a scheme whereby variables are encoded as integers referring to their binders, e.g.

$$\lambda x.(\lambda y.(x \otimes y)) \text{ is encoded as } L(L(\mathbf{var}(1) \otimes \mathbf{var}(0))).$$

The question of the interpretation of terms such as $\lambda x.(x \otimes x)$ is lengthily addressed as we draw a distinction between *cloning* and *copying*. The semantic of our calculus forbids only the former, non-linear operation, by enforcing a higher priority of the addition's distributivity over substitution (Section 5).

Erasure on the other hand remains allowed in our calculus, because we do not restrict ourselves to unitary operations. Whilst we discuss possible well-formedness conditions to implement this restriction (a crucial one for quantum computation), the claim here is to have provided a "*linear*", λ-calculus, in the sense of *linear algebra*. We discuss the various notions of 'linearity' used in quantum programming languages, such as the one by Van Tonder [24] (Section 6).

$$\begin{aligned}
\lambda.(\mathbf{u}+\mathbf{v}) &\longrightarrow \lambda.\mathbf{u}+\lambda.\mathbf{v}\\
\lambda.\mathbf{u}+\mu.\mathbf{u} &\longrightarrow (\lambda+\mu).\mathbf{u}\\
\lambda.(\mu.\mathbf{u}) &\longrightarrow (\lambda\times\mu).\mathbf{u}\\
\mathbf{u}+\mathbf{0} &\longrightarrow \mathbf{u}\\
1.\mathbf{u} &\longrightarrow \mathbf{u}\\
0.\mathbf{u} &\longrightarrow \mathbf{0}
\end{aligned}$$

with + an AC symbol.

Figure 1: VECTORIAL SPACES: AXIOMS

2 Vectorial spaces

We seek to represent quantum programs, their input vectors, their output vectors and their applications as terms of a first-order language. Moreover we seek to provide rules such that the term formed by the application of a quantum program onto its input vector should reduce to its output vector. Several terms may be used to express one output vector, as a consequence we must ensure that these all reduce to one unique, *normal form*, upon which there is nothing more to compute.
Since quantum theory is based upon vectorial spaces it seems natural to consider $\mathfrak{L}$ a two-sorted language having sort K for scalars and sort E for vectors - together with: two constants 0 and 1 of sort K; a constant $\mathbf{0}$ of sort E; two binary symbols $+$ and $\times$ of rank $\langle K,K,K\rangle$; a binary symbol $+$ (also) of rank $\langle E,E,E\rangle$; and a binary symbol . of rank $\langle K,E,E\rangle$. Moreover the most natural normal form to aim for is that of a linear combination of the unknowns, i.e. the computation finishes once we have to coordinates of the vector. For instance we must develop:

$$4.(\mathbf{false}+\mathbf{true}) \longrightarrow 4.\mathbf{false}+4.\mathbf{true}$$

But factorize:

$$4.\mathbf{false}+6.\mathbf{false} \longrightarrow 10.\mathbf{false}$$

A TRS which has this effect can be obtained by orienting the 8 equations axiomatizing vectorial spaces, as given in Figure 1. Only those two rules corresponding to associativity and commutativity of vector addition are missing - because we use rewriting modulo AC(+). For confluence we need to add three more rules, as given in Figure 2. The TRS containing these 9 rules is denoted R and has all the desired properties, which we now list. For in-depth discussion and proofs the reader is referred to [4].

Proposition 1 *For any terminating rewrite system S on scalars, the system $R\cup S$ terminates.*

This implies in particular the system R terminates on its own. To prove confluence of $R\cup S$, we first prove, by a simple analysis of critical pairs, the confluence of the system $R\cup S_0$ (Figure 3) and we use the following proposition:

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

Figure 2: VECTORIAL SPACES: CONFLUENCE

$$0 + \lambda \longrightarrow \lambda$$
$$0 \times \lambda \longrightarrow 0$$
$$1 \times \lambda \longrightarrow \lambda$$
$$\lambda \times (\mu + \nu) \longrightarrow (\lambda \times \mu) + (\lambda \times \nu)$$

where + and $\times$ are AC symbols.

Figure 3: SCALAR MINIMAL RULES

Proposition 2 *Let R, S, and S_0 be three relations defined on a set such that $R \cup S_0$ is confluent, S is confluent and terminating, S subsumes S_0, and R commutes with the reflexive-transitive closure of S. Then the relation $R \cup S$ is confluent.*

Corollary 2.1 *Let S be a ground confluent and terminating rewrite system on scalars subsuming S_0. Then the rewrite system $R \cup S$ is confluent on terms containing vector variables but no scalar variables.*

Proposition 3 *Let $\mathbf{t}$ be a normal term whose free variables are amongst $\mathbf{x}_1, ..., \mathbf{x}_n$. The term $\mathbf{t}$ is $\mathbf{0}$ or it has the form $\lambda_1 \mathbf{x}_{i_1} + ... \lambda_k \mathbf{x}_{i_k} + \mathbf{x}_{i_{k+1}} + \mathbf{x}_{i_{k+l}}$ where the indices $i_1, ..., i_{k+l}$ are distinct and the λ_k's are neither 0 nor 1.*

Note that the algorithm defined by R is relatively common in computing, for presenting any number as a linear combination of unknowns. But it does in fact define vector spaces - as any mathematical structure validating the algorithm. Furthermore note that support for tensor products is easily added into the TRS, through the six rules given in Figure 4. Proposition 1 to 2 remain true when R is extended with those six additional rules, whilst proposition 3 now yields normal forms for terms in $E \otimes E$ of the form $\mathbf{0}$ or

$$\lambda_1 \mathbf{x}_{i_1} \otimes \mathbf{y}_{j_1} + \ldots + \lambda_k \mathbf{x}_{i_k} \otimes \mathbf{y}_{j_k} + \mathbf{x}_{i_{k+1}} \otimes \mathbf{y}_{j_{k+1}} + \ldots + \mathbf{x}_{i_{k+l}} \otimes \mathbf{y}_{j_{k+l}},$$

where the pairs of indices $< i_1, j_1 >, \ldots, < i_{k+l}, j_{k+l} >$ are distinct and the λ_k's are neither 0 nor 1 [4].

3 The field of quantum computing

Fields are not easily implemented as term rewrite systems, because of the conditional rewriting required for the division by zero. In the previous section such problems were

$$
\begin{aligned}
(\mathbf{u}+\mathbf{v})\otimes\mathbf{w} &\longrightarrow \mathbf{u}\otimes\mathbf{w}+\mathbf{v}\otimes\mathbf{w}\\
(\lambda.\mathbf{u})\otimes\mathbf{v} &\longrightarrow \lambda.(\mathbf{u}\otimes\mathbf{v})\\
\mathbf{u}\otimes(\mathbf{v}+\mathbf{w}) &\longrightarrow \mathbf{u}\otimes\mathbf{v}+\mathbf{u}\otimes\mathbf{w}\\
\mathbf{u}\otimes(\lambda.\mathbf{v}) &\longrightarrow \lambda.(\mathbf{u}\otimes\mathbf{v})\\
\mathbf{0}\otimes\mathbf{u} &\longrightarrow \mathbf{0}\\
\mathbf{u}\otimes\mathbf{0} &\longrightarrow \mathbf{0}
\end{aligned}
$$

Figure 4: VECTORIAL SPACES: TENSORS

avoided by simply assuming a TRS for scalars having a certain number of basic rules, but if the objective is to lay the ground for formal quantum programming languages then we must provide such a TRS. The present section briefly outlines how this is achieved.

3.1 Background

We seek to model quantum computation as a formal rewrite system upon a finite set of symbols. Since the complex numbers are uncountable, we must therefore depart from using the whole of $\mathbb{C}$ as the field $\mathbb{K}$ of our vector space. Such considerations are commonplace in computation theory, and were successfully addressed with the provision of the first rigorous definition of a quantum Turing machine [6]. In short the quantum Turing machines are brought as an extension of probabilistic Turing machines

$$
\begin{aligned}
&<Q\text{: head states, } \Sigma\text{: alphabet,}\\
&\quad \delta\text{: transition function, } q_o, q_f\text{: start,end state}>
\end{aligned}
$$

whose transition functions are no longer valued over the efficiently computable positive reals (probabilities)

$$\delta : Q \times \Sigma \longrightarrow (Q \times \Sigma \times \{Left,\, Right\} \to \tilde{\mathbb{R}}^{+})$$

but over the efficiently computable complex numbers (amplitudes)

$$\delta : Q \times \Sigma \longrightarrow (Q \times \Sigma \times \{Left,\, Right\} \to \tilde{\mathbb{C}}).$$

In both cases δ is constrained to be a *unit* function (probabilities/squared modulus summing to one), and for the quantum Turing machine δ is additionally required to induce a *unitary* global evolution. A well-known result of complexity theory is that probabilistic Turing machines remain as powerful when the transition function δ is further restricted to take values in the set $\{0, \frac{1}{2}, 1\}$. The result in [6] is analogous: quantum Turing machines remain as powerful when the transition function δ is further restricted to take values in the set $\{-1, -\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}, 1\}$. Later it was shown in [2], and independently in [23] that no irrational number is necessary, i.e. δ may be restricted to take values in the set $\{-1, -\frac{8}{5}, -\frac{3}{5}, 0, \frac{3}{5}, \frac{8}{5}, 1\}$ without loss of power for the quantum Turing machine.

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. This line of research (cf. [22, 16] to cite a few) has so far culminated with [7], 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 above strict sense. A weaker requirement for a set of gates is the ability to *simulate* any unitary transform, a notion which is also referred to as *encoded universality* - since a computation on n qubits may for instance be represented as a computation on $n+1$ 'real bits', through a simple mapping. A recent paper shows that the gate

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & -b \\ 0 & 0 & b & a \end{pmatrix},$$

with either $a = b = \frac{1}{\sqrt{2}}$, or $a = \frac{3}{5}$ and $b = \frac{8}{5}$, has this property [20]. Do appreciate how the result falls into line with those regarding the quantum Turing machine.

Definition 1 *We call computational scalars, and denote $\tilde{\mathbb{K}}$ the ring formed by the additive and multiplicative closure of the complex numbers $\{-1, 1, \frac{1}{\sqrt{2}}, i\}$.*

Once we have shown that the computational scalars arithmetics can be performed by a TRS, it will be sufficient to express the basic gates (1) in our formalism to immediately obtain the more traditional notion of quantum computation universality. Hence our choice.

3.2 Rules

We begin by implementing natural numbers and unsigned binary numbers. That such TRS can be made ground confluent and terminating are now well-established results [9, 28]. This places us in a position to build up dyadic floats out of a sign, an unsigned binary number and an exponent, e.g. $\mathrm{fl}(\mathrm{neg}, 1, S(\mathrm{zeron}))$ is to stand for $-\frac{1}{2}$, as exemplified in Figure 5.

Reached this point it suffices to notice that $\tilde{\mathbb{K}}$, i.e. dyadic floats together with imaginary number i and real number $\frac{1}{\sqrt{2}}$, can be viewed as a four-dimensional module upon dyadic floats. Indeed any such number could be represented as a linear combination of the form:

$$\alpha.\mathbf{1} + \beta.\frac{\mathbf{1}}{\sqrt{\mathbf{2}}} + \gamma.\mathbf{i} + \delta.\frac{\mathbf{i}}{\sqrt{\mathbf{2}}}.$$

As a consequence we can reuse the results of section 2 to implement computational scalars and their additions. Computational scalars multiplication then needs to be defined, we do

$$\begin{aligned}
\text{fl}(s, n :: 0, S(p)) &\longrightarrow \text{fl}(s, n, p) \\
&\quad \%\text{to remove trailing zeroes after the point} \\
\text{fl}(\text{neg}, 0, p) &\longrightarrow \text{fl}(\text{pos}, 0, p) \quad \%\text{to normalize zero} \\
\text{fl}(s, 0, S(p)) &\longrightarrow \text{fl}(s, 0, \text{zeron}) \quad \%\text{to normalize zero} \\
&\vdots \\
\text{fl}(\text{pos}, m_1, e_1) \text{ timesf } \text{fl}(\text{neg}, m_2, e_2) &\longrightarrow \text{fl}(\text{neg}, m1 \text{ timesb } m2, \text{addn}(e_1, e_2)) \\
\text{fl}(\text{neg}, m_1, e_1) \text{ timesf } \text{fl}(\text{pos}, m_2, e_2) &\longrightarrow \text{fl}(\text{neg}, m_1 \text{ timesb } m_2, \text{addn}(e_1, e_2)) \\
&\vdots
\end{aligned}$$

Figure 5: DYADIC FLOATS

so modulo AC in Figure 6. Augmented with these rules the TRS remains ground confluent, as one may check through the method outlined below:

- First showing that $\mathbf{t} \longrightarrow \mathbf{t}'$ implies $[\![\mathbf{t}]\!] = [\![\mathbf{t}']\!]$, where $[\![]\!] : \mathfrak{L} \longrightarrow \mathbb{C}$ is the valuation defined so that

$$\begin{aligned}
&[\![\mathbf{1}]\!] = 1,\ [\![\mathbf{i}]\!] = i,\ [\![\mathbf{1}/\sqrt{\mathbf{2}}]\!] = 1/\sqrt{2},\ [\![\mathbf{i}/\sqrt{\mathbf{2}}]\!] = i/\sqrt{2}, \\
&[\![\lambda.\mathbf{u}]\!] = [\![\lambda]\!] * [\![\mathbf{u}]\!],\ [\![\text{fl}(s, b, e)]\!] = [\![s]\!] * [\![b]\!] * [\![e]\!], \\
&[\![\text{pos}]\!] = -[\![\text{neg}]\!] = 1,\ [\![S(p)]\!] = [\![p]\!]/2,\ [\![\text{zeron}]\!] = 1, \\
&[\![b :: 0]\!] = 2 * [\![b]\!],\ [\![b :: 1]\!] = 1 + 2 * [\![b]\!], \\
&[\![\lambda \times \mu]\!] = \lambda * \mu,\ [\![\lambda + \mu]\!] = \lambda + \mu.
\end{aligned}$$

  (done by examining each rule in the TRS).

- Second showing that $[\![\mathbf{t}]\!] = [\![\mathbf{u}]\!]$, with $\mathbf{t}$ and $\mathbf{u}$ normal implies that $\mathbf{t} =_{\text{AC}} \mathbf{u}$.

Moreover we believe the TRS is also terminating, but have not yet a formal proof for this assertion.
Notice we have never defined a division operation. This is because only the ring properties of these numbers are required for expressing linear operations: we place ourselves upon a 'module' rather that a full vectorial space.
Convention. Even though computational scalars were themselves implemented as a vector space, from now on we treat them and denote them simply and solely like scalars, e.g. $\lambda.\mathbf{u} + \mu.\mathbf{v}$ will refer to the superposition of vectors $\mathbf{u}$ and $\mathbf{v}$, with λ and μ in $\tilde{\mathbb{K}}$.

4 Matching construct

4.1 Notations

Your typical functional language (`Haskell`, `ML`...) will always have "matching" constructs (for branching). Here is a piece of `Caml`:

$$
\begin{aligned}
\mathbf{1} * \mathbf{v} &\longrightarrow \mathbf{v} \\
\frac{\mathbf{1}}{\sqrt{\mathbf{2}}} * \frac{\mathbf{1}}{\sqrt{\mathbf{2}}} &\longrightarrow \mathrm{fl}(\mathrm{pos}, 1, S(\mathrm{zeron})).\mathbf{1} \\
\frac{\mathbf{1}}{\sqrt{\mathbf{2}}} * \mathbf{i} &\longrightarrow \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} \\
&\vdots \\
\mathbf{i} * \mathbf{i} &\longrightarrow \mathrm{fl}(\mathrm{neg}, 1, \mathrm{zeron}).\mathbf{1} \\
\mathbf{i} * \frac{\mathbf{i}}{\sqrt{\mathbf{2}}} &\longrightarrow \mathrm{fl}(\mathrm{neg}, 1, \mathrm{zeron}).\frac{\mathbf{1}}{\sqrt{\mathbf{2}}} \\
&\vdots \\
(\lambda.\mathbf{u}) * \mathbf{v} &\longrightarrow \lambda.(\mathbf{u} * \mathbf{v}) \\
(\mathbf{t} + \mathbf{u}) * \mathbf{v} &\longrightarrow \mathbf{t} * \mathbf{v} + \mathbf{u} * \mathbf{v}
\end{aligned}
$$

Figure 6: SCALAR MULTIPLICATION

```
let rec not b = match b with
| false -> true
| true -> false ;;
```

We wish to provide such constructs in our linear-algebraic calculus, but mathematicians and physicist in this field would normally write linear maps instead: NOT $= |\text{true}\rangle\langle\text{false}| + |\text{false}\rangle\langle\text{true}|$. However here the $\langle\text{false}|$ and $\langle\text{true}|$ may be viewed as patterns, waiting to be compared to the input vector through a scalar product. Thus we choose to reconcile both worlds and write:

$$\text{NOT} = \mathbf{false} \rhd \mathbf{true} + \mathbf{true} \rhd \mathbf{false}.$$

An expression $(\mathbf{t} \rhd \mathbf{u})$ applied to a vector $\mathbf{v}$ will then reduce into $(\mathbf{t} \bullet v).\mathbf{u}$, with $\bullet$ the scalar product. In this sense $(\mathbf{t} \rhd \mathbf{u}) * \mathbf{v}$ does return $\mathbf{u}$ *in so far as* $\mathbf{t}$ overlaps with $\mathbf{v}$. More formal justifications, and formal rewrite rules follow in the next two subsections. For now we give the reduction steps involved in the application of the phase gate P upon the vector **true**, as a motivating example for these rules:

$$
\begin{aligned}
&\left((\mathbf{false} \rhd \mathbf{false}) + \mathbf{true} \rhd (\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}).\mathbf{true}\right) * \mathbf{true} \\
\longrightarrow^* \ &(\mathbf{false} \rhd \mathbf{false}) * \mathbf{true} + \big(\mathbf{true} \rhd (\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}).\mathbf{true}\big) * \mathbf{true} \\
\longrightarrow^* \ &(\mathbf{false} \bullet \mathbf{true}).\mathbf{false} + (\mathbf{true} \bullet \mathbf{true}).\big((\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}).\mathbf{true}\big) \\
\longrightarrow^* \ &0.\mathbf{false} + (\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}).\mathbf{true} \\
\longrightarrow^* \ &(\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}).\mathbf{true}.
\end{aligned}
$$

All of the three gates forming a universal set for quantum computation are trivially expressed as terms in this notation:

$$\begin{aligned} CNOT = &(\mathbf{false} \otimes \mathbf{false}) \rhd (\mathbf{false} \otimes \mathbf{false}) \\ &+ (\mathbf{false} \otimes \mathbf{true}) \rhd (\mathbf{false} \otimes \mathbf{true}) \\ &+ (\mathbf{true} \otimes \mathbf{false}) \rhd (\mathbf{true} \otimes \mathbf{true}) \\ &+ (\mathbf{true} \otimes \mathbf{true}) \rhd (\mathbf{true} \otimes \mathbf{false}) \\ H = &\left(\mathbf{false} \rhd \frac{1}{\sqrt{2}}.(\mathbf{false} + \mathbf{true})\right) + \left(\mathbf{true} \rhd \frac{1}{\sqrt{2}}.(\mathbf{false} - \mathbf{true})\right) \\ P = &(\mathbf{false} \rhd \mathbf{false}) + \left(\mathbf{true} \rhd (\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}).\mathbf{true}\right). \end{aligned}$$

4.2 Higher-order programming and quantum operations as quantum states

Higher-order programming languages allow functions to be passed as arguments, and to undergo processing just as any other data. Informally such languages are often said to place functions as "first class citizens", i.e. on an equal footing with data. The paradigm is pushed to some extreme in the untyped λ-calculus, where one can no longer differentiate functions from data - for instance the boolean **true** gets represented by $\lambda x.(\lambda y.x)$, which is also the function that discards its second argument whilst returning the first.

There are several well-established reasons to favour higher-order programming: as a way to define useful higher-order functions (e.g. `Map f [1 2 3]` to mean `[f(1) f(2) f(3)]`); in order to obtain a type system akin to propositional logic (e.g. with `Map` of type `(a -> b) -> (list(a) -> list(b))` and `f` of type `(a -> b)`, `Map f` is of type `(list(a) -> list(b))`); so as to implement control flow as a particular case of data flow (e.g. $(\lambda x.(x * x))(\lambda x.(x * x))$ will recurse for ever, where * denotes application).

Suppose we want a higher-order programming language for quantum computation. We therefore want a scheme whereby a quantum operation $\widehat{\$}$ can act upon another quantum operation $\widehat{\rho}$, *just as though it was a state*[1]. As a consequence we need an encoding of quantum operations $\widehat{\rho}$ as quantum state ρ. We will then let $\widehat{\$}(\widehat{\rho})$ stand for $\widehat{\left(\widehat{\$}(\rho)\right)}$.

There exists at least one such encoding in the quantum information literature, which dates back to the work of Jamiolkowski [15] and Choi [8], reviewed and taken further in [3]. Although the isomorphism has several deep consequences on the mathematics of the density matrix formalism of quantum theory, it remains straightforward upon pure states, as it relates vectors of $\mathbb{C}^m \otimes \mathbb{C}^n$ to endomorphisms from $\mathbb{C}^n$ to $\mathbb{C}^m$.

[1] There are no formal definitions for these notions, but here we usually mean something stronger than just the composition $\left(\widehat{\$} \circ \widehat{\rho}\right)$. For instance the higher-order function `Map` cannot be implement through composition mechanisms alone.

$$
\begin{aligned}
(\mathbf{t}+\mathbf{u}) \rhd \mathbf{v} &\longrightarrow \mathbf{t} \rhd \mathbf{v} + \mathbf{u} \rhd \mathbf{v} \\
\mathbf{t} \rhd (\mathbf{v}+\mathbf{w}) &\longrightarrow \mathbf{t} \rhd \mathbf{v} + \mathbf{t} \rhd \mathbf{w} \\
(\lambda.\mathbf{u}) \rhd \mathbf{v} &\longrightarrow \overline{\lambda}.(\mathbf{u} \rhd \mathbf{v}) \\
\mathbf{u} \rhd (\mu.\mathbf{v}) &\longrightarrow \mu.(\mathbf{u} \rhd \mathbf{v}) \\
\mathbf{0} \rhd \mathbf{u} &\longrightarrow \mathbf{0} \\
\mathbf{u} \rhd \mathbf{0} &\longrightarrow \mathbf{0}
\end{aligned}
$$

Figure 7: MATCHING OPERATOR BILINEARITY

Isomorphism 1 *The following linear map*

$$
\begin{aligned}
\hat{} : \mathbb{C}^m \otimes \mathbb{C}^n &\to End(\mathbb{C}^n \to \mathbb{C}^m) \\
A &\mapsto \hat{A} \\
\sum_{ij} A_{ij}|i\rangle|j\rangle &\mapsto \sum_{ij} A_{ij}|i\rangle\langle j|
\end{aligned}
$$

where $i = 1, \ldots, m$ and $j = 1, \ldots, n$, is an isomorphism taking mn vectors A into $m \times n$ matrices $\hat{A}$.

Until Section 6 we do not worry about normalization and unitarity conditions. For now the isomorphism provides exactly what is needed: linear operations can be encoded as sums of tensors describing which vector is associated to which. This is the rationale behind the $\rhd$ notation.

4.3 Rules

Since $\rhd$ is just another type of tensor product, bilinearity applies (see Figure 7. Notice the conjugation of the λ scalar, denoted $\overline{\lambda}$, easily implemented in the TRS). Other than its left-hand-side antilinearity, the particularity of $\rhd$ is the reduction it induces when placed left of an application symbol *, as described in Figure 8. Finally note that the application needs itself be made bilinear, i.e. we need six rules as in Figure 4 - replacing $\otimes$ by *.

5 Lambda calculus construct

5.1 Physical considerations

Whatever formalism we choose for quantum theory (vectors or density matrices), quantum operations act linearly upon their input states. This, in turn, implies that quantum states

$$
\begin{aligned}
(t \rhd u) * v &\longrightarrow (t \bullet v).u \\
(t \otimes u) \bullet (v \otimes w) &\longrightarrow (t \bullet v) \times (u \bullet w) \\
(t \otimes u) \bullet (v \rhd w) &\longrightarrow 0 \\
(t \otimes u) \bullet \mathbf{true} &\longrightarrow 0 \\
(t \otimes u) \bullet \mathbf{false} &\longrightarrow 0 \\
(t \otimes u) \bullet \mathbf{0} &\longrightarrow 0 \\
(t \rhd u) \bullet (v \otimes w) &\longrightarrow 0 \\
(t \rhd u) \bullet (v \rhd w) &\longrightarrow (t \bullet v) \times (u \bullet w) \\
&\vdots \\
\mathbf{true} \bullet (v \otimes w) &\longrightarrow 0 \\
\mathbf{true} \bullet (v \rhd w) &\longrightarrow 0 \\
\mathbf{true} \bullet \mathbf{true} &\longrightarrow 1 \\
\mathbf{true} \bullet \mathbf{false} &\longrightarrow 0 \\
&\vdots \\
\mathbf{0} \bullet \mathbf{v} &\longrightarrow 0
\end{aligned}
$$

Figure 8: MATCHING OPERATOR AND THE SCALAR PRODUCT

cannot be *cloned*. Indeed such an evolution acts upon a qubit as follows:

$$
\begin{gathered}
(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \overset{\text{CLONE}}{\longmapsto} (\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle) \\
\begin{pmatrix} \alpha \\ 0 \\ \beta \\ 0 \end{pmatrix} \overset{\text{CLONE}}{\longmapsto} \begin{pmatrix} \alpha^2 \\ \alpha\beta \\ \alpha\beta \\ \beta^2 \end{pmatrix},
\end{gathered}
$$

which cannot be linear (a more formal discussion can be found in [27]). Cloning should be distinguished from *copying* however, as we now illustrate once more on qubit:

$$
\begin{array}{rl}
|0\rangle \otimes |0\rangle & \overset{\text{COPY}}{\longmapsto} |0\rangle \otimes |0\rangle \\
|1\rangle \otimes |0\rangle & \overset{\text{COPY}}{\longmapsto} |1\rangle \otimes |1\rangle \\
(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle & \overset{\text{COPY}}{\longmapsto} \alpha|0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle
\end{array}
\qquad
\begin{pmatrix} \alpha \\ 0 \\ \beta \\ 0 \end{pmatrix} \overset{\text{COPY}}{\longmapsto} \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \beta \end{pmatrix}.
$$

Such an evolution is perfectly valid, and in the above case it may be implemented as a single application of the quantum gate $CNOT$, which is of course both linear and unitary. In classical functional languages terms such as $\lambda x.f(x,x)$, with $(\lambda x.f(x,x)\ \ t) \xrightarrow{\beta} f(t,t)$, are crucial for the expressiveness. Recursion, for instance, relies upon such terms, and is absolutely necessary for universality. When designing a quantum functional language we therefore face a choice:

- Either we prevent terms such as $\lambda x.f(x,x)$ from being applied to quantum states - thereby ensuring that no quantum cloning is allowed. But we must authorize their

$$
\begin{array}{l}
L(\mathbf{u}) * \mathbf{v} \longrightarrow (\mathbf{u} \text{ of } \mathbf{v}) \\
\mathbf{t} \text{ of } (r.\mathbf{v}) \longrightarrow r.(\mathbf{t} \text{ of } \mathbf{v}) \\
\mathbf{t} \text{ of } (\mathbf{v} + \mathbf{w}) \longrightarrow (\mathbf{t} \text{ of } \mathbf{v}) + (\mathbf{t} \text{ of } \mathbf{w}) \\
\mathbf{t} \text{ of } \mathbf{true} \longrightarrow \mathbf{t} \text{ bof subst}(\mathbf{true}) \\
\mathbf{t} \text{ of } \mathbf{false} \longrightarrow \mathbf{t} \text{ bof subst}(\mathbf{false}) \\
\mathbf{t} \text{ of } \mathbf{0} \longrightarrow \mathbf{t} \text{ bof subst}(\mathbf{0}) \\
\mathbf{t} \text{ of } (\mathbf{v} \otimes \mathbf{w}) \longrightarrow \\
\Big(\Big(((\mathbf{t} \text{ bof } \Uparrow(\uparrow)) \text{ bof } \Uparrow(\uparrow)) \text{ bof subst}\big(\mathbf{var}(0) \otimes \mathbf{var}(S(0))\big)\Big) \text{ of } \mathbf{v}\Big) \text{ of } \mathbf{w} \\
\mathbf{t} \text{ of } (\mathbf{v} \triangleright \mathbf{w}) \longrightarrow \\
\Big(\Big(((\mathbf{t} \text{ bof } \Uparrow(\uparrow)) \text{ bof } \Uparrow(\uparrow)) \text{ bof subst}\big(\mathbf{var}(0) \triangleright \mathbf{var}(S(0))\big)\Big) \text{ of } \mathbf{v}\Big) \text{ of } \mathbf{w}
\end{array}
$$

Figure 9: ENFORCING LINEARITY OVER SUBSTITUTION

applications upon 'classical terms' for expressiveness. As a consequence the language must be able to keep track of quantum resources versus classical resources. This is the approach followed by Van Tonder [24].

- Or we allow terms such as $\lambda x.f(x,x)$ from being applied to quantum states - only to be interpreted as a quantum copy. We may still want to keep track of quantum resources versus classical resources, but not for the purpose of forbidding cloning. This is the approach we take.

For now the latter option seems preferable, since it models the *dos* and *don'ts* of the linearity requirement more closely, whilst keeping the calculus to a minimum. Moreover copying, as we now show, can be imposed over cloning by the semantics of the calculus alone.

5.2 Substitution of de Bruijn indices

The point of the previous discussion is that we can only duplicate basis vectors. Informally

$$
\begin{array}{rl}
(\lambda x.(x \otimes x)) * \mathbf{true} \longrightarrow^* \mathbf{true} \otimes \mathbf{true} & \text{is OK;} \\
(\lambda x.(x \otimes x)) * (\mathbf{false} + \mathbf{true}) \longrightarrow^* \big((\lambda x.(x \otimes x)) * \mathbf{false}\big) + \big((\lambda x.(x \otimes x)) * \mathbf{true}\big) & \\
\longrightarrow^* \big(\mathbf{true} \otimes \mathbf{true}\big) + \big(\mathbf{false} \otimes \mathbf{false}\big) & \text{is OK;} \\
(\lambda x.(x \otimes x)) * (\mathbf{false} + \mathbf{true}) \longrightarrow^* (\mathbf{false} + \mathbf{true}) \otimes (\mathbf{false} + \mathbf{true}) & \text{is not OK.}
\end{array}
$$

Again another way to grasp this idea is to realize that faced with a term of the form $(\lambda x.t) * (u + v)$, one could either start by proceeding to the substitution, or start by applying the right-hand-side linearity of *, leading to two different results. So that operations remain linear, we must favour the right-hand-side linearity of * over substitution. The rules of Figure 9 accomplish exactly that. The three first rules are the most straightforward, they invoke the linearity of the vector to be substituted. The three following rules treat the base

$$
\begin{aligned}
(r.\mathbf{u})\ \text{bof}\ \mathbf{s} &\longrightarrow (r\ \text{bof}\ \mathbf{s}).(\mathbf{u}\ \text{bof}\ \mathbf{s})\\
(\mathbf{t}\bullet\mathbf{u})\ \text{bof}\ \mathbf{s} &\longrightarrow (\mathbf{t}\ \text{bof}\ \mathbf{s})\bullet(\mathbf{u}\ \text{bof}\ \mathbf{s})\\
(r\times s)\ \text{bof}\ \mathbf{s} &\longrightarrow (r\ \text{bof}\ \mathbf{s})\times(s\ \text{bof}\ \mathbf{s})\\
&\vdots\\
(\mathbf{t}+\mathbf{u})\ \text{bof}\ \mathbf{s} &\longrightarrow (\mathbf{t}\ \text{bof}\ \mathbf{s})+(\mathbf{u}\ \text{bof}\ \mathbf{s})\\
(\mathbf{t}\otimes\mathbf{u})\ \text{bof}\ \mathbf{s} &\longrightarrow (\mathbf{t}\ \text{bof}\ \mathbf{s})\otimes(\mathbf{u}\ \text{bof}\ \mathbf{s})\\
(\mathbf{t}\rhd\mathbf{u})\ \text{bof}\ \mathbf{s} &\longrightarrow (\mathbf{u}\ \text{bof}\ \mathbf{s})\rhd(\mathbf{v}\ \text{bof}\ \mathbf{s})\\
(\mathbf{t}*\mathbf{u})\ \text{bof}\ \mathbf{s} &\longrightarrow (\mathbf{t}\ \text{bof}\ \mathbf{s})*(\mathbf{u}\ \text{bof}\ \mathbf{s})\\
L(\mathbf{t})\ \text{bof}\ \mathbf{s} &\longrightarrow L(\mathbf{t}\ \text{bof}\ \Uparrow(\mathbf{s}))\\
\textbf{true}\ \text{bof}\ \mathbf{s} &\longrightarrow \textbf{true}\\
&\vdots\\
\textbf{var}(S(p))\ \text{bof subst}(\mathbf{v}) &\longrightarrow \textbf{var}(p)\\
\textbf{var}(0)\ \text{bof subst}(\mathbf{v}) &\longrightarrow \mathbf{v}\\
\textbf{var}(0)\ \text{bof}\ \Uparrow(\mathbf{s}) &\longrightarrow \textbf{var}(0)\\
\textbf{var}(S(p))\ \text{bof}\ \Uparrow(\mathbf{s}) &\longrightarrow (\textbf{var}(p)\ \text{bof}\ \mathbf{s})\ \text{bof}\ \uparrow\\
\textbf{var}(p)\ \text{bof}\ \uparrow &\longrightarrow \textbf{var}(S(p))
\end{aligned}
$$

Figure 10: EXPLICIT SUBSTITUTION OF DE BRUIJN INDICES

cases, when the vector to be substituted is down to a basic state. The last two rules handle the more subtle case of tensor states $\mathbf{u}\otimes\mathbf{v}$ or $\mathbf{u}\rhd\mathbf{v}$. In a word the trick is to treat

$$\big(\lambda x.(\ldots x\ldots)\big)*(\mathbf{u}\otimes\mathbf{v})\quad\text{as}\quad\Big(\lambda x.\big(\lambda y.(\ldots x\otimes y\ldots)\big)*\mathbf{u}\Big)*\mathbf{v}$$

with y a fresh variable, and then proceed recursively.
Once the vector to be substituted is a basic state, we can safely proceed to the substitution. Here we have chosen to represent variables by their de Bruijn indices [1, 10, 17], i.e. each variable is now an integer number corresponding to number of binders ('L' or 'λ' symbols) one must go through before reaching the binding occurrence. For instance

$$\lambda x.(\lambda y.(x\otimes y))\ \text{is encoded as}\ L(L(\textbf{var}(1)\otimes\textbf{var}(0)))$$

since there is one λ symbol lying between x and the binding occurrence of x. This variable numbering scheme is often used for implementing functional languages. Notice how in this scheme a variable may be denoted differently depending upon its position in the term (i.e. depending upon how far it lies from its biding occurrence). The rules of Figure 10 implement this mechanism.

6 Discussion

6.1 Unitarity

In its simplest formulation quantum theory only allows unitary evolutions, i.e. vectors evolve in time according to square matrices U verifying $U^\dagger U = \mathbb{I}$. In this framework it is impossible to delete, say, a qubit:

$$|0\rangle \stackrel{\text{ERASE}}{\longmapsto} |0\rangle$$
$$|1\rangle \stackrel{\text{ERASE}}{\longmapsto} |0\rangle$$
$$(\alpha|0\rangle + \beta|1\rangle) \stackrel{\text{ERASE}}{\longmapsto} |0\rangle,$$

the evolution is not injective and therefore not unitary. Von Neumann's projective measurements help us only partially: if the vector is measured in the canonical basis, and when this measurement yields outcome '0', then the qubit undergoes the above exact dynamics. But this will only occur with probability $|\alpha|^2$.
The ERASE operation is however perfectly physical, as one can always ignore a qubit and focus upon another, taken to be in state $|0\rangle$. Moreover the process needs not be probabilistic. There are two well-established formulations of quantum theory which cater for this possibility:

- The *generalized measurement* formalism unifies quantum evolutions and quantum measurements as one single object. Mathematically a generalized measurement is given by a set of matrices $\{M_m\}$ verifying

$$\sum_m M_m^\dagger M_m \ll Id. \tag{2}$$

  A vector v will then evolve in time according to the matrix M_m with probability $p_m = |M_m v|^2$ - in which case we shall say that outcome 'm' has occurred. As an example the following generalized measurement performs the ERASE operation:

$$\left\{ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right\}.$$

  For a more detailed presentation of these concepts the reader is referred to [19], page 84.

- The *density matrix* formalism represents quantum states as positive matrices instead of vectors. These evolve in time according to Completely Positive-preserving maps, i.e. operations of the form

$$\rho \longmapsto \sum_m M_m \rho M_m^\dagger \quad \text{with probability} \quad p_m = \text{Tr}(M_m \rho M_m^\dagger)$$

  and verifying Equation (2). This framework is traditional when dealing with open quantum systems, and therefore well appropriate as one discards a qubit.

In classical functional languages terms such as $\lambda y.(\lambda x.x)$, with $(\lambda y.(\lambda x.x)\mathbf{t}) \xrightarrow{\beta} \lambda x.x$, are commonly used. Boolean values and branching, for instance, are encoded in such manners. Although convenient, non-injective functions are not absolutely necessary, for reversible computation can be both universal and efficient [5]. Whether a quantum functional language should allow erasure or not must therefore depend upon which of the three above mentioned formulation of quantum theory gets chosen.

If we adopt the simplest formulation of quantum theory, our language must be restricted to operation which are a not only linear but also unitary. At first sight this seems feasible by imposing the standard $U^{\dagger}U = \mathbb{I}$ condition upon the matching constructs of Section 4, and the *relevance* condition upon the λ-terms of 5 (i.e. for all term $\lambda x.t$ the variable x must occur at least once in t). This, together with the search for a state-operator correspondence similar to Isomorphism 1, but yielding normalized states, remains a subject for future work.

6.2 Linearity?

Linear logic appears in [13] as a mean to express and prove properties of dynamical systems where the *consumption* of resources is important. The standard example (price updated) is $A \equiv$"I have 6€", $B \equiv$"I have a paquet of Gauloises", and the statement $A \multimap B$ to express the possibility of using up A to obtain B. With $\otimes$ now expressing a conjunction, it is clear one cannot have $A \multimap (B \otimes B)$, since this would mean buying two paquets for the price of one. Neither can we have $A \otimes A \multimap B$: we must get something for our money - at worse the feeling of getting cheated. Therefore the rules governing symbols $\multimap, \otimes$ differ from those of classical logic for $\Rightarrow, \wedge$. Unless there is an abundance of resources (denoted by the exclamation mark '!'), in which is case they coincide again: $!A \multimap (B \otimes B \otimes \ldots)$. Whilst considering this point the father of Linear logic has the following thought [14]: '*Classical logic appears to be the logic of macro-actions, as opposed to linear logic which would be a logic of micro-actions. The unusual character of linear logic may therefore be considered similar to the strange character of micro-mechanics, i.e. quantum mechanics.*'.

Specifications expressed in linear logics can be seen as types for programs expressed in linear λ-calculus. In the linear λ-calculus one distinguishes *linear resources*, which may not be copied nor discarded, from *nonlinear resources*, which are denoted by the exclamation mark '!' and whose fate is not subjected to particular restrictions. Van Tonder's quantum λ-calculus (λ_q) is founded upon these ideas. As we have mentioned in 5.1, he uses this well-established framework in order to distinguish quantum resources (treated as linear) from classical resources (treated as nonlinear):

(λ_q) $\quad t ::= x \mid \lambda x.t \mid (t\ \ t) \mid c \mid !t \mid \lambda !x.t$

$\quad\quad\quad\ \ c ::= 0 \mid 1 \mid H \mid CNOT \mid P$

(together with the well-formedness rules of the classical linear λ calculus)

$(\mathcal{R}_q)$ $\quad (\lambda x.t\ \ s) \xrightarrow{\beta} t[s/x]$

$\quad\quad\quad\ \ H\ \ 0 \longrightarrow 0+1$

$\quad\quad\quad\ \ H\ \ 1 \longrightarrow 0-1$

$\quad\quad\quad\ \ \vdots$

Here the well-formedness conditions of the classical linear calculus (which prevents linear terms from being discarded), together with !-suspension (which stops quantum terms from being treated as nonlinear) maintain unitarity throughout the reductions.

The connection between the linear λ-calculus and quantum functional languages is striking. It comes at a price however: the λ_q-calculus remains heterogeneous, i.e. a juxtaposition of quantum resources (linear resources) and classical resources (nonlinear resources). In some sense it is twice linear, both in the sense of linear λ-calculus and linear algebra, and thus perhaps overrestricted. In particular it forbids both the cloning of quantum data (which needs be done) and the copying of quantum data (which needs not be done). As a consequence its control flow remains inherently based upon classical resources.

The *linear-algebraic λ-calculus* constructed in this paper is homogeneous, i.e. it does not draw a line between quantum resources and classical resources (the latter are merely thought of as basis states of the former). Moreover it exhibits only one notion of linearity, which is that of linear algebra. Thus cloning remains disallowed (just by the semantics) but not copy. Control flow is still provided as a consequence. These results seem to open the way to a linear algebraic interpretation of linear logic, in the spirit of Girard's Geometry of interaction, although much work remains ahead in order to strengthen this connection.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, *Explicit Substitutions*, J. Funct. Program. **1**(4), 375-416, (1991).

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

[3] P. Arrighi, C. Patricot, *On Quantum Operations as Quantum States*, to appear in Ann. of Phys., arXiv:quant-ph/0307024.

[4] P. Arrighi, G. Dowek, *Some remarks on the definition of the notion of vectorial space* , WRLA 2004, Barcelona, March 2004.

[5] C. Bennett, *Logical reversibility of computation*, IBM J. Res. Develop., **17**, 525532, (1973).

[6] E. Bernstein, U. Vazirani, *Quantum complexity theory*, Proc. of the 25th STOCS, 11-20, ACM Press, New York (1993). http://citeseer.nj.nec.com/bernstein97quantum.html

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

[8] M.D. Choi, *Completely Positive linear maps on complex matrices*, Lin. Alg. Appl., **10**, 285-290, (1975).

[9] D. Cohen, P. Watson, *An efficient representation of arithmetic for term rewriting*, Proc. of the 4th Conference on Rewrite Techniques and Applications, LNCS **488**, 240-251, (1991).

[10] P.-L. Curien , T. Hardin , J.-J. Lévy, *Confluence properties of weak and strong calculi of explicit substitutions*, J. of the ACM, **43**(2), 362-397, (1996).

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

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

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

[14] J.-Y. Girard in *Logique et informatique : une introduction*, Ecole de printemps d'Informatique thorique, Albi, Eds. B. Courcelle *et al.*, INRIA, (1991).

[15] A. Jamiolkowski, *Linear transformations which preserve trace and positive semidefinite operators*, Rep. Mod. Phys., **3**, 275-278, (1972).

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

[17] P. Lescanne, *From lambda-sigma to lambda-upsilon, a journey through calculi of explicit substitutions* 21st ACM Symposium on Principles of Programming Languages (POPL), 60-69, (1994).

[18] P. Maymin, *Extending the lambda calculus to express randomized and quantumized algorithms*, quant-ph/9702057, (1996).

[19] M.A. Nielsen, I.L. Chuang, *Quantum computation and quantum information*, Cambridge University Press, (2000).

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

[21] P. Selinger, *Towards a quantum programming language*, to appear in Math. Struc. in Comp. Sci., (2003).

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

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

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

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

[26] F. Verstraete, H. Verschelde, *On quantum Channels.*, Internal Report 02-176, ESAT-SISTA, K.U.Leuven, (2002).

[27] W. Wootters, W. Zurek, *A single quantum cannot be cloned*, Nature, 299, 802-803, (1982).

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

[29] A. Yao, *Quantum circuit complexity*, Proc. of the 34th FOCS, 352-361, (1993).