

Wave-Style Token Machines and Quantum Lambda Calculi

Ugo Dal Lago¹ and Margherita Zorzi²

¹ Università di Bologna & INRIA, Italy

dallago@cs.unibo.it

² Università di Verona, Italy.

margherita.zorzi@univr.it

Abstract

Particle-style token machines are a way to interpret proofs and programs, when the latter are written following the principles of linear logic. In this paper, we show that token machines also make sense when the programs at hand are those of a simple quantum λ -calculus with implicit qubits. This, however, requires generalizing the concept of a token machine to one in which more than one particle travel around the term *at the same time*. The presence of multiple tokens is intimately related to entanglement and allows to give a simple operational semantics to the calculus, coherently with the principles of quantum computation.

1 Introduction

One of the strongest trends in computer science is the (relatively recent) interest in exploiting new computing paradigms which go beyond the usual, classical one. Among these paradigms, quantum computing plays an important role. In particular, the quantum paradigm is having a deep impact on the notion of a computationally (in)tractable problem [15].

Even if quantum computing has catalyzed the interest of a quite large scientific community, several theoretical aspects are still unexplored. As an example, the definition of a robust theoretical framework for quantum programming is nowadays still a challenge. A number of (paradigmatic) calculi for quantum computing have been introduced in the last ten years. Among them, some functional calculi, typed and untyped, have been proposed [4, 5, 6, 13, 16, 18], but we are still at a stage where it is not clear whether one calculus could be considered *canonical*. Since quantum data have to undergo restrictions such as no-cloning and no-erasing, it is not surprising that in most of the cited quantum calculi the use of resources is controlled. Linear logic therefore provides an ideal framework where rooting quantum data treatment, but also offers another tool which has not been widely exploited in the quantum setting: its mathematical model in terms of operator algebras, i.e. the Geometry of Interaction (GoI in the following). Indeed, the latter provides a dynamical interpretation and a semantic account of the cut-elimination procedure as a flow of information circulating into a net structure. This idea can be formulated both as an algebra of bounded operators on a infinitely dimensional Hilbert space [10] or as a token-based machine. On the one hand, the Hilbert space on top of which the first formulation of GoI is given is precisely the canonical state space of a quantum Turing machine (see for example [1]). On the other hand, the definition of a token machine provides a mathematically simpler setting, which has already found a role in this context [3, 11].

In this paper, we show that token machines are also a model of a linear quantum λ -calculus with implicit qubits, called Q Λ and defined along the lines of van Tonder's λ_q [16]. This allows to give an operational semantics to Q Λ which renders the quantum nature of Q Λ explicit: *type derivations become quantum circuits built on exactly the set of gates occurring in the underlying λ -term*. This frees us from the burden of having to define the operational semantics of quantum calculi in reduction style, which

is known to be technically challenging in a similar setting [16]. On the other hand, the power of β -style axioms is retained in the form of an equational theory for which our operational semantics can be proved sound. Technically, the design of our token machine for $Q\Lambda$, called $IAM_{Q\Lambda}$, is arguably more challenging than the one of classical token machines. Indeed, the principles of quantum computing, and the so-called *entanglement* in particular, force us to go towards *wave-style* machines, i.e., to machines where more than one particle can travel inside the program at the same time. Moreover, the possibly many tokens at hand are subject to synchronization points, each one corresponding to unitary operators of arity greater than 1. This means that $IAM_{Q\Lambda}$, in principle, could suffer from deadlocks, let alone the possibility of non-termination. We here prove that these pathological situations can *never* happen. In the present paper we also establish a soundness theorem: we state and prove that the semantics induced by the token machine $IAM_{Q\Lambda}$ is sound with respect to $Q\Lambda$'s equational theory, i.e. it is invariant with respect to term equivalence. The proof, which can be found in [7], is not trivial, since our notion of type derivation has to deal with quantum superposition [12] and is thus non-standard. Finally, it is mandatory to recall that, even if the possibility of observing quantum data is an useful and expressive programming tool, considering a measurement-free calculus is a theoretically well-founded choice, since measurements can always been postponed [12]. Thus, this is not a limitation when one addresses computability issues.

The calculus $Q\Lambda$ and its token machine $IAM_{Q\Lambda}$ are introduced in Section 2 and Section 3, respectively. Main results about $IAM_{Q\Lambda}$ are in Section 4. An extended version of this paper with more details, proofs and a gentle introduction to quantum computing is available [7].

2 The Calculus $Q\Lambda$

An essential property of quantum programs is that quantum data, i.e. quantum bits, should always be uniquely referenced. This restriction follows from the well-known *no-cloning* and *no-erasing* properties of quantum physics, which state that a quantum bit cannot in general be duplicated nor canceled [12]. Syntactically, one captures this restriction by means of linearity: if every abstraction $\lambda x.M$ is such that there is *exactly one* free occurrence of x in M , then the substitution triggered by firing *any* redex is neither copying nor canceling and, as a consequence, coherent with the just stated principles. In this Section, we introduce a quantum linear λ -calculus in the style of van Tonder's λ_q [16] and give an equational theory for it.

The Language of Terms. Let us fix a finite set $\mathcal{U}$ of *unitary operators*, each on a finite-dimensional Hilbert space $\mathbb{C}^{2^n}$, where n can be arbitrary. To each such $U \in \mathcal{U}$ we associate a symbol U and call n the *arity* of U . The syntactic categories of *patterns*, *bits*, *constants* and *terms* are defined by the following grammar:

$$\begin{array}{lll}
 \pi & ::= & x \mid \langle x, y \rangle; & \text{patterns} \\
 B & ::= & |0\rangle_n \mid |1\rangle_n; & \text{bits} \\
 C & ::= & B \mid U; & \text{constants} \\
 M, N & ::= & x \mid C \mid M \otimes N \mid MN \mid \lambda \pi.M; & \text{terms}
 \end{array}$$

where n ranges over $\mathbb{N}$ and x ranges over a denumerable, totally ordered set of variables $\mathbb{V}$. We always assume that the natural numbers occurring next to bits in any term M are pairwise distinct. This condition, by the way, is preserved by substitution when the substituted variable occurs (free) exactly once. Whenever this does not cause ambiguity, we elide labels and simply write $|b\rangle$ for a bit. Notice that pairs can be formed via the binary operator $\otimes$. In the following, capital letters such as M, N, L, Q (possibly indexed), denote terms. We work modulo variable renaming; in other words, terms are equivalence

$$\boxed{
\begin{array}{c}
\frac{}{x : A \vdash x : A} (a_v) \quad \frac{}{\cdot \vdash |0\rangle : \mathbb{B}} (a_{q0}) \quad \frac{}{\cdot \vdash |1\rangle : \mathbb{B}} (a_{q1}) \quad \frac{}{\cdot \vdash U : \mathbb{B}^n \multimap \mathbb{B}^n} (a_U) \\
\\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \multimap B} (l_{\multimap}^1) \quad \frac{\Gamma, x : A, y : B \vdash M : C}{\Gamma \vdash \lambda \langle x, y \rangle. M : (A \otimes B) \multimap C} (l_{\multimap}^2) \\
\\
\frac{\Gamma \vdash M : A \multimap B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B} (E_{\multimap}) \quad \frac{\Gamma \vdash M : A \quad \Delta \vdash N : B}{\Gamma, \Delta \vdash M \otimes N : A \otimes B} (l_{\otimes})
\end{array}
}$$

Figure 1: Typing Rules.

classes modulo α -conversion. Substitution up to α -equivalence is defined in the usual way. Observe that the terms of Q Λ are the ones of a λ -calculus with pairs (which are accessed by pattern-matching) endowed with constants for bits and unitary operators. We don't consider measurements here, and discuss the possibility of extending the language of terms in Section 6.

Judgements and Typing Rules. Since in Q Λ all terms are assumed to be non-duplicable by default, we adopt a linear type-discipline. Formally, the set of types is defined as

$$A ::= \mathbb{B} \mid A \multimap B \mid A \otimes B,$$

where $\mathbb{B}$ is the ground type of qubits. We write $\mathbb{B}^n$ for the n -fold tensor product $\mathbb{B} \otimes \dots \otimes \mathbb{B}$ (n times). A *judgement* is an expression $\Gamma \vdash M : A$, where Γ is a linear environment, M is a term, and A is a type in Q Λ . *Typing rules* are in Figure 1. Observe that contexts are treated multiplicatively and, as a consequence, variables always appear exactly once in terms. In other words, a *strictly linear type discipline* is enforced.

Example 1 (EPR States). Consider the term $M_{EPR} = \lambda \langle x, y \rangle. CNOT(Hx \otimes y)$. M_{EPR} encodes the quantum circuit on two input qubits which has the ability to produce an entangled state from any element of the underlying computational basis. It can be given the type $\mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}$ in the empty context. The following is a type derivation π_{EPR} for it, where we mark different occurrences of $\mathbb{B}$ with distinct indexes. (Indexed occurrences will be exploited in Section 3, Example 2.)

$$\frac{
\frac{
\frac{
\frac{
\frac{}{\cdot \vdash H : \mathbb{B}_{21} \multimap \mathbb{B}_{22}}{} \quad \frac{}{x : \mathbb{B}_{23} \vdash x : \mathbb{B}_{24}}{}
}{\frac{}{x : \mathbb{B}_{17} \vdash Hx : \mathbb{B}_{18}}}{} (E_{\multimap})
\quad
\frac{}{y : \mathbb{B}_{19} \vdash y : \mathbb{B}_{20}}{}
}{\frac{}{x : \mathbb{B}_{13}, y : \mathbb{B}_{14} \vdash Hx \otimes y : \mathbb{B}_{15} \otimes \mathbb{B}_{16}}}{} (l_{\otimes})
\quad
\frac{}{\cdot \vdash CNOT : \mathbb{B}_9 \otimes \mathbb{B}_{10} \multimap \mathbb{B}_{11} \otimes \mathbb{B}_{12}}{}
}{\frac{}{x : \mathbb{B}_5, y : \mathbb{B}_6 \vdash CNOT(Hx \otimes y) : \mathbb{B}_7 \otimes \mathbb{B}_8}}{} (E_{\multimap})
\quad
\frac{}{\cdot \vdash M_{EPR} : \mathbb{B}_1 \otimes \mathbb{B}_2 \multimap \mathbb{B}_3 \otimes \mathbb{B}_4}}{} (l_{\multimap}^2)$$

An Equational Theory. The λ -calculus is usually endowed with notions of reduction or equality, both centered around the β -rule. A reduction relation implicitly provides the underlying calculus with a notion of computation, while an equational theory is more akin to a reasoning technique. Giving a reduction relation on Q Λ terms directly, however, is problematic. What happens when a n -ary unitary operator U is faced with an n -tuple of qubits $|b_1 \dots b_n\rangle = |b_q\rangle \otimes \dots \otimes |b_n\rangle$? Superposition should somehow arise, but how can we capture it?

In this section, an equational theory for $Q\Lambda$ will be introduced (see [7] for more details). In the next sections, we will show that the semantics induced by token machines is *sound* with respect to it. The equational theory we are going to introduce will be a binary relation on formal, weighted sums of type derivations for $Q\Lambda$ terms. A *superposed type derivation* of type (Γ, A) is a formal sum $\mathcal{T} = \sum_{i=1}^n \kappa_i \pi_i$, where for every $1 \leq i \leq n$, $\kappa_i \in \mathbb{C}$ and it holds that $\pi_i \triangleright \Gamma \vdash M_i : A$. In this case, we write $\Gamma \vdash \mathcal{T} : A$. Superposed type derivations will be denoted by metavariables like $\mathcal{T}$ or $\mathcal{S}$. If $\pi \triangleright \Gamma \vdash M : A$ and M is a redex (i.e., either $U|b_1 \dots b_k\rangle$ or $(\lambda x.M)N$ or $(\lambda \langle x, y \rangle.M)(N \otimes L)$), one can easily define a (superposed) type derivation $\pi^\Downarrow$ for the term(s) obtained by firing the redex. The binary relation $\approx$ on superposed type derivations is defined as the smallest equivalence relation including all pairs in the form $(\pi, \pi^\Downarrow)$ and closed under all contexts. When the underlying type derivation is clear from the context, we denote superposed derivations simply by superposed *terms*. As an example, consider the term $M_{EPR}(|0\rangle_1 \otimes |1\rangle_2)$ from Example 1 and the corresponding type derivation ρ_{EPR} , easily definable from π_{EPR} , for it (see also [7]). It is convenient to be able to reason as follows, directly on the former:

$$\begin{aligned} M_{EPR}(|0\rangle \otimes |1\rangle) &\approx CNOT(H|0\rangle \otimes |1\rangle) \approx \frac{1}{\sqrt{2}} CNOT(|0\rangle \otimes |1\rangle) + \frac{1}{\sqrt{2}} CNOT(|1\rangle \otimes |1\rangle) \\ &\approx \frac{1}{\sqrt{2}} |0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}} CNOT(|1\rangle \otimes |1\rangle) \approx \frac{1}{\sqrt{2}} |0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}} |1\rangle \otimes |0\rangle. \end{aligned}$$

Please observe that the equational theory we have just defined can *hardly* be seen as an operational semantics for $Q\Lambda$. Although equations can of course be oriented, it is the very nature of a superposed type derivation which is in principle problematic from the point of view of quantum computation: what is the mathematical nature of a superposed type derivation? Is it an element of an Hilbert Space? And if so, of *which one*? If we consider a simple language such as $Q\Lambda$, the questions above may appear overly rhetorical, but we claim they are not. For example, what would be the quantum meaning of linear beta-reduction? If we want to design beta-reduction according to the principles of quantum computation, it has to be, at least, easily reversible (unless measurement is implicit in it). Moving towards more expressive languages, this non-trivial issue become more difficult and a number of constraints have to be imposed (for example, superposition of terms can be allowed, but only between “homogeneous” terms, i.e. terms which have an identical skeleton [16]). This is the reason for which promising calculi [16] fail to be canonical models for quantum programming languages. This issue has been faced in literature without satisfactory answers, yielding a number of convincing arguments in favor of the (implicit or explicit) classical control of quantum data [4, 13].

3 A Token Machine for $Q\Lambda$

In this section we describe an interpretation of $Q\Lambda$ type derivations in terms of a specific token machine called $IAM_{Q\Lambda}$. With a slight abuse of notation, a permutation $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ will be often applied to sequences of length n with the obvious meaning: $\sigma(a_1, \dots, a_n) = a_{\sigma(1)}, \dots, a_{\sigma(n)}$. Similarly, such a permutation can be seen as the *unique* unitary operator on $\mathbb{C}^{2^n}$ which sends $|b_1 \dots b_n\rangle$ to $|b_{\sigma(1)} \dots b_{\sigma(n)}\rangle$. Suppose given an operator $U \in \mathcal{U}$ of arity $n \in \mathbb{N}$. Now, take a natural number $m \geq n$ and n distinct natural numbers $j_1, \dots, j_n$, all of them smaller or equal to m . With $U_m^{j_1, \dots, j_n}$ (or simply with $U^{j_1, \dots, j_n}$) we indicate the operator of arity m which acts like U on the qubits indexed with $j_1, \dots, j_n$ and leave all the other qubits unchanged. In the following, with a slight abuse of notation, occurrences of types in type derivations are confused with types themselves. On the other hand, occurrences of types *inside other types* will be defined quite precisely, as follows. *Contexts* (types with a hole) are denoted by metavariables like C, D . A context C is said to be a *context for a type A* if $C[\mathbb{B}] = A$. *Negative contexts* (i.e., contexts where the hole is in negative position) are denoted by metavariables like N, M . *Positive ones* are denoted by metavariables like P, Q . An *occurrence* of $\mathbb{B}$ in the type derivation

$$\begin{array}{c}
\frac{}{x : A_1 \vdash x : A_2} \quad ((\varphi, (A_1, P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_2, P), \psi), \mathbf{Q}) \\
\quad \quad \quad ((\varphi, (A_2, N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_1, N), \psi), \mathbf{Q}) \\
\\
\frac{\Gamma_1, x : A_1 \vdash M : B_1}{\Gamma_2 \vdash \lambda x. M : A_2 \multimap B_2} \quad \begin{array}{l}
((\varphi, (A_1, N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_2 \multimap B_2, N \multimap B_2), \psi), \mathbf{Q}) \\
((\varphi, (A_2 \multimap B_2, P \multimap B_2), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_1, P), \psi), \mathbf{Q}) \\
((\varphi, (B_1, P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_2 \multimap B_2, A_2 \multimap P), \psi), \mathbf{Q}) \\
((\varphi, (A_2 \multimap B_2, A_2 \multimap N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (B_1, N), \psi), \mathbf{Q}) \\
((\varphi, (\Gamma_2, P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (\Gamma_1, P), \psi), \mathbf{Q}) \\
((\varphi, (\Gamma_1, N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (\Gamma_2, N), \psi), \mathbf{Q})
\end{array} \\
\\
\frac{\Gamma_1 \vdash M : A_1 \multimap B_1 \quad \Delta_1 \vdash N : A_2}{\Gamma_2, \Delta_2 \vdash MN : B_2} \quad \begin{array}{l}
((\varphi, (A_2, P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_1 \multimap B_1, P \multimap B_1), \psi), \mathbf{Q}) \\
((\varphi, (A_1 \multimap B_1, N \multimap B_1), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_2, N), \psi), \mathbf{Q}) \\
((\varphi, (A_1 \multimap B_1, A_1 \multimap P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (B_2, P), \psi), \mathbf{Q}) \\
((\varphi, (B_2, N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (A_1 \multimap B_1, A \multimap N), \psi), \mathbf{Q}) \\
((\varphi, (\Gamma_2, P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (\Gamma_1, P), \psi), \mathbf{Q}) \\
((\varphi, (\Gamma_1, N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (\Gamma_2, N), \psi), \mathbf{Q}) \\
((\varphi, (\Delta_2, P), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (\Delta_1, P), \psi), \mathbf{Q}) \\
((\varphi, (\Delta_1, N), \psi), \mathbf{Q}) \rightarrow_{\pi} ((\varphi, (\Delta_2, N), \psi), \mathbf{Q})
\end{array} \\
\\
\frac{}{\cdot \vdash U : \mathbb{B}_1 \otimes \dots \otimes \mathbb{B}_m \multimap \mathbb{B}_{m+1} \otimes \dots \otimes \mathbb{B}_{2m}} \quad \begin{array}{l}
((\varphi_1, \mathbb{B}_{j_1}, \varphi_2, \dots, \varphi_m, \mathbb{B}_{j_m}, \varphi_{m+1}), \mathbf{Q}) \\
\rightarrow_{\pi} \\
((\varphi_1, \mathbb{B}_{j_1+m}, \varphi_2, \dots, \varphi_m, \mathbb{B}_{j_m+m}, \varphi_{m+1}), \mathbf{U}^{i_1, \dots, i_m}(\mathbf{Q}))
\end{array}
\end{array}$$

Figure 2: IAM_{Q Λ} Transition Rules — Selection

π is a pair (A, C) , where A is an occurrence of a type in π and C is a context for A . Sequences of occurrences are indicated with metavariables like φ, ψ (possibly indexed). All sequences of occurrences we will deal with do not contain duplicates. Type constructors $\multimap$ and $\otimes$ can be generalized to operators on occurrences and sequences of occurrences, e.g. $(A, C) \multimap B$ is just $(A \multimap B, C \multimap B)$.

Given (an occurrence of) a type A , all positive and negative occurrences of $\mathbb{B}$ inside A can be put in sequences called $\mathcal{P}(A)$ and $\mathcal{N}(A)$. As an example, the positive occurrences in the type $\mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}$ should be the two rightmost ones. And, indeed, $\mathcal{P}(\mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}) = (\mathbb{B}, \mathbb{B} \multimap ([\cdot] \otimes \mathbb{B}))$, $(\mathbb{B}, \mathbb{B} \multimap (\mathbb{B} \otimes [\cdot]))$. For every type derivation π , $\mathcal{B}(\pi)$ is the sequence of all occurrences of $\mathbb{B}$ in π which are introduced by the rules (a_{q0}) and (a_{q1}) (from Figure 1). Similarly, $\mathcal{V}(\pi)$ is the corresponding sequence of binary digits, seen as a vector in $\mathbb{C}^{2^{|\mathcal{B}(\pi)|}}$. Both in $\mathcal{B}(\pi)$ and in $\mathcal{V}(\pi)$, the order is the one induced by the natural number labeling the underlying bit in π . We are finally able to define, for every π , the abstract machine $\mathcal{A}_\pi$ interpreting it:

- The *states* of $\mathcal{A}_\pi$ form a set $\mathcal{S}_\pi$ and are in the form $S = (O_1, \dots, O_n, \mathbf{Q})$ where:
 - $O_1, \dots, O_n$ are occurrences of the type $\mathbb{B}$ in π ;
 - $\mathbf{Q}$ is a *quantum register* on n qubits, i.e. a normalized vector in $\mathbb{C}^{2^n}$.
- The *transition relation* $\rightarrow_\pi \subseteq \mathcal{S}_\pi \times \mathcal{S}_\pi$ is defined based on π , following Figure 2. In the latter rule, each of the $2n$ occurrences of $\mathbb{B}$ in the type of U is simply denoted through its index, and for every $1 \leq k \leq m$, i_k is the position of $\mathbb{B}_k$ in the sequence $(\varphi_1, \mathbb{B}_{j_1}, \varphi_2, \dots, \varphi_m, \mathbb{B}_{j_m}, \varphi_{m+1})$. The transition rules induced by $(\mathbb{I}_{\multimap}^-)$ and $\mathbb{I}_{\otimes}$ have been elided for the sake of simplicity (see [7]).

The number of positive (negative, respectively) occurrences of $\mathbb{B}$ in the conclusion of π is said to be the *output arity* (the *input arity*, respectively) of π .

Given a type derivation π , the relation $\rightarrow_\pi$ enjoys a strong form of confluence [7]:

Proposition 1 (One-step Confluence of $\rightarrow_\pi$). *Let $S, R, T \in \mathcal{S}_\pi$ be such that $S \rightarrow_\pi R$ and $S \rightarrow_\pi T$.*

Then either $R = T$ or there exists a state U such that $R \rightarrow_\pi U$ and $T \rightarrow_\pi U$.

Suppose, for the sake of simplicity, that π is a type derivation of $\cdot \vdash M : A$. An *initial state* for $\mathbf{Q}$ is a state in the form $(\mathcal{N}(A) \cdot \mathcal{B}(\pi), \mathbf{Q} \otimes \mathcal{V}(\pi))$. Given a permutation σ on n elements, a *final state* for $\mathbf{Q}$ and σ is one in the form $(\varphi, \mathbf{Q})$, where $\varphi = \sigma(\mathcal{P}(A))$. Given a type derivation π , the partial function *computed by* π is $[\pi] : \mathbb{C}^{2^n} \rightarrow \mathbb{C}^{2^m}$ (where n and m are the input and output arity of π) and is defined by stipulating that $[\pi](\mathbf{Q}) = \sigma^{-1}(\mathbf{R})$ iff any initial state for $\mathbf{Q}$ rewrites into a final state for $\mathbf{R}$ and σ . Please observe that $[\pi]$ is either always undefined or always defined. Indeed, the fact that any initial configuration (for, say, $\mathbf{Q}$) rewrites to a final configuration or not does *not* depend on $\mathbf{Q}$ but only on π .

Example 2. Consider the term M_{EPR} and its type derivation π_{EPR} (see Example 1). Forgetting about terms and marking different occurrences of $\mathbb{B}$ with distinct indices, we obtain:

$$\begin{aligned} (\mathbb{B}_1, \mathbb{B}_2, \mathbf{Q}) &\rightarrow_\pi^* (\mathbb{B}_5, \mathbb{B}_6, \mathbf{Q}) \rightarrow_\pi^* (\mathbb{B}_{13}, \mathbb{B}_{14}, \mathbf{Q}) \rightarrow_\pi (\mathbb{B}_{17}, \mathbb{B}_{19}, \mathbf{Q}) \rightarrow_\pi^* (\mathbb{B}_{23}, \mathbb{B}_{20}, \mathbf{Q}) \\ &\rightarrow_\pi^* (\mathbb{B}_{24}, \mathbb{B}_{16}) \rightarrow_\pi (\mathbb{B}_{24}, \mathbb{B}_{10}, \mathbf{Q}) \rightarrow_\pi (\mathbb{B}_{21}, \mathbb{B}_{10}, \mathbf{Q}) \rightarrow_\pi (\mathbb{B}_{22}, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \\ &\rightarrow_\pi (\mathbb{B}_{18}, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \rightarrow_\pi (\mathbb{B}_{15}, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \rightarrow_\pi (\mathbb{B}_9, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \\ &\rightarrow_\pi (\mathbb{B}_{11}, \mathbb{B}_{12}, \mathbf{CNOT}^{1,2}(\mathbf{H}^1(\mathbf{Q}))) \rightarrow_\pi^* (\mathbb{B}_7, \mathbb{B}_8, \mathbf{CNOT}^{1,2}(\mathbf{H}^1(\mathbf{Q}))) \\ &\rightarrow_\pi (\mathbb{B}_3, \mathbb{B}_4, \mathbf{CNOT}^{1,2}(\mathbf{H}^1(\mathbf{Q}))). \end{aligned}$$

Notice that the occurrence of *CNOT* acts as a synchronization operator: the second token is stuck in the occurrence $\mathbb{B}_{10}$ until the first token arrives (from the occurrence $\mathbb{B}_{15}$) as a control input of the *CNOT* and the corresponding reduction step actually occurs.

4 Main Properties of IAM_{QL} : Termination, Progress, and Soundness

In this section, we will give some crucial results about IAM_{QL} . More specifically, we prove that runs of this token machine are indeed finite and end in final states. We proceed by putting QL in correspondence to MLL , inheriting the same kind of very elegant and powerful results enjoyed by MLL token machines. We do not have enough space to describe the translation of QL type derivations into MLL proof-nets here (see for more details [7]). The idea, however, is quite simple: while constructs like abstractions, applications, and pairs can be easily put in relation with logical rules from MLL , the same cannot be said about bits and unitary transformations. The idea, then, is to make the former to correspond to a dangling premise in the underlying sequent, while any n -ary unitary transformation should be put in relation with any permutation of n atoms. This way, one can prove that execution paths in the QL type derivation can be somehow simulated by paths in the MLL proof.

The first property we want to be sure about is that every computation of any token machine $\mathcal{A}_\pi$ always terminates. The second one is progress (i.e. deadlock-freedom). In both cases, we use in an essential way the correspondence between QL and MLL .

Proposition 2 (Termination). *For any quantum token machine $\mathcal{A}_\pi$, any sequence $S \rightarrow_\pi R \rightarrow_\pi \dots$ is finite.*

Proposition 3 (Progress). *Suppose π is a type derivation in QL and $S \in \mathcal{S}_\pi$ is initial. Moreover, suppose that $S \rightarrow_\pi^* R$. Then either R is final or $R \rightarrow_\pi T$ for some $T \in \mathcal{S}_\pi$.*

The immediate consequence of the termination and progress results is that $[\pi]$ is always a *total* function. The way $\mathcal{A}_\pi$ is defined ensures that $[\pi]$ is obtained by feeding some of the inputs of a unitary

operator U with some bits (namely those occurring in π). U is itself obtained by composing the unitary operators occurring in π , which can thus be seen as a program computing a quantum circuit. In a way, then, token machines both show that $Q\Lambda$ is a truly quantum calculus and can be seen as the right operational semantics for it.

The last step is to formalize the relation between token machines and the equational theory on superposed type derivations introduced in Section 2. It is easy to extend the definition of $[\cdot]$ to superposed type derivations: if $\mathcal{T} = \sum_{i=1}^n \kappa_i \pi_i$ then $[\mathcal{T}]$, when fed with a vector x , returns $\sum_{i=1}^n \kappa_i [\pi_i](x)$. Remarkably, token machines behave in accordance to the equational theory: this is our Soundness Theorem and the proof can be found in [7].

Theorem 1 (Soundness). *Given $\mathcal{T}$ and S superposed type derivations, if $\mathcal{T} \approx S$, then $[\mathcal{T}] = [S]$.*

5 Related Work

In [11], a geometry of interaction model for Selinger and Valiron’s quantum λ -calculus [13] is defined. The model is formulated in particle-style. In [3] QMLL, an extension of MLL with quantum modalities, is studied. QMLL is sound and complete with respect to quantum circuits, and an interactive (particle-style) abstract machine is defined. In both cases, adopting a particle-style approach has a bad consequence: the “quantum” tensor product does *not* coincide with the tensor product in the sense of linear logic. Here we show that adopting the wave-style approach solves the problem. Also quantum extensions of game semantics are partially connected to this work. See, for example [9, 8]. Purely linear quantum lambda-calculi (*with* measurements) can be given a fully abstract denotational semantics, like the one proposed by Selinger and Valiron [14]. In their work, closure (necessary to interpret higher-order functions) is not obtained via traces and is not directly related in any way to the geometry of interaction. Moreover, morphisms are just linear maps, and so the model is far from being a quantum operational semantics. A language of terms similar to $Q\Lambda$ has been also studied in [17], where the calculus of proof-nets MLL_{qm} is introduced. MLL_{qm} ’s syntax also includes a measurement box-like operator (which models the possibility of “observe” the value of a quantum bit [12]). A multi-token machine semantics for MLL_{qm} proof-nets is defined and proved to be *sound*, i.e. invariant along reduction of proof nets. Moreover, although a λ -calculus is given, together with a compilation scheme to MLL_{qm} proof-nets, the considered λ -calculus is one with *explicit* qubits, contrary to $Q\Lambda$.

6 Conclusions

We have introduced $IAM_{Q\Lambda}$, an interactive abstract machine which provides a sound operational characterization of any type derivation in a linear quantum λ -calculus $Q\Lambda$. This is an example of a concrete wave-style token machine whose runs cannot be seen simply as the asynchronous parallel composition of particle-style runs. Interestingly, synchronization is intimately related to entanglement: if, for example, only unary operators occur in a term (i.e. entanglement is *not* possible), synchronization is not needed and everything collapses to the particle-style. Our investigation is open to some possible future directions. A natural step would be to extend the syntax of terms and types with an exponential modality. The generalization of the token machine to this more expressive language would be an interesting and technically challenging subject. Moreover, giving a formal status to the connection between wave-style and the presence of entanglement is a fascinating subject which we definitely aim to investigate further. Finally-also in the light of what proved in [17]-an interesting proof-theoretical investigation will be to analyze the possible connections the with the deep inference-oriented graph formalism developed in [2].

References

- [1] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997.
- [2] R. Blute, A. Guglielmi, I. T. Ivanov, P. Panangaden, and L. Straßburger. A logical basis for quantum evolution and entanglement. In *Categories and Types in Logic, Language, and Physics*, volume 8222 of *LNCS*, pages 90–107, 2014.
- [3] U. Dal Lago and C. Faggian. On multiplicative linear logic, modality and quantum circuits. In *QPL*, volume 95 of *Electron. Proc. Theor. Comput. Sci.*, pages 55–66, 2011.
- [4] U. Dal Lago, A. Masini, and M. Zorzi. On a measurement-free quantum lambda calculus with classical control. *Math. Structures Comput. Sci.*, 19(2):297–335, 2009.
- [5] U. Dal Lago, A. Masini, and M. Zorzi. Quantum implicit computational complexity. *Theoret. Comput. Sci.*, 411(2):377–409, 2010.
- [6] U. Dal Lago, A. Masini, and M. Zorzi. Confluence results for a quantum lambda calculus with measurements. *Electron. Notes Theor. Comput. Sci.*, 270(2):251–261, 2011.
- [7] U. Dal Lago and M. Zorzi. Wave-style token machines and quantum lambda calculi (long version). <http://arxiv.org/abs/1307.0550>, 2013.
- [8] Y. Delbecque. Game semantics for quantum data. *Electron. Notes Theor. Comput. Sci.*, 270(1):41–57, 2011.
- [9] Y. Delbecque and P. Panangaden. Game semantics for quantum stores. *Electron. Notes Theor. Comput. Sci.*, 218:153–170, 2008.
- [10] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In *Proc. of the Logic Colloquium '88*, pages 221–260, 1989.
- [11] I. Hasuo and N. Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS*, pages 237–246, 2011.
- [12] M. Nielsen and I. Chuang. *Quantum computation and quantum information*. CUP, 2000.
- [13] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Math. Structures Comput. Sci.*, 16(3):527–552, 2006.
- [14] P. Selinger and B. Valiron. On a fully abstract model for a quantum linear functional language. *Electron. Notes Theor. Comput. Sci.*, 210:123–137, 2008.
- [15] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
- [16] A. van Tonder. A lambda calculus for quantum computation. *SIAM J. Comput.*, 33(5):1109–1135, 2004.
- [17] A. Yoshimizu, I. Hasuo, C. Faggian, and U. Dal Lago. Measurements in proof nets as higher-order quantum circuits. In *ESOP*, volume 8410 of *LNCS*, pages 371–391, 2014.
- [18] M. Zorzi. On quantum lambda calculi: a foundational perspective. To appear in *Math. Struct. in Comput. Sci.*, 2014.