Under consideration for publication in Math. Struct. in Comp. Science

A lambda calculus for quantum computation with classical control

PETER SELINGER[1†] and BENOIT VALIRON[2]

[1] *Department of Mathematics and Statistics, Dalhousie University, Halifax, Nova Scotia, Canada*
[2] *Department of Mathematics and Statistics, University of Ottawa, Ottawa, Ontario, Canada*

Received 18 April 2005; Revised 15 December 2005

In this paper, we develop a functional programming language for quantum computers, by extending the simply-typed lambda calculus with quantum types and operations. The design of this language adheres to the "quantum data, classical control" paradigm, following the first author's work on quantum flow-charts. We define a call-by-value operational semantics, and we give a type system using affine intuitionistic linear logic. The main results of this paper are the safety properties of the language and the development of a type inference algorithm.

1. Introduction

The objective of this paper is to develop a functional programming language for quantum computers. Quantum computing is a theory of computation based on the laws of quantum physics, rather than of classical physics. While no large-scale general-purpose quantum computer has yet been built, it is known that certain hard computational problems, such as integer factoring, can theoretically be solved efficiently on a quantum computer (Shor 1994). For this and other reasons, quantum computing has become a fast growing research area in recent years. For a good introduction to the subject, see Nielsen and Chuang (2002) or Preskill (1999).

The laws of quantum physics dictate that there are only two kinds of elementary operations that one can perform on a quantum state, namely *unitary transformations* and *measurements*. Many existing formalisms for quantum computation, such as the quantum circuit model, put an emphasis on the former, i.e., a computation is understood as the evolution of a quantum state by means of unitary gates. In these models, measurements are usually performed at the end of the computation, by an outside observer who is not part of the formalism proper. This means that a quantum computer is considered as a purely quantum system, without any classical parts. Examples of such models include the quantum Turing machine (Benioff 1980; Deutsch 1985), where the entire machine state, including the tape, the finite control, and the position of the head, is assumed to be a quantum state. Another example is the quantum lambda calculus of van Tonder (2004), which is a higher-order, purely quantum language without an explicit measurement operation.

On the other hand, some models for quantum computing have been proposed that combine

† Research supported by NSERC.

unitary operations and measurements into a single formalism. One such example is the *QRAM model* of Knill (1996), which is also described by Bettelli, Calarco and Serafini (2003). Here, a quantum computer consists of a classical computer connected to a quantum device. The operation of the machine is controlled by a classical program that emits a sequence of instructions to the quantum device for performing measurements and unitary operations. This situation is summarized by the slogan "quantum data, classical control" (Selinger 2004). In such a model, there is no explicit need for an outside "observer", as measurements can be performed by the device itself. Several programming languages have been proposed to deal with such a model (Bettelli, Calarco, and Serafini 2003; Sanders and Zuliani 2000), and the present paper is based on the work of Selinger (2004).

The main novelty of this paper is that we propose a *higher-order* quantum programming language, i.e., one in which functions can be considered as data. A typical feature of higher-order programming languages is that a program can take another program as an input (a situation called a "blackbox experiment" in physical terminology), or can produce another program as an output. There is no limit to the number of nesting levels of "programs within programs". Higher-order programming languages are often described in terms of the *lambda calculus*, a prototypical formalism introduced by Church and Curry in the 1930's, and we also follow this approach.

Because our language combines classical and quantum features, it is natural to consider two distinct basic data types: a type bit of classical bits and a type $qbit$ of quantum bits. These two types have very different properties. For instance, the value of a classical bit can be copied as many times as needed. On the other hand, a quantum bit cannot be duplicated, due to the well-known *no cloning property* of quantum physics (Nielsen and Chuang 2002; Preskill 1999). We therefore introduce a type system for our language that distinguishes between types whose elements are duplicable, and types whose elements are not. This distinction not only exists at basic types, but also at higher-order types: for example, some functions of type $qbit \to qbit$ can be called an unlimited number of times (such as the identity function), whereas others can only be called once (such as the function that returns a fixed qubit ϕ of unknown state). Hence, the question of whether a function is duplicable or not cannot be directly seen from the types of its arguments or of its result, but must be determined by inspecting the types of any free variables occurring in the function definition. As we will show, the appropriate type system for higher-order quantum functions in our setting is a variant of *affine intuitionistic linear logic* (Girard 1987).

We specify the behavior of programs in our language in terms of an operational semantics with probabilistic reduction rules. One of the main results of this paper is a set of safety properties (subject reduction and progress) of the operational semantics with respect to well-typed programs. We also give a type inference algorithm, which can be used to determine whether a given term is typable in the linear type system, and to find a type for it. Type inference is an interesting problem for this language, because the linear type system does not satisfy the principal type property. Our algorithm is based on the idea that linear types are decorations of intuitionistic ones.

This work is based on the second author's Master's thesis (Valiron 2004). A preliminary version of this paper appeared in TLCA 2005.

2. Quantum computing basics

We briefly recall the basic definitions of quantum computing; please see Nielsen and Chuang (2002) or Preskill (1999) for a complete introduction to the subject. The basic unit of information in quantum computation is a quantum bit or *qubit*. The state of a single qubit is described by a normalized vector of the 2-dimensional Hilbert space $\mathbb{C}^2$. We denote the standard basis of $\mathbb{C}^2$ as $\{|0\rangle, |1\rangle\}$, so that the general state of a single qubit can be written as $\alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. It is customary to identify any states that differ only by a global phase, i.e., $\alpha|0\rangle + \beta|1\rangle$ and $\alpha'|0\rangle + \beta'|1\rangle$ denote the same physical state if there is some scalar λ such that $\alpha' = \lambda\alpha$ and $\beta' = \lambda\beta$.

The state of n qubits is described by a normalized vector in $\otimes_{i=1}^{n}\mathbb{C}^2 \cong \mathbb{C}^{2^n}$. We write $|xy\rangle = |x\rangle \otimes |y\rangle$, so that a standard basis vector of $\mathbb{C}^{2^n}$ can be denoted $|\ulcorner i \urcorner^n\rangle$, where $\ulcorner i \urcorner^n$ is the binary representation of i in n digits, for $0 \leqslant i < 2^n$. As a special case, if $n = 0$, we denote the unique standard basis vector in $\mathbb{C}^1$ by $|\rangle$.

The basic operations on quantum states are unitary operations and measurements. A unitary operation maps an n-qubit state to an n-qubit state, and is given by a unitary $2^n \times 2^n$-matrix. It is common to assume that the computational model provides a certain set of built-in unitary operations, including for example the *Hadamard gate* H and the *controlled not-gate* $CNOT$, among others:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The measurement acts as a projection. When a qubit $\alpha|0\rangle + \beta|1\rangle$ is measured, the observed outcome is a classical bit. The two possible outcomes 0 and 1 are observed with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Moreover, the state of the qubit is affected by the measurement, and collapses to $|0\rangle$ if 0 was observed, and to $|1\rangle$ if 1 was observed. More generally, given an n-qubit state $|\phi\rangle = \alpha_0|0\rangle \otimes |\psi_0\rangle + \alpha_1|1\rangle \otimes |\psi_1\rangle$, where $|\psi_0\rangle$ and $|\psi_1\rangle$ are normalized $(n-1)$-qubit states, then measuring the leftmost qubit results in the answer i with probability $|\alpha_i|^2$, and the resulting state will be $|i\rangle \otimes |\psi_i\rangle$.

3. The untyped quantum lambda calculus

3.1. *Terms*

Our language uses the notation of the intuitionistic lambda calculus. For a detailed introduction to the lambda calculus, see e.g. Barendregt (1984). We start from a standard lambda calculus with booleans and finite products. We extend this language with three special quantum operations, which are new, $meas$, and built-in n-ary gates. new maps a classical bit to a quantum bit. $meas$ maps a quantum bit to a classical bit by performing a measurement operation; this is a probabilistic operation. Finally, we assume that there is a set $\mathcal{U}^n$ of built-in n-ary n-ary gates for each n. We use the letter U to range over built-in n-ary gates. Thus, the syntax of our language

is as follows:

$$\begin{array}{llcl} \mathit{Term} & M, N, P & ::= & x \mid MN \mid \lambda x.M \mid \mathit{if}\ M\ \mathit{then}\ N\ \mathit{else}\ P \mid 0 \mid 1 \mid \mathit{meas} \\ & & & \mid \mathit{new} \mid U \mid * \mid \langle M, N\rangle \mid \mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ N, \end{array}$$

We follow Barendregt's convention for identifying terms up to α-equivalence. We also sometimes use the shorthand notations

$$\begin{array}{rcl} \langle M_1, \ldots, M_n\rangle & = & \langle M_1, \langle M_2, \ldots\rangle\rangle, \\ \mathit{let}\ x = M\ \mathit{in}\ N & = & (\lambda x.N)M, \\ \lambda\langle x, y\rangle.M & = & \lambda z.(\mathit{let}\ \langle x, y\rangle = z\ \mathit{in}\ N). \end{array}$$

3.2. *Programs*

The reader will have noticed that we have not provided a syntax for constant quantum states such as $\alpha|0\rangle + \beta|1\rangle$ in our language. One may ask why we did not allow the insertion of quantum states into a lambda term, such as $\lambda x.(\alpha|0\rangle + \beta|1\rangle)$. The reason is that, in the general case, such a syntax would be insufficient. Consider for instance the lambda term $(\lambda y.\lambda f.fpy)(q)$, where p and q are entangled quantum bits in the state $|pq\rangle = \alpha|00\rangle + \beta|11\rangle$. Such a state cannot be represented locally by replacing p and q with some constant qubit expressions. The non-local nature of quantum states thus forces us to introduce a level of indirection into the representation of a state of a quantum program.

Definition 1. A *program state* is represented by a triple $[Q, L, M]$, where

— Q is a normalized vector of $\otimes_{i=0}^{n-1}\mathbb{C}^2$, for some $n \geqslant 0$
— M is a lambda term,
— L is a function from W to $\{0, \ldots, n-1\}$, where $FV(M) \subseteq W \subseteq \mathcal{V}_{term}$. L is also called the *linking function* or the *qubit environment*.

The purpose of the linking function is to assign specific free variables of M to specific quantum bits in Q. The notion of α-equivalence extends naturally to programs, for instance, the states $[|1\rangle, \{x \mapsto 0\}, \lambda y.x]$ and $[|1\rangle, \{z \mapsto 0\}, \lambda y.z]$ are equivalent. The set of program states, up to α-equivalence, is denoted by $\mathbb{S}$.

Convention 2. In order to simplify the notation, we will often use the following convention: we use p_i to denote the free variable x such that $L(x) = i$. A program $[Q, L, M]$ is abbreviated to $[Q, M']$ with $M' = M[p_{i_1}/x_1]\ldots[p_{i_n}/x_n]$, where $i_k = L(x_k)$.

3.3. *Linearity*

An important well-formedness property of quantum programs is that quantum bits should always be *uniquely referenced*: roughly, this means that no two variable occurrences should refer to the same physical quantum bit. The reason for this restriction is the well-known no-cloning property of quantum physics, which states that a quantum bit cannot be duplicated: there exists no physically meaningful operation which maps an arbitrary quantum bit $|\phi\rangle$ to $|\phi\rangle \otimes |\phi\rangle$.

Syntactically, the requirement of unique referencing translates into a *linearity condition*: A lambda abstraction $\lambda x.M$ is called *linear* if the variable x is used at most once during the evaluation of M. A well-formed program should be such that quantum data is only used linearly;

however, classical data, such as ordinary bits, can of course be used non-linearly. Since the decision of which subterms must be used linearly depends on type information, we will not formally enforce any linearity constraints until we discuss a type system in Section 4; nevertheless, we will assume that all our untyped examples are well-formed in the above sense.

3.4. *Evaluation strategy*

As is usual in defining a programming language, we need to settle on a reduction strategy. The obvious candidates are call-by-name and call-by-value. Because of the probabilistic nature of measurement, the choice of reduction strategy affects the behavior of programs, not just in terms of efficiency, but in terms of the actual answer computed. We demonstrate this in an example. Let $\mathbf{plus}$ be the boolean addition function, which is definable as $\mathbf{plus} = \lambda xy.\, \mathit{if}\ x\ \mathit{then}\ (\mathit{if}\ y\ \mathit{then}\ 0\ \mathit{else}\ 1)\ \mathit{else}\ (\mathit{if}\ y\ \mathit{then}\ 1\ \mathit{else}\ 0)$. Consider the term $M = (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}(H(\mathit{new}\ \ 0)))$.

Call-by-value. Reducing this in the empty environment, using the call-by-value reduction strategy, we obtain the following reductions:

$$\longrightarrow_{CBV} \quad [|0\rangle, (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}(H\ p_0))]$$

$$\longrightarrow_{CBV} \quad [\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}\ \ p_0)]$$

$$\longrightarrow_{CBV} \quad \begin{cases} [\ |0\rangle, (\lambda x.\mathbf{plus}\ x\ x)(0)] \\ [\ |1\rangle, (\lambda x.\mathbf{plus}\ x\ x)(1)] \end{cases}$$

$$\longrightarrow_{CBV} \quad \begin{cases} [\ |0\rangle, \mathbf{plus}\ 0\ 0] \\ [\ |1\rangle, \mathbf{plus}\ 1\ 1] \end{cases}$$

$$\longrightarrow_{CBV} \quad \begin{cases} [\ |0\rangle, 0] \\ [\ |1\rangle, 0] \end{cases}$$

where the two branches are taken with probability $1/2$ each. Thus, under call-by-value reduction, this program produces the boolean value 0 with probability 1. Note that we have used Convention 2 for writing these program states.

Call-by-name. Reducing the same term under the call-by-name strategy, we obtain in one step $[\ |\rangle, \mathbf{plus}\ (\mathit{meas}(H(\mathit{new}\ \ 0)))\ (\mathit{meas}(H(\mathit{new}\ \ 0)))]$, and then with probability $1/4$, $[\ |01\rangle, 1\ ]$, $[\ |10\rangle, 1\ ]$, $[\ |00\rangle, 0\ ]$ or $[\ |11\rangle, 0\ ]$. Therefore, the boolean output of this function is 0 or 1 with equal probability.

Mixed strategy. Moreover, if we mix the two reduction strategies, the program can even reduce to an ill-formed term. Namely, reducing by call-by-value until we reach the term $[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), (\lambda x.\mathbf{plus}\ x\ x)(\mathit{meas}\ \ p_0)]$, and then changing to call-by-name, we obtain in one step the term $[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \mathbf{plus}\ (\mathit{meas}\ \ p_0)\ (\mathit{meas}\ \ p_0)]$, which is not a valid program since there are two occurrences of p_0.

In the remainder of this paper, we will only consider the call-by-value reduction strategy, which seems to us to be the most natural.

3.5. *Probabilistic reduction systems*

In order to formalize the operational semantics of the quantum lambda calculus, we need to introduce the notion of a probabilistic reduction system.

Definition 3. A *probabilistic reduction system* is a tuple (X, U, R, prob) where X is a set of *states*, $U \subseteq X$ is a subset of *value states*, $R \subseteq (X \setminus U) \times X$ is a set of *reductions*, and $\mathit{prob} : R \to [0,1]$ is a *probability function*, where $[0,1]$ is the real unit interval. Moreover, we impose the following conditions:

— For any $x \in X$, $R_x = \{\ x' \mid (x, x') \in R\ \}$ is finite.
— $\sum_{x' \in R_x} \mathit{prob}(x, x') \leqslant 1$

We call prob the one-step reduction, and denote $x \to_p y$ to be $\mathit{prob}(x, y) = p$. Let us extend prob to the n-step reduction

$$\begin{aligned} \mathit{prob}^0(x,y) &= \begin{cases} 0 & \text{if} \quad x \neq y \\ 1 & \text{if} \quad x = y \end{cases} \\ \mathit{prob}^1(x,y) &= \begin{cases} \mathit{prob}(x,y) & \text{if} \quad (x,y) \in R \\ 0 & \text{else} \end{cases} \\ \mathit{prob}^{n+1}(x,y) &= \textstyle\sum_{z \in R_x} \mathit{prob}(x,z)\mathit{prob}^n(z,y), \end{aligned}$$

and the notation is extended to $x \to_p^n y$ to mean $\mathit{prob}^n(x,y) = p$.

We say that y is *reachable in one step with non-zero probability* from x, denoted $x \to_{>0} y$ when $x \to_p y$ with $p > 0$. We say that y is *reachable with non-zero probability* from x, denoted $x \to_{>0}^* y$ when there exists $n \geqslant 0$ such that $x \to_p^n y$ with $p > 0$.

We can then compute the probability to reach $u \in U$ from x: It is a function from $X \times U$ to $\mathbb{R}$ defined by $\mathit{prob}_U(x,u) = \sum_{n=0}^{\infty} \mathit{prob}^n(x,u)$. The total probability for reaching U from x is $\mathit{prob}_U(x) = \sum_{n=0}^{\infty} \sum_{u \in U} \mathit{prob}^n(x,u)$.

On the other hand, there is also the probability to *diverge* from x, or never reaching anything. This value is $\mathit{prob}_\infty(x) = \lim_{n \to \infty} \sum_{y \in X} \mathit{prob}^n(x,y)$.

Lemma 4. *For all* $x \in X$, $\mathit{prob}_U(x) + \mathit{prob}_\infty(x) \leqslant 1$. □

We define the *error probability of* x to be the number $\mathit{prob}_{err}(x) = 1 - \mathit{prob}_U(x) - \mathit{prob}_\infty(x)$.

Definition 5. We can define a notion of equivalence in X:

$$x \approx y \quad \text{iff} \quad \forall u \in U \begin{cases} \mathit{prob}_U(x,u) = \mathit{prob}_U(y,u) \\ \mathit{prob}_\infty(x) = \mathit{prob}_\infty(y) \end{cases}$$

Definition 6. In addition to the notion of reachability with non-zero probability, there is also a weaker notion of reachability, given by R: We will say that y is *reachable in one step* from x, written $x \rightsquigarrow y$, if xRy. By the properties of prob, $x \to_{>0} y$ implies $x \rightsquigarrow y$. As usual, $\rightsquigarrow^*$ denotes the transitive reflexive closure of $\rightsquigarrow$, and we say that y is *reachable* from x if $x \rightsquigarrow^* y$.

Definition 7. In a probabilistic reduction system, a state x is called an *error-state* if $x \notin U$ and $\sum_{x' \in X} prob(x, x') < 1$. An element $x \in X$ is *consistent* if there is no error-state e such that $x \rightsquigarrow^* e$.

Lemma 8. *If x is consistent, then $prob_{err}(x) = 0$.* □

Remark 9. We need the weaker notion of reachability $x \rightsquigarrow^* y$, in addition to reachability with non-zero probability $x \to_{>0}{}^* y$, because a null probability of getting a certain result is not an absolute warranty of its impossibility. In the QRAM, suppose we have a qubit in state $|0\rangle$. Measuring it cannot theoretically yield the value 1, but in practice, this might happen with small probability, due to imprecision of the physical operations and decoherence. Therefore, when we prove type safety (see Theorem 27), we will use the stronger notion. In short: a type-safe program should not crash, even in the event of random QRAM errors.

Remark 10. The converse of Lemma 8 is false. For instance, if $X = \{a, b\}$, $U = \emptyset$, $a \to_1 a$, and $a \to_0 b$, then b is an error state, and b is reachable from a, but only with probability zero. Hence $prob_{err}(a) = 0$, although a is inconsistent.

3.6. *Operational semantics*

We define a probabilistic call-by-value reduction procedure for the quantum lambda calculus. Note that, although the reduction itself is probabilistic, the choice of which redex to reduce at each step is deterministic.

Definition 11. *A* value *is a term of the following form:*

$$\mathit{Value} \qquad V, W \qquad ::= \qquad x \mid \lambda x.M \mid 0 \mid 1 \mid \mathit{meas} \mid \mathit{new} \mid U \mid * \mid \langle V, W\rangle.$$

The set of value states is $\mathbb{V} = \{[Q, L, V] \in \mathbb{S} \mid V \in \mathit{Value}\}$.

The reduction rules are shown in Table 1, where we have used Convention 2 to shorten the description of states. We write $[Q, L, M] \to_p [Q', L', M']$ for a single-step reduction of states which takes place with probability p. In the rule for reducing the term $U\langle p_{j_1}, \ldots, p_{j_n}\rangle$, U is an n-ary built-in unitary gate, $j_1, \ldots, j_n$ are pairwise distinct, and Q' is the quantum state obtained from Q by applying this gate to qubits $j_1, \ldots, j_n$. In the rule for measurement, $|Q_0\rangle$ and $|Q_1\rangle$ are normalized states of the form $|Q_0\rangle = \sum_j \alpha_j |\phi_j^0\rangle \otimes |0\rangle \otimes |\psi_j^0\rangle$ and $|Q_1\rangle = \sum_j \beta_j |\phi_j^1\rangle \otimes |1\rangle \otimes |\psi_j^1\rangle$, where ϕ_j^0 and ϕ_j^1 is an i-qubit state (so that the measured qubit is the one pointed to by p_i). In the rule for new, Q is an n-qubit state, so that $Q \otimes |i\rangle$ is an $(n+1)$-qubit state, and p_n refers to its rightmost qubit.

We define a weaker relation $\rightsquigarrow$. This relation models the transformations that can happen in the presence of decoherence and imprecision of physical operations. We define $[Q, M] \rightsquigarrow [Q', M']$ to be $[Q, M] \to_p [Q', M']$, even when $p = 0$, plus the additional rule, if Q and Q' are vectors of equal dimensions: $[Q, M] \rightsquigarrow [Q', M]$.

Lemma 12. *Let prob be the function such that for $x, y \in \mathbb{S}$, $prob(x, y) = p$ if $x \to_p y$ and 0 else. Then $(\mathbb{S}, \mathbb{V}, \rightsquigarrow, prob)$ is a probabilistic reduction system.* □

$$[Q, (\lambda x.M)V] \to_1 [Q, M[V/x]]$$

$$\frac{[Q, N] \to_p [Q', N']}{[Q, MN] \to_p [Q', MN']}$$

$$\frac{[Q, M] \to_p [Q', M']}{[Q, MV] \to_p [Q', M'V]}$$

$$\frac{[Q, M_1] \to_p [Q', M_1']}{[Q, \langle M_1, M_2\rangle] \to_p [Q', \langle M_1', M_2\rangle]}$$

$$\frac{[Q, M_2] \to_p [Q', M_2']}{[Q, \langle V_1, M_2\rangle] \to_p [Q', \langle V_1, M_2'\rangle]}$$

$$[Q, \mathit{if}\ 0\ \mathit{then}\ M\ \mathit{else}\ N] \to_1 [Q, N]$$

$$[Q, \mathit{if}\ 1\ \mathit{then}\ M\ \mathit{else}\ N] \to_1 [Q, M]$$

$$[Q, U\langle p_{j_1}, \ldots, p_{j_n}\rangle] \to_1 [Q', \langle p_{j_1}, \ldots, p_{j_n}\rangle]$$

$$[\alpha|Q_0\rangle + \beta|Q_1\rangle, \mathit{meas}\ \ p_i] \to_{|\alpha|^2} [|Q_0\rangle, 0]$$

$$[\alpha|Q_0\rangle + \beta|Q_1\rangle, \mathit{meas}\ \ p_i] \to_{|\beta|^2} [|Q_1\rangle, 1]$$

$$[Q, \mathit{new}\ \ 0] \to_1 [Q \otimes |0\rangle, p_n]$$

$$[Q, \mathit{new}\ \ 1] \to_1 [Q \otimes |1\rangle, p_n]$$

$$\frac{[Q, P] \to_p [Q', P']}{[Q, \mathit{if}\ P\ \mathit{then}\ M\ \mathit{else}\ N] \to_p [Q', \mathit{if}\ P'\ \mathit{then}\ M\ \mathit{else}\ N]}$$

$$\frac{[Q, M] \to_p [Q', M']}{[Q, \text{let } \langle x_1, x_2\rangle = M \text{ in } N] \to_p [Q', \text{let } \langle x_1, x_2\rangle = M' \text{ in } N]}$$

$$[Q, \text{let } \langle x_1, x_2\rangle = \langle V_1, V_2\rangle \text{ in } N] \to_1 [Q, N[V_1/x_1, V_2/x_2]]$$

Table 1. *Reductions rules of the quantum lambda calculus*

This probabilistic reduction system has error states, for example, the states $[Q, H(\lambda x.x)]$ or $[Q, U\langle p_0, p_0\rangle]$. Such error states correspond to run-time errors. In the next section, we introduce a type system designed to rule out such error states.

4. The typed quantum lambda calculus

We will now define a type system designed to eliminate all run-time errors arising from the reduction system of the previous section. We need base types (such as bit and qbit), function types, and product types. In addition, we need the type system to capture a notion of duplicability, as discussed in Section 3.3. We follow the notation of linear logic (Girard 1987). By default, a term of type A is assumed to be non-duplicable, and duplicable terms are given the type $!A$ instead. Formally, the set of types is defined as follows, where α ranges over a set of type constants and X ranges over a countable set of type variables:

$$\mathit{qType} \qquad A, B \qquad ::= \qquad \alpha \mid X \mid\ !A \mid (A \multimap B) \mid \top \mid (A \otimes B)$$

Note that, because all terms are assumed to be non-duplicable by default, the language has a linear function type $A \multimap B$ and a linear product type $A \otimes B$. This reflects the fact that there is in general no canonical diagonal function $A \to A \otimes A$. Also, $\top$ is the linear unit type. This will be made more formal in the typing rules below. We write $!^n A$ for $!!! \ldots !!A$, with n repetitions of !. We also write A^n for the n-fold tensor product $A \otimes \ldots \otimes A$.

4.1. *Subtyping*

The typing rules will ensure that any value of type $!A$ is duplicable. However, there is no harm in using it only once; thus, such a value should also have type A. For this reason, we define a subtyping relation $<:$ as follows:

$$\frac{}{\alpha <: \alpha}\ (\alpha) \qquad \frac{}{X <: X}\ (X) \qquad \frac{}{\top <: \top}\ (\top) \qquad \frac{A <: B}{!A <: B}\ (D) \qquad \frac{!A <: B}{!A <: !B}\ (!)$$

$$\frac{A_1 <: B_1 \quad A_2 <: B_2}{A_1 \otimes A_2 <: B_1 \otimes B_2}\ (\otimes) \qquad \frac{A <: A' \quad B <: B'}{A' \multimap B <: A \multimap B'}\ (\multimap)$$

Lemma 13. *For types A and B, if $A <: B$ and $(m = 0) \vee (n \geqslant 1)$, then $!^n A <: !^m B$.*

Proof. Repeated application of (D) and $(!)$. □

Notice that one can rewrite types using the notation:

$$qType \qquad A, B \qquad ::= \qquad !^n\alpha \mid !^n X \mid !^n(A \multimap B) \mid !^n\top \mid !^n(A \otimes B)$$

with $n \in \mathbb{N}$. Using the overall condition on n and m that $(m = 0) \vee (n \geqslant 1)$, the rules can be re-written as:

$$\frac{}{!^n\alpha <: !^m\alpha}\ (\alpha_2) \qquad \frac{}{!^n X <: !^m X}\ (X_2) \qquad \frac{}{!^n\top <: !^m\top}\ (\top_2)$$

$$\frac{A_1 <: B_1 \quad A_2 <: B_2}{!^n(A_1 \otimes A_2) <: !^m(B_1 \otimes B_2)}\ (\otimes_2) \qquad \frac{A <: A' \quad B <: B'}{!^n(A' \multimap B) <: !^m(A \multimap B')}\ (\multimap_2)$$

The two sets of rules are equivalent.

Lemma 14. *The rules of the second set are reversible.*

Proof. Note that for each possible type only one rule can be used. □

Lemma 15. *$(qType, <:)$ is reflexive and transitive. If we define an equivalence relation $\doteqdot$ by $A \doteqdot B$ iff $A <: B$ and $B <: A$, $(qType/\doteqdot, <:)$ is a poset.*

Proof. Both properties are shown by induction on the second set of rules. For transitivity, note that the condition $(m = 0) \vee (n \geqslant 1)$ can be re-written as $(n = 0) \Rightarrow (m = 0)$, which is transitive. □

Lemma 16. *If $A <: !B$, then there exists C such that $A = !C$.*

Proof. A direct application of the second set of rules. □

Remark 17. The subtyping rules are a syntactic device, and are not intended to catch all plausible type isomorphisms. For instance, the types $!A \otimes !B$ and $!(A \otimes B)$ are not subtypes of each other, although an isomorphism between these types is easily definable in the language.

4.2. *Typing rules*

We need to define what it means for a quantum state $[Q, L, M]$ to be well-typed. It turns out that the typing does not depend on Q and L, but only on M. We introduce typing judgments of the form $\Delta \rhd M : B$. Here M is a term, B is a $qType$, and Δ is a typing context, i.e., a function from a set of variables to $qType$. As usual, we write $|\Delta|$ for the domain of Δ, and we denote typing

$$\frac{A <: B}{\Delta, x{:}A \rhd x : B}\ (var) \qquad \frac{A_c <: B}{\Delta \rhd c : B}\ (const)$$

$$\frac{\Gamma_1, !\Delta \rhd P : bit \quad \Gamma_2, !\Delta \rhd M : A \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd \textit{if } P \textit{ then } M \textit{ else } N : A}\ (if)$$

$$\frac{\Gamma_1, !\Delta \rhd M : A \multimap B \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd MN : B}\ (app)$$

$$\frac{x{:}A, \Delta \rhd M : B}{\Delta \rhd \lambda x.M : A \multimap B}\ (\lambda_1) \qquad \begin{array}{c}\text{If } FV(M) \cap |\Gamma| = \emptyset\text{:}\\ \dfrac{\Gamma, !\Delta, x{:}A \rhd M : B}{\Gamma, !\Delta \rhd \lambda x.M : !^{n+1}(A \multimap B)}\end{array}\ (\lambda_2)$$

$$\frac{!\Delta, \Gamma_1 \rhd M_1 : !^n A_1 \quad !\Delta, \Gamma_2 \rhd M_2 : !^n A_2}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M_1, M_2 \rangle : !^n(A_1 \otimes A_2)}\ (\otimes.I) \qquad \frac{}{\Delta \rhd * : !^n\top}\ (\top)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes A_2) \quad !\Delta, \Gamma_2,\ x_1{:}!^n A_1,\ x_2{:}!^n A_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd \text{let } \langle x_1, x_2 \rangle = M \text{ in } N : A}\ (\otimes.E)$$

Table 2. *Typing rules*

contexts as $x_1{:}A_1, \ldots, x_n{:}A_n$. As usual, we write $\Delta, x{:}A$ for $\Delta \cup \{x{:}A\}$ if $x \notin |\Delta|$. Also, if $\Delta = x_1{:}A_1, \ldots, x_n{:}A_n$, we write $!\Delta = x_1{:}!A_1, \ldots, x_n{:}!A_n$. A typing judgment is called *valid* if it can be derived from the rules in Table 2.

The typing rule (ax) assumes that to every constant c of the language, we have associated a fixed type A_c. The types A_c are defined as follows:

$$\begin{array}{lll} A_0 = !bit & A_{new} = !(bit \multimap qbit) & \\ A_1 = !bit & A_{meas} = !(qbit \multimap !bit) & A_U = !(qbit^n \multimap qbit^n) \end{array}$$

Note that we have given the type $!(bit \multimap qbit)$ to the term new. Another possible choice would have been $!(!bit \multimap qbit)$, which makes sense because all classical bits are duplicable. However, since $!(bit \multimap qbit) <: !(!bit \multimap qbit)$, the second type is less general, and can be inferred by the typing rules.

The shorthand notations have the required behavior:

$$\frac{!\Delta, \Gamma_1, x{:}A \rhd N{:}B \quad !\Delta, \Gamma_2 \rhd M{:}A}{!\Delta, \Gamma_1, \Delta_2 \rhd \textit{let } x = M \textit{ in } N{:}B}, \quad \frac{!\Delta, \Gamma, x{:}A, y{:}B \rhd M{:}C}{!\Delta, \Gamma \rhd \lambda\langle x, y \rangle.M{:}(A \otimes B) \multimap C},$$

and if $FV(M) \cap |\Gamma| = \emptyset$, $\dfrac{!\Delta, \Gamma, x{:}!^n A, y{:}!^n B \rhd M{:}C}{!\Delta, \Gamma \rhd \lambda\langle x, y \rangle.M{:}!^{m+1}(!^n(A \otimes B) \multimap C)}$ are provable.

Note that, if $[Q, L, M]$ is a program state, the term M need not be closed; however, all of its free variables must be in the domain of L, and thus must be of type $qbit$. We therefore define:

Definition 18. A program state $[Q, L, M]$ is *well-typed of type B* if $\Delta \rhd M : B$ is derivable, where $\Delta = \{x{:}\, qbit \mid x \in FV(M)\}$. In this case, we write $[Q, L, M] : B$.

Note that the type system enforces that variables holding quantum data cannot be duplicated; thus, $\lambda x.\langle x, x \rangle$ is not a valid term of type $qbit \multimap qbit \otimes qbit$. On the other hand, we allow variables to be discarded freely. Other approaches are also possible, for instance, Altenkirch and Grattage (2004) propose a syntax that allows duplication but restricts discarding of quantum values.

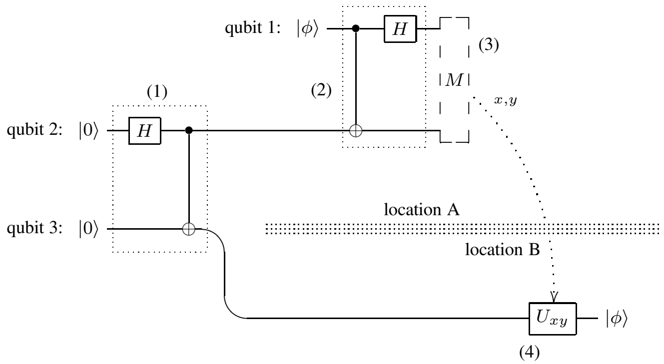

Table 3. *Quantum teleportation protocol*

4.3. *Example: quantum teleportation*

Let us illustrate the quantum lambda calculus and the typing rules with an example. The following is an implementation of the well-known quantum teleportation protocol (see e.g. Nielsen and Chuang (2002)). The purpose of the teleportation protocol is to send a qubit from location A to location B, using only classical communication and a pre-existing shared entangled quantum state. In fact, this can be achieved by communicating only the content of two classical bits. In the usual quantum circuit formalism, the teleportation protocol is described in Table 3.

The state $|\phi\rangle$ of the first qubit is "teleported" from location A to location B. The important point of the protocol is that the only quantum interaction between locations A and B (shown as (1) in the illustration) can be done *ahead of time*, i.e., before the state $|\phi\rangle$ is prepared.

The dashed box M (shown as (3)) represents a measurement of two qubits. The gate U_{xy} (shown as (4)) depends on two classical bits x and y, which are the result of this measurement. It is defined as:

$$U_{00} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, U_{01} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, U_{10} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, U_{11} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

The teleportation protocol consists of four steps:

(1) Create an entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ between qubits 2 and 3.
(2) At location A, rotate qubits 1 and 2.
(3) At location A, measure qubits 1 and 2, obtaining two classical bits x and y.
(4) At location B, apply the correct transformation U_{xy} to qubit 3.

Proof of the correctness of the teleportation protocol. The rotation (2) has the following effect:

$$\begin{array}{ccccc} & & CNOT & & H \otimes id \\ |00\rangle & \mapsto & |00\rangle & \mapsto & \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle), \\ |01\rangle & \mapsto & |01\rangle & \mapsto & \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle), \\ |10\rangle & \mapsto & |11\rangle & \mapsto & \frac{1}{\sqrt{2}}(|01\rangle - |11\rangle), \\ |11\rangle & \mapsto & |10\rangle & \mapsto & \frac{1}{\sqrt{2}}(|00\rangle - |10\rangle). \end{array}$$

If we apply it to the two first qubits of

$$(\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad = \quad \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle)$$

we get

$$\begin{array}{ll} & \frac{1}{2}(\alpha(|000\rangle + |100\rangle) + \alpha(|011\rangle + |111\rangle) + \beta(|010\rangle - |110\rangle) + \beta(|001\rangle - |101\rangle)) \\ = & \frac{1}{2}(|00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle) + |01\rangle \otimes (\alpha|1\rangle + \beta|0\rangle) \\ & +|10\rangle \otimes (\alpha|0\rangle - \beta|1\rangle) + |11\rangle \otimes (\alpha|1\rangle - \beta|0\rangle)) \end{array}$$

If we measure the two first qubits, the third qubit becomes

$$\begin{array}{ll} \alpha|0\rangle + \beta|1\rangle & \text{if 00 was measured,} \\ \alpha|1\rangle + \beta|0\rangle & \text{if 01 was measured,} \\ \alpha|0\rangle - \beta|1\rangle & \text{if 10 was measured,} \\ \alpha|1\rangle - \beta|0\rangle & \text{if 11 was measured.} \end{array}$$

Finally, note that if U_{xy} is applied in the case where x, y was measured, then the state of the last qubit is $\alpha|0\rangle + \beta|1\rangle = |\phi\rangle$. □

To express the quantum teleportation protocol in our quantum lambda calculus, we implement each part of the protocol as a function. We define three functions

$$\begin{array}{ll} \mathbf{EPR}: & !(\top \multimap (qbit \otimes qbit)) \\ \mathbf{BellMeasure}: & !(qbit \multimap (qbit \multimap bit \otimes bit)) \\ \mathbf{U}: & !(qbit \multimap (bit \otimes bit \multimap qbit)) \end{array}$$

The function **EPR** corresponds to step (1) of the protocol, and creates an entangled 2-qubit state. The function **BellMeasure** corresponds to steps (2) and (3), and takes two qubits, rotates and measures them. The function **U** corresponds to step (4). It takes a qubit q and two bits x, y and returns $U_{xy}q$. These functions are defined as follows:

$$\begin{array}{lll} \mathbf{EPR} & = & \lambda x.CNOT\langle H(new\,0), new\,0\rangle, \\ \mathbf{BellMeasure} & = & \lambda q_2.\lambda q_1.(let\ \langle p, p'\rangle = CNOT\langle q_1, q_2\rangle \\ & & \quad in\ \langle\, meas(Hp), meas\,p'\rangle), \\ \mathbf{U} & = & \lambda q.\lambda\langle x, y\rangle.\ if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q) \\ & & \qquad else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q), \end{array}$$

where U_{xy} are defined as above when the measured qubits were x and y.

The teleportation procedure can be seen as the creation of two non-duplicable functions f and g

$$\begin{array}{l} f : qbit \multimap bit \otimes bit, \\ g : bit \otimes bit \multimap qbit, \end{array}$$

such that $g \circ f(q) = q$ for an arbitrary qubit q. We can construct such a pair of functions by the following code:

$$\begin{array}{l} let\ \langle p, p'\rangle = \mathbf{EPR}\ * \\ \quad in\ let\ f = \mathbf{BellMeasure}\,p \\ \quad in\ let\ g = \mathbf{U}\,p' \\ \quad in\ \langle f, g\rangle. \end{array}$$

Note that, since f and g depend on the state of the qubits p and p', respectively, these functions cannot be duplicated, which is reflected in the fact that the types of f and g do not contain a top-level "!". The detailed typing derivation of these terms, and a proof that $g(f(q)) \to q$, using the reduction rules of Table 1, are given in the Appendix.

Superdense coding. The two functions f and g generated for the quantum teleportation protocol also satisfy a dual property, namely $(f \circ g)\langle x, y\rangle = \langle x, y\rangle$, for an arbitrary pair of classical bits $\langle x, y\rangle$. This property can be used to send two classical bits along a channel that can hold a single quantum bit, in the presence of a pre-existing shared entangled quantum state. This procedure is known as *superdense coding* (see Nielsen and Chuang (2002)), and it is dual to quantum teleportation. A detailed proof of $(f \circ g)\langle x, y\rangle \to \langle x, y\rangle$ from the reduction rules is given in the Appendix.

Remark 19. Note that the types $qbit$ and $bit \otimes bit$ are clearly not isomorphic. However, we have $f : qbit \multimap bit \otimes bit$ and $g : bit \otimes bit \multimap qbit$ such that $f \circ g = id$ and $g \circ f = id$. This is not a contradiction, of course, because each of f and g can only be used once, and therefore they are not isomorphisms in the usual sense. We might describe such a pair of functions as a pair of "single-use isomorphisms".

While this behavior of the functions f and g, and the corresponding properties of teleportation and superdense coding, are well-understood in quantum mechanics, this is still something of a mystery to us in the context of programming language semantics. We are not aware of any other situation in programming languages that produces such single-use isomorphisms.

4.4. *Properties of the type system*

We derive some basic properties of the type system.

Definition 20. We extend the subtyping relation to contexts by writing $\Delta <: \Delta'$ if $|\Delta'| = |\Delta|$ and for all x in $|\Delta'|$, $\Delta_f(x) <: \Delta'_f(x)$.

Lemma 21.

(1) If $x \notin FV(M)$ and $\Delta, x{:}A \rhd M{:}B$, then $\Delta \rhd M{:}B$.
(2) If $\Delta \rhd M{:}A$, then $\Gamma, \Delta \rhd M{:}A$.
(3) If $\Gamma <: \Delta$ and $\Delta \rhd M : A$ and $A <: B$, then $\Gamma \rhd M : B$.

Proof. By structural induction on the type derivation of M. □

The next lemma is crucial in the proof of the substitution lemma. Note that it is only true for a value V, and in general fails for an arbitrary term M.

Lemma 22. *If V is a value and $\Delta \rhd V : !A$, then for all $x \in FV(V)$, there exists some $U \in qType$ such that $\Delta(x) = !U$.*

Proof. By induction on V.

— If V is a variable x, then the last rule in the derivation was $\dfrac{B <: !A}{\Delta', x : B \rhd x : !A}$. Since $B <: !A$, B must be exponential by Lemma 16.

— If V is a constant c, then $FV(V) = \emptyset$, hence the result holds vacuously.

— If $V = \lambda x.M$, the only typing rule that applies is (λ_2), and $\Delta = \Gamma, !\Delta'$ with $FV(M) \cap |\Delta'| = \emptyset$. So every $y \in FV(M)$ except maybe x is exponential. Since $FV(\lambda x.M) = (FV(M) \setminus \{x\})$, this suffices.

— The remaining cases are similar. □

Lemma 23 (Substitution). *If V is a value such that $\Gamma_1, !\Delta, x{:}A \rhd M : B$ and $\Gamma_2, !\Delta \rhd V : A$, then $\Gamma_1, \Gamma_2, !\Delta \rhd M[V/x] : B$.*

Proof. By structural induction on the derivation of $\Gamma_1, !\Delta, x{:}A \rhd M : B$. □

Corollary 24. *If $\Gamma_1, !\Delta, x{:}A \rhd M : B$ and $\Gamma_2, !\Delta \rhd V : !^n A$, then $\Gamma_1, \Gamma_2, !\Delta \rhd M[V/x] : B$.*

Proof. From Lemma 23 and Lemma 21(3). □

Remark 25. We note that all the usual rules of affine intuitionistic linear logic are derived rules of our type system, *except* for the general promotion rule. Indeed, $\rhd\ new\,0 : qbit$ is valid, but $\rhd\ new\,0 : !qbit$ is not. However, the promotion rule is derivable when V is a *value*:

$$\frac{!\Gamma \rhd V : A}{!\Gamma \rhd V : !A.}$$

4.5. *Subject reduction and progress*

Theorem 26 (Subject Reduction). *Given a well-typed program $[Q, L, M]{:}B$ such that $[Q, L, M] \rightsquigarrow^* [Q', L', M']$, then $[Q', L', M'] : B$.*

Proof. It suffices to show this for $[Q, L, M] \to_p [Q', L', M']$, and we proceed by induction on the rules in Table 1. The rule $[Q, (\lambda x.M)V] \to_1 [Q, M[V/x]]$ and the rule for "let" use the substitution lemma. The remaining cases are direct applications of the induction hypothesis. □

Theorem 27 (Progress). *Let $[Q, L, M] : B$ be a well typed program. Then $[Q, L, M]$ is not an error state in the sense of Definition 7. In particular, either $[Q, L, M]$ is a value, or else there exist some state $[Q', L', M']$ such that $[Q, L, M] \to_p [Q', L', M']$. Moreover, the total probability of all possible single-step reductions from $[Q, L, M]$ is 1.*

Corollary 28. *Every sequence of reductions of a well-typed program either converges to a value, or diverges.* □

The proof of the Progress Theorem is similar to the usual proof, with two small differences. The first is the presence of probabilities, and the second is the fact that M is not necessarily closed. However, all the free variables of M are of type $qbit$, and this property suffices to prove the following lemma, which generalizes the usual lemma on the shape of closed well-typed values:

Lemma 29. *Suppose $\Delta = x_1{:}qbit, \ldots, x_n{:}qbit$, and V is a value. If $\Delta \rhd V{:}A \multimap B$, then V is new, $meas$, U, or a lambda abstraction. If $\Delta \rhd V{:}A \otimes B$, then $V = \langle V_1, V_2\rangle$. If $\Delta \rhd V{:}bit$, then $V = 0$ or $V = 1$.*

Proof. By inspection of the typing rules. □

Proof of the Progress Theorem. By induction on M. The claim follows immediately in the cases when M is a value, or when M is a left-hand-side of one of the rules in Table 1 that have no hypotheses. Otherwise, using Lemma 29, M is one of the following: PN, NV, $\langle N, P\rangle$, $\langle V, N\rangle$, $\mathit{if}\ N\ \mathit{then}\ P\ \mathit{else}\ Q$, $\mathit{let}\ \langle x, y\rangle = N\ \mathit{in}\ P$, where N is not a value. In this case, the free variables of N are still all of type qbit, and by induction hypothesis, the term $[Q, L, N]$ has reductions with total probability 1, and the rules in Table 1 ensure that the same is true for $[Q, L, M]$. □

5. Type inference algorithm

It is well-known that the simply-typed lambda calculus, as well as many programming languages, satisfies the *principal type property*: every untyped expression has a most general type, provided that it has any type at all. Since most principal types can usually be determined automatically, the programmer can be relieved from the need to write any types at all.

In the context of our quantum lambda calculus, it would be nice to have a type inference algorithm; however, the principal type property fails due to the presence of exponentials $!A$. Not only can an expression have several different types, but in general none of the types is "most general". For example, the term $M = \lambda xy.xy$ has possible types $T_1 = (A \multimap B) \multimap (A \multimap B)$ and $T_2 = !(A \multimap B) \multimap !(A \multimap B)$, among others. Neither of T_1 and T_2 is a substitution instance of the other, and in fact the most general type subsuming T_1 and T_2 is $X \multimap X$, which is not a valid type for M. Also, neither of T_1 and T_2 is a subtype of the other, and the most general type of which they are both subtypes is $(A \multimap B) \multimap !(A \multimap B)$, which is not a valid type for M.

In the absence of the principal type property, we need to design a type inference algorithm based on a different idea. The approach we follow is the one suggested by Danos, Joinet, and Schellinx (1995). The basic idea is to view a linear type as a "decoration" of an intuitionistic type. Our type inference algorithm is based on the following technical fact, given below: if a given term has an intuitionistic type derivation π of a certain kind, then it is linearly typable if and only if there exists a linear type derivation which is a decoration of π. Typability can therefore be decided by first doing intuitionistic type inference, and then checking finitely many possible linear decorations.

5.1. *Skeletons and decorations*

The class of *intuitionistic types* is

$$\mathit{iType} \qquad U, V \qquad ::= \qquad \alpha \mid X \mid (U \Rightarrow V) \mid (U \times V) \mid \top$$

where α ranges over the type constants and X over the type variables.

To each $A \in \mathit{qType}$, we associate its *type skeleton* ${}^\dagger A \in \mathit{iType}$, which is obtained by removing all occurrences of "!". Conversely, every $U \in \mathit{iType}$ can be lifted to some ${}^\clubsuit U \in \mathit{qType}$ with no occurrences of "!". Formally:

Definition 30. Define functions $\dagger : \mathit{qType} \to \mathit{iType}$ and $\clubsuit : \mathit{iType} \to \mathit{qType}$ by:

$$\begin{array}{lll}
{}^\dagger !^n\alpha = \alpha, \quad {}^\dagger !^n X = X, \quad {}^\dagger !^n\top = \top, & & {}^\clubsuit\alpha = \alpha, \quad {}^\clubsuit X = X, \quad {}^\clubsuit\top = \top,\\
{}^\dagger !^n(A \multimap B) = {}^\dagger A \Rightarrow {}^\dagger B, & & {}^\clubsuit(U \Rightarrow V) = {}^\clubsuit U \multimap {}^\clubsuit V,\\
{}^\dagger !^n(A \otimes B) = {}^\dagger A \times {}^\dagger B, & & {}^\clubsuit(U \times V) = {}^\clubsuit U \otimes {}^\clubsuit V.
\end{array}$$

If $U = {}^\dagger A$, then we also say that A is a *decoration* of U.

Lemma 31. *If $A <: B$, then ${}^\dagger A = {}^\dagger B$. If $U \in \mathit{iType}$, then $U = {}^{\dagger\clubsuit}U$.* □

Writing $\Delta \blacktriangleright M : U$ for a typing judgment of the simply-typed lambda calculus, we can extend the notion of skeleton to contexts, typing judgments, and derivations as follows:

$$\begin{array}{rcl}
{}^\dagger\{x_1{:}A_1, \ldots, x_n{:}A_n\} & = & \{x_1{:}{}^\dagger A_1, \ldots, x_n{:}{}^\dagger A_n\}\\
{}^\dagger(\Delta \rhd M : A) & = & ({}^\dagger\Delta \blacktriangleright M : {}^\dagger A).
\end{array}$$

From the rules in Table 2, it is immediate that if $\Delta \rhd M : A$ is a valid typing judgment in the quantum lambda calculus, then ${}^\dagger(\Delta \rhd M : A) = ({}^\dagger\Delta \blacktriangleright M : {}^\dagger A)$ is a valid typing judgment in the simply-typed lambda calculus.

5.2. *Decorating intuitionistic type derivations*

The basic idea of our quantum type inference algorithm is the following: given a term M, first find an intuitionistic typing judgment $\Delta \blacktriangleright M : U$, say with type derivation π, if such a typing exists. Then look for a quantum type derivation which is a decoration of π. Clearly, if the term M is not quantum typable, this procedure will fail to yield a quantum typing of M. For the algorithm to be correct, we also need the converse property to be true: if M has any quantum type derivation, then it has a quantum type derivation which is a decoration of the given intuitionistic derivation π. We therefore would ideally like to prove the following property:

Property 32 (desired). Let M be a term with an intuitionistic type derivation π. Then M is quantum typable if and only if there exists a quantum type derivation π' of M such that ${}^\dagger\pi' = \pi$.

Unfortunately, this property is false, as the following example shows.

Example 33. Consider the term $M = (\lambda x.\, \mathit{meas}\, x)(\mathit{new}\, 0)$. Clearly this term is quantum typable, for instance, it has type bit (also $!\mathit{bit}$, $!!\mathit{bit}$ etc.). Consider the following intuitionistic type derivation π for M:

$$\dfrac{\dfrac{\dfrac{x : \mathit{qbit} \blacktriangleright \mathit{meas} : \mathit{qbit} \Rightarrow \mathit{bit} \quad x : \mathit{qbit} \blacktriangleright x : \mathit{qbit}}{x : \mathit{qbit} \blacktriangleright \mathit{meas}\, x : \mathit{bit}}}{\blacktriangleright \lambda x.\, \mathit{meas}\, x : \mathit{qbit} \Rightarrow \mathit{bit}} \qquad \dfrac{\blacktriangleright \mathit{new} : \mathit{bit} \Rightarrow \mathit{qbit} \quad \blacktriangleright 0 : \mathit{bit}}{\blacktriangleright \mathit{new}\, 0 : \mathit{qbit}}}{\blacktriangleright (\lambda x.\, \mathit{meas}\, x)(\mathit{new}\, 0) : \mathit{bit}}$$

This particular intuitionistic type derivation is not the skeleton of any valid quantum type derivation of M. To see this, note that the variable x has been duplicated in the typing rule for $\mathit{meas}\, x$. Therefore, any valid decoration of π has to give the type $!\mathit{qbit}$ to x. On the other hand, the only valid quantum type for $\mathit{new}\, 0$ is qbit, which is not a subtype of $!\mathit{qbit}$. Hence, there is no quantum type derivation for M whose skeleton is π, demonstrating that Property 32 fails.

5.3. *Normal derivations*

The reason Property 32 fails is because an intuitionistic derivation can duplicate variables unnecessarily, as shown in Example 33. The duplication of a variable in a typing rule is unnecessary if the variable does not actually occur in one of the premises. We can avoid this problem by slightly changing the typing rules to disallow such unnecessary duplications. This is done by eliminating all "dummy" variables from typing contexts.

Definition 34. A typing judgment $\Delta \rhd M : A$ of the quantum lambda calculus is called *normal* if $|\Delta| = FV(M)$. If $\Delta \rhd M : A$ is any typing judgment, then its *normal form* is $\Delta|_{FV(M)} \rhd M : A$. We also write $\Delta|_M$ for $\Delta|_{FV(M)}$. If π is a type derivation, then its normal form is the derivation $\mathcal{N}(\pi)$ obtained by taking the normal form of each of its nodes.

Note that the normal form of a type derivation is not necessarily a type derivation in the strict sense, because the rules of Table 2 are not invariant under taking normal forms. However, we can define a new set of typing rules, called the *normal typing rules*, which are obtained by normalizing the rules from Table 2. For example, the new rule for application is:

$$\frac{\{\Gamma_1, !\Delta\}|_{FV(M)} \rhd M : A \multimap B \quad \{\Gamma_2, !\Delta\}|_{FV(N)} \rhd N : A}{\{\Gamma_1, \Gamma_2, !\Delta\}|_{FV(MN)} \rhd MN : B}\ (app_{norm})$$

We treat all the other typing rules analogously.

Lemma 35. *Let $\Delta \rhd M : A$ be any typing judgment. Then $\Delta \rhd M : A$ is derivable from the rules in Table 2 if and only if $\Delta|_{FV(M)} \rhd M : A$ is derivable from the normal typing rules.*

Proof. The left-to-right implication follows by normalizing the type derivation of $\Delta \rhd M : A$. The opposite implication follows because the normal typing rules are admissible by Lemma 21. □

The normal form of intuitionistic typing judgments, rules, and derivations is defined analogously. The counterpart of Lemma 35 also holds in the intuitionistic case.

Relative to the normal typing rules, the analog of Property 32 holds.

Theorem 36. *Let M be a term with a normal intuitionistic type derivation π. Then M is quantum typable if and only if there exists a normal quantum type derivation π' of M such that ${}^\dagger\pi' = \pi$.*

5.4. *Proof of Theorem 36*

The proof of Theorem 36 requires us to find a suitable decoration π' of π. For this purpose we are going to introduce the concept of the decoration of an intuitionistic type *along* a quantum type. Intuitively, $U \looparrowright A$ takes the skeleton from U and the exponentials from A.

Definition 37. Given $A \in qType$ and $U \in iType$, we define the *decoration* $U \looparrowright A \in qType$ *of U along A* by

$$\begin{array}{l}
U \looparrowright !^n A = !^n(U \looparrowright A),\\
(U \Rightarrow V) \looparrowright (A \multimap B) = (U \looparrowright A) \multimap (V \looparrowright B),\\
(U \times V) \looparrowright (A \otimes B) = (U \looparrowright A) \otimes (V \looparrowright B),\\
\text{in all other cases:} \quad U \looparrowright A = {}^\clubsuit U.
\end{array}$$

Lemma 38. *If $U, V \in iType$ and $A, B \in qType$, then the following are true:*

(a) ${}^\dagger(U \looparrowright A) = U$,

(b) If ${}^\dagger A = U$ *then* $U \looparrowright A = A$,

(c) If $A <: B$ *then* $(U \looparrowright A) <: (U \looparrowright B)$. □

Definition 39. Let Γ be an intuitionistic typing context, and Δ a quantum typing context, such that $|\Gamma| \subseteq |\Delta|$. Then we define $\Gamma \looparrowright \Delta := \Gamma'$, where $|\Gamma'| = |\Gamma|$, and for all x in $|\Gamma|$, $\Gamma'(x) = \Gamma(x) \looparrowright \Delta(x)$. This notation is extended to typing judgments in the following way, provided that $|\Gamma| \subseteq |\Delta|$:

$$(\Gamma \blacktriangleright M : U) \looparrowright (\Delta \rhd M : A) := \Gamma \looparrowright \Delta \rhd M : U \looparrowright A,$$

and to type derivations by structural induction, provided that the intuitionistic derivation is normal.

Lemma 40. *If π is a normal intuitionistic type derivation and if ρ is any quantum type derivation, then $\pi' := (\pi \looparrowright \rho)$ is a normal quantum type derivation.*

Proof. By structural induction on ρ, and by case distinction on the last typing rule used. For instance, suppose the last rule used was the (app) rule. Then $M = NP$ and the type derivation ρ ends in

$$\frac{\begin{matrix}\vdots\,\rho_1 \\ \Delta_1, !\Delta_3 \rhd N : A \multimap B\end{matrix} \quad \begin{matrix}\vdots\,\rho_2 \\ \Delta_2, !\Delta_3 \rhd P : A\end{matrix}}{\Delta_1, \Delta_2, !\Delta_3 \rhd NP : B}$$

In normal intuitionistic lambda calculus the type derivation π is of the form:

$$\frac{\begin{matrix}\vdots\,\pi_1 \\ \Gamma|_{FV(N)} \blacktriangleright N : U \Rightarrow V\end{matrix} \quad \begin{matrix}\vdots\,\pi_2 \\ \Gamma|_{FV(P)} \blacktriangleright P : U\end{matrix}}{\Gamma|_{FV(NP)} \blacktriangleright NP : V}$$

Writing $\Gamma|_X$ for $\Gamma|_{FV(X)}$, the type derivation $\pi \looparrowright \rho$ is

$$\frac{\begin{matrix}\vdots\,\pi_1 \looparrowright \rho_1 \\ \Gamma|_N \looparrowright (\Delta_1, !\Delta_3) \rhd N : (U \Rightarrow V) \looparrowright (A \multimap B)\end{matrix} \quad \begin{matrix}\vdots\,\pi_2 \looparrowright \rho_2 \\ \Gamma|_P \looparrowright (\Delta_2, !\Delta_3) \rhd P : U \looparrowright A\end{matrix}}{\Gamma|_{NP} \looparrowright (\Delta_1, \Delta_2, !\Delta_3) \rhd NP : V \looparrowright B.}$$

By induction hypothesis, $\pi_1 \looparrowright \rho_1$ and $\pi_2 \looparrowright \rho_2$ are quantum normal type derivations. If we write Γ_i for $\Gamma|_{dom\Delta_i} \looparrowright \Delta_i$, using Lemma 38 and the definition of $\looparrowright$, the last rule of the derivation above becomes:

$$\frac{\{\Gamma_1, !\Gamma_3\}|_N \rhd N : (U \looparrowright A) \multimap (V \looparrowright B) \quad \{\Gamma_2, !\Gamma_3\}|_P \rhd P : U \looparrowright A}{\{\Gamma_1, \Gamma_2, !\Gamma_3\}|_{NP} \rhd NP : V \looparrowright B,}$$

which is an instance of the normal quantum (app) rule. Thus $\pi' := (\pi \looparrowright \rho)$ is a normal quantum type derivation. The other typing rules are treated similarly. □

Proof of Theorem 36. For the left-to-right implication, if ρ is some quantum type derivation of M, we can define $\pi' = (\pi \looparrowright \rho)$ as in Lemma 40. Then ${}^\dagger\pi' = \pi$ follows from Lemma 38. The right-to-left implication follows trivially from Lemma 35. □

5.5. *Elimination of repeated exponentials*

The type system in Section 4 allows types with repeated exponentials such as $!!A$. While this is useful for compositionality, it is not very convenient for type inference. We therefore consider a reformulation of the typing rules which only requires single exponentials.

Definition 41. For $A \in qType$, we define ${}^{\#}A \in qType$ to be the result of erasing multiple exponentials in A. Formally, if $\sigma(0) = 0$ and $\sigma(n+1) = 1$,

$$
\begin{array}{l}
{}^{\#}!^n\alpha = !^{\sigma(n)}\alpha, \quad {}^{\#}!^nX = !^{\sigma(n)}X, \quad {}^{\#}!^n\top = !^{\sigma(n)}\top, \\
{}^{\#}!^n(A \multimap B) = !^{\sigma(n)}({}^{\#}A \multimap {}^{\#}B), \quad {}^{\#}!^n(A \otimes B) = !^{\sigma(n)}({}^{\#}A \otimes {}^{\#}B),
\end{array}
$$

We also extend this operation to typing contexts and judgments in the obvious way.

Lemma 42. *The following are derived rules of the type system in Table 2, for all $\tau, \sigma \in \{0, 1\}$.*

$$
\frac{!\Delta, \Gamma_1 \rhd M_1 : !A_1 \quad !\Delta, \Gamma_2 \rhd M_2 : !A_2}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M_1, M_2\rangle : !(!^\tau A_1 \otimes !^\sigma A_2)} \ (\otimes.I')
$$

$$
\frac{!\Delta, \Gamma_1 \rhd M : !(!^\tau A_1 \otimes !^\sigma A_2) \quad !\Delta, \Gamma_2,\ x_1{:}!A_1,\ x_2{:}!A_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd \mathit{let}\ \langle x_1, x_2\rangle = M\ \mathit{in}\ N : A} \ (\otimes.E')
$$

Further, the normal forms of $(\otimes.I')$ and $(\otimes.E')$ are derivable in the normal type system.

Proof. Suppose $!\Delta, \Gamma_1 \rhd M_1 : !A_1$ and $!\Delta, \Gamma_2 \rhd M_2 : !A_2$ are derivable. Since $!A_1 <: !!^\tau A_1$ and $!A_2 <: !!^\sigma A_2$, therefore $!\Delta, \Gamma_1 \rhd M_1 : !!^\tau A_1$ and $!\Delta, \Gamma_2 \rhd M_2 : !!^\sigma A_2$ are also derivable by Lemma 21(3). But then $!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M_1, M_2\rangle : !(!^\tau A_1 \otimes !^\sigma A_2)$ follows from rule $(\otimes.I)$. The proof of the second rule is similar. Finally, the last claim follows from Lemma 35. □

Lemma 43. *If π is a derivation of a typing judgment $\Delta \rhd M : A$ from the normal quantum typing rules, then ${}^{\#}\pi$ is a valid normal derivation of ${}^{\#}\Delta \rhd M : {}^{\#}A$, possibly using the normal forms of $(\otimes.I')$ and $(\otimes.E')$ as additional rules. Moreover, ${}^{\dagger}\pi = {}^{\dagger\#}\pi$.*

Proof. By inspection of the rules. For each normal typing rule r, ${}^{\#}r$ is either an instance of the same rule, or of the normal form of $(\otimes.I')$ or $(\otimes.E')$. □

5.6. *Description of the type inference algorithm*

Theorem 36 yields a simple type inference algorithm. Given a term M, we can perform type inference in the quantum lambda calculus in three steps:

(1) Find an intuitionistic type derivation π of M, if any.

(2) Eliminate "dummy" variables to obtain its normal form $\mathcal{N}\pi$.

(3) Find a decoration of $\mathcal{N}\pi$ which is a valid normal quantum type derivation, if any.

Step (1) is known to be decidable, and step (2) is computationally trivial. By Theorem 36, step (3) will succeed if and only if M is quantum typable. Note that by Lemma 43, it suffices to consider decorations of $\mathcal{N}\pi$ without repeated exponentials. Since there are only finitely many such decorations, step (3) is clearly decidable. Also note that if the algorithm succeeds, then it returns a possible type for M. However, it does not return a description of all possible types.

Remark 44 (Efficiency of the algorithm). In principle, the search space of all possible decorations of $\mathcal{N}\pi$ is exponential in size. However, this space can be searched efficiently by solving a system of constraints. More precisely, if we create a boolean variable for each place in the type derivation which potentially can hold a "!", then the constraints imposed by the linear type system can all be written in the form of implications $x_1 \wedge \ldots \wedge x_n \Rightarrow y$, where $n \geqslant 0$, and negations $\neg z$. It is well-known that such a system can be solved in polynomial time in the number of variables and clauses. Therefore, the type inference problem can be solved in time polynomial in the size of the type derivation π.

Note, however, that the size of an intuitionistic type derivation π need not be polynomial in the size of the term M, because in the worst case, π can contain types that are exponentially larger than M. We do not presently know whether quantum typability can be decided in time polynomial in M.

6. Conclusion and further work

In this paper, we have defined a higher-order quantum programming language based on a linear typed lambda calculus. Compared to the quantum lambda calculus of van Tonder (2004), our language is characterized by the fact that it contains classical as well as quantum features; for instance, we provide classical datatypes and measurements as a primitive feature of our language. Moreover, we provide a subject reduction result and a type inference algorithm. As the language shows, linearity constraints do not just exist at base types, but also at higher types, due to the fact that higher-order functions are represented as closures, which may in turns contain embedded quantum data. We have shown that a version of affine intuitionistic linear logic provides the right type system to deal with this situation.

There are many open problems left for further work. Several interesting variations of the language presented here need to be explored in more detail. For instance, we have not included a syntax for recursive function definitions in this paper. We believe that this can be done, but the details are more subtle than we first expected. Another obvious extension is to add the additive types of linear logic to the system. One may also study alternative reduction strategies. In this paper, we have considered the call-by-value strategy, because it conforms with our intuition of quantum computation as being essentially value-driven. However, a call-by-name strategy is also conceivable and would lead to a very different semantics and type system. Finally, an important problem that we have not addressed here is how to give a denotational semantics for higher order quantum programming languages. This appears to be a difficult problem and is the subject of ongoing research.

Appendix A. Examples

A.1. *Example: Type derivation of the teleportation protocol*

To illustrate the linear type system from Section 4.2, we give a complete derivation of the type of the quantum teleportation term from Section 4.3. The notation $(L.x.y)$ means that Lemma.$x.y$ is used.

Computing some subtypes:

1	α_2	$!^n\alpha <: \alpha$
2	α_2	$!^m\beta <: \beta$
3	$\multimap_2, 1, 2$	$!^k(\alpha \multimap !^m\beta) <: (!^n\alpha \multimap \beta)$
4	$(L.15)$	$A <: A$
5	$D, 4$	$!A <: A$

Computing the type of **EPR**:

6	$const, 3$	$\rhd\ new : bit \multimap qbit$
7	$const, 5$	$\rhd\ 0 \colon bit$
8	$app, 6, 7$	$\rhd\ new\, 0 \colon qbit$
9	$const, 3$	$\rhd\ H \colon qbit \multimap qbit$
10	$app, 9, 8$	$\rhd\ H(new\, 0) \colon qbit$
11	$\otimes.I, 10, 9$	$\rhd\ \langle H(new\, 0), new\, 0\rangle : qbit \otimes qbit$
12	$const, 3$	$x{:}\top \rhd CNOT{:}(qbit \otimes qbit) \multimap (qbit \otimes qbit)$
13	$app, 12, 11$	$x{:}\top \rhd CNOT\langle H(new\, 0), new\, 0\rangle \colon qbit \otimes qbit$
14	$\lambda_2, 13$	$\rhd\ \lambda x.CNOT\langle H(new\, 0), new\, 0\rangle{:}!(\top \multimap (qbit \otimes qbit))$

Computing the type of **BellMeasure**:

15	$var, 1$	$y{:}qbit \rhd y{:}qbit$
16	$const, 3$	$\rhd\ meas{:}qbit \multimap bit$
17	$app, 16, 15$	$y{:}\, qbit \rhd meas\, y{:}bit$
18	$var, 1$	$x{:}\, qbit \rhd x{:}qbit$
19	$app, 9, 18$	$x{:}qbit \rhd Hx{:}qbit$
20	$app, 16, 19$	$x{:}\, qbit \rhd meas(Hx){:}bit$
21	$var, 1$	$q_1{:}\, qbit \rhd q_1{:}\, qbit$
22	$var, 1$	$q_2{:}\, qbit \rhd q_2{:}\, qbit$
23	$\otimes.I, 21, 22$	$q_2{:}\, qbit, q_1{:}\, qbit \rhd \langle q_1, q_2\rangle \colon qbit \otimes qbit$
24	$const, 3$	$\rhd\ CNOT{:}(qbit \otimes qbit) \multimap (qbit \otimes qbit)$
25	$app, 24, 23$	$q_2{:}\, qbit, q_1{:}\, qbit \rhd CNOT\langle q_1, q_2\rangle \colon qbit \otimes qbit$
26	$\otimes.I, 20, 17$	$x{:}\, qbit, y{:}\, qbit \rhd \langle\, meas(Hx), meas\, y\rangle \colon bit \otimes bit$
27	$\otimes.E, 25, 26$	$q_2{:}\, qbit, q_1{:}\, qbit \rhd let\ \langle x, y\rangle = CNOT\langle q_1, q_2\rangle$ $in\langle\, meas(Hx), meas\, y\rangle \colon bit \otimes bit$
28	$\lambda_1, 27$	$q_2{:}\, qbit \rhd \lambda q_1.(let\ \langle x, y\rangle = CNOT\langle q_1, q_2\rangle$ $in\ \langle\, meas(Hx), meas\, y\rangle){:}qbit \multimap bit \otimes bit$
29	$\lambda_2, 28$	$\rhd\ \lambda q_2.\lambda q_1.(let\ \langle x, y\rangle = CNOT\langle q_1, q_2\rangle$ $in\ \langle\, meas(Hx), meas\, y\rangle){:}!(qbit \multimap (qbit \multimap bit \otimes bit))$

Computing the type of **U**:

30	$var, 1$	$q{:}qbit \rhd q{:}qbit$
31	$const, 3$	$\rhd\ U_{ij}{:}qbit \multimap qbit$
32	$app, 30, 31$	$q{:}qbit \rhd U_{ij}q{:}qbit$
33	$var, 1$	$y{:}bit \rhd y{:}!bit$
34	$var, 1$	$x{:}bit \rhd x{:}!bit$
35	$if, 33, 32, 32$	$q{:}qbit, y{:}bit \rhd \text{if } y \text{ then } U_{i1}q \text{ else } U_{i0}q{:}qbit$
36	$if, 34, 35, 35$	$q{:}qbit, x{:}bit, y{:}bit \rhd if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)$ $else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q) \colon qbit$
37	$\multimap_1', 36$	$q{:}qbit \rhd \lambda\langle x, y\rangle.if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)$ $else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q) \colon bit \otimes bit \multimap qbit$
38	$\multimap_2, 37$	$\rhd\ \lambda q.\lambda\langle x, y\rangle.if\ x\ then\ (if\ y\ then\ U_{11}q\ else\ U_{10}q)$ $else\ (if\ y\ then\ U_{01}q\ else\ U_{00}q){:}!(qbit \multimap (bit \otimes bit \multimap qbit))$

Finally, computing the type of the pair $\langle f, g\rangle$:

39	$\top$	$\rhd\ *{:}\top$
40	$(L.21), 14, 5$	$\rhd\ \mathbf{EPR}{:}\top \multimap (qbit \otimes qbit)$
41	$app, 40, 39$	$\rhd\ \mathbf{EPR}\, *{:}qbit \otimes qbit$
42	$(L.21), 29, 5$	$\rhd\ \mathbf{BellMeasure}{:}qbit \multimap (qbit \multimap bit \otimes bit)$
43	$var, 1$	$x{:}qbit \rhd x{:}qbit$
44	$app, 42, 43$	$x{:}qbit \rhd \mathbf{BellMeasure}\ x \colon qbit \multimap bit \otimes bit$
45	$var, 1$	$y{:}qbit \rhd y{:}qbit$
46	$(L.21), 38, 5$	$\rhd\ \mathbf{U} \colon qbit \multimap (bit \otimes bit \multimap qbit)$
47	$app, 46, 45$	$y{:}qbit \rhd \mathbf{U}\ y \colon bit \otimes bit \multimap qbit$
48	$var, 1$	$f{:}qbit \multimap bit \otimes bit \rhd f{:}qbit \multimap bit \otimes bit$
49	$var, 1$	$g{:}\, bit \otimes bit \multimap qbit \rhd g{:}\, bit \otimes bit \multimap qbit$
50	$\otimes, 48, 49$	$g{:}\, bit \otimes bit \multimap qbit, f{:}\, qbit \multimap bit \otimes bit \rhd \langle f, g\rangle{:}$ $(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$
51	$let, 47, 50$	$f{:}\, qbit \multimap bit \otimes bit, y{:}qbit \rhd let\ g = \mathbf{U}\ y\ in\ \langle f, g\rangle{:}$ $(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$
52	$let, 44, 51$	$x{:}qbit, y{:}qbit \rhd let\ f = \mathbf{BellMeasure}\ x\ in\ let\ g = \mathbf{U}\ y$ $in\langle f, g\rangle){:}(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$
53	$let, 41, 52$	$\rhd\ let\ \langle x, y\rangle = \mathbf{EPR}\, *\ in\ let\ f = \mathbf{BellMeasure}\ x$ $in\ let\ g = \mathbf{U}\ y\ in\ \langle f, g\rangle)){:}$ $(qbit \multimap bit \otimes bit) \otimes (bit \otimes bit \multimap qbit)$

A.2. *Example: Reduction of the teleportation term*

As an illustration of the reduction rules of the quantum lambda calculus we show the detailed reduction of the term from the teleportation example from Section 4.3. The reduction of the teleportation term corresponds to the equality $g \circ f = id$. We use the following abbreviations:

$$\begin{array}{lcl} M_{p,p'} & := & \mathit{let}\ f = \mathbf{BellMeasure}\, p\ \mathit{in}\ \mathit{let}\ g = \mathbf{U}\, p'\ \mathit{in}\ g(f\ p_0) \\ B_{p_1} & := & \lambda q_1.(\mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle q_1,p_1\rangle\ \mathit{in}\ \langle\, \mathit{meas}(Hp), \mathit{meas}\, p'\rangle) \\ U_{p_2} & := & \lambda\langle x,y\rangle.\ (\mathit{if}\ x\ \mathit{then}\ (\mathit{if}\ y\ \mathit{then}\ U_{11}p_2\ \mathit{else}\ U_{10}p_2) \\ & & \qquad\qquad \mathit{else}\ (\mathit{if}\ y\ \mathit{then}\ U_{01}p_2\ \mathit{else}\ U_{00}p_2)) \end{array}$$

The reduction of the term is then as follows:

$$\begin{array}{ll}
& \left[\alpha|0\rangle + \beta|1\rangle,\ \begin{array}{l} \mathit{let}\ \langle p,p'\rangle = \mathbf{EPR}\,* \\ \quad \mathit{in}\ \mathit{let}\ f = \mathbf{BellMeasure}\, p \\ \quad \mathit{in}\ \mathit{let}\ g = \mathbf{U}\, p' \\ \quad \mathit{in}\ g(f\ p_0) \end{array}\right] \\
\to_1 & [\alpha|0\rangle + \beta|1\rangle, \mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle H(\mathit{new}\,0), \mathit{new}\,0\rangle\ \mathit{in}\ M_{p,p'}] \\
\to_1 & [(\alpha|0\rangle + \beta|1\rangle)\otimes|0\rangle, \mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle Hp_1, \mathit{new}\,0\rangle\ \mathit{in}\ M_{p,p'}] \\
\to_1 & \left[(\alpha|0\rangle + \beta|1\rangle)\otimes\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle), \mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle p_1, \mathit{new}\,0\rangle\ \mathit{in}\ M_{p,p'}\right] \\
\to_1 & \left[(\alpha|0\rangle + \beta|1\rangle)\otimes\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\otimes|0\rangle, \mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle p_1, p_2\rangle\ \mathit{in}\ M_{p,p'}\right] \\
\to_1 & \left[(\alpha|0\rangle + \beta|1\rangle)\otimes\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle), \mathit{let}\ \langle p,p'\rangle = \langle p_1, p_2\rangle\ \mathit{in}\ M_{p,p'}\right] \\
\to_1 & \left[(\alpha|0\rangle + \beta|1\rangle)\otimes\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle),\ \begin{array}{l} \mathit{let}\ f = \mathbf{BellMeasure}\, p_1 \\ \quad \mathit{in}\ \mathit{let}\ g = \mathbf{U}\, p_2 \\ \quad \mathit{in}\ g(f\ p_0) \end{array}\right] \\
\to_1{}^{*} & \left[(\alpha|0\rangle + \beta|1\rangle)\otimes\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle), U_{p_2}(B_{p_1}p_0)\right] \\
\to_1 & \left[(\alpha|0\rangle + \beta|1\rangle)\otimes\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle), U_{p_2}\begin{pmatrix} \mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle p_0,p_1\rangle \\ \quad \mathit{in}\ \langle\, \mathit{meas}(Hp), \mathit{meas}\, p'\rangle \end{pmatrix}\right] \\
\to_1 & \left[\frac{1}{\sqrt{2}}\begin{pmatrix} \alpha|000\rangle + \alpha|011\rangle \\ +\beta|110\rangle + \beta|101\rangle \end{pmatrix}, U_{p_2}\begin{pmatrix} \mathit{let}\ \langle p,p'\rangle = \langle p_0,p_1\rangle \\ \quad \mathit{in}\ \langle\, \mathit{meas}(Hp), \mathit{meas}\, p'\rangle \end{pmatrix}\right] \\
\to_1 & \left[\frac{1}{\sqrt{2}}\begin{pmatrix} \alpha|000\rangle + \alpha|011\rangle \\ +\beta|110\rangle + \beta|101\rangle \end{pmatrix}, U_{p_2}\langle\, \mathit{meas}(Hp_0), \mathit{meas}\, p_1\rangle\right] \\
\to_1 & \left[\frac{1}{2}\begin{pmatrix} \alpha|000\rangle + \alpha|011\rangle \\ +\alpha|100\rangle + \alpha|111\rangle \\ +\beta|010\rangle + \beta|001\rangle \\ -\beta|110\rangle - \beta|101\rangle \end{pmatrix}, U_{p_2}\langle\, \mathit{meas}\, p_0, \mathit{meas}\, p_1\rangle\right]
\end{array}$$

$$\begin{cases} \xrightarrow{\frac{1}{2}} \left[\frac{1}{\sqrt{2}}\begin{pmatrix} \alpha|000\rangle + \alpha|011\rangle \\ +\beta|010\rangle + \beta|001\rangle \end{pmatrix}, U_{p_2}\langle 0, \mathit{meas}\, p_1\rangle\right] \\ \xrightarrow{\frac{1}{2}} \left[\frac{1}{\sqrt{2}}\begin{pmatrix} \alpha|100\rangle + \alpha|111\rangle \\ -\beta|110\rangle - \beta|101\rangle \end{pmatrix}, U_{p_2}\langle 1, \mathit{meas}\, p_1\rangle\right] \end{cases}$$

$$\begin{cases} \xrightarrow{1/2} \left[\left(\alpha|000\rangle + \beta|001\rangle\right), U_{p_2}\langle 0,0\rangle\right] & \to_1{}^{*} & \left[\left(\alpha|000\rangle + \beta|001\rangle\right), U_{00}p_2\right] \\ \xrightarrow{1/2} \left[\left(\alpha|011\rangle + \beta|010\rangle\right), U_{p_2}\langle 0,1\rangle\right] & \to_1{}^{*} & \left[\left(\alpha|011\rangle + \beta|010\rangle\right), U_{01}p_2\right] \\ \xrightarrow{1/2} \left[\left(\alpha|100\rangle - \beta|101\rangle\right), U_{p_2}\langle 1,0\rangle\right] & \to_1{}^{*} & \left[\left(\alpha|100\rangle - \beta|101\rangle\right), U_{10}p_2\right] \\ \xrightarrow{1/2} \left[\left(\alpha|111\rangle - \beta|110\rangle\right), U_{p_2}\langle 1,1\rangle\right] & \to_1{}^{*} & \left[\left(\alpha|111\rangle - \beta|110\rangle\right), U_{11}p_2\right] \end{cases}$$

$$\begin{cases} \to_1 & [\ (\alpha|000\rangle + \beta|001\rangle), p_2\ ] = & [\ |00\rangle\otimes(\alpha|0\rangle + \beta|1\rangle), p_2\ ] \\ \to_1 & [\ (\alpha|010\rangle + \beta|011\rangle), p_2\ ] = & [\ |01\rangle\otimes(\alpha|0\rangle + \beta|1\rangle), p_2\ ] \\ \to_1 & [\ (\alpha|100\rangle + \beta|101\rangle), p_2\ ] = & [\ |10\rangle\otimes(\alpha|0\rangle + \beta|1\rangle), p_2\ ] \\ \to_1 & [\ (\alpha|110\rangle + \beta|111\rangle), p_2\ ] = & [\ |11\rangle\otimes(\alpha|0\rangle + \beta|1\rangle), p_2\ ] \end{cases}$$

A.3. *Example: Reduction of the superdense coding term*

As another example of the reduction rules, we give the reduction of the superdense coding example from Section 4.3. This reduction shows the equality $f \circ g = id$. Of the four possible cases, we only give one case, namely $(f \circ g)\langle 0,1\rangle = \langle 0,1\rangle$; the remaining cases are similar. We use the same abbreviations as above.

$$\begin{array}{ll}
& \left[|\rangle,\ \begin{array}{l} \mathit{let}\ \langle p,p'\rangle = \mathbf{EPR}\,* \\ \quad \mathit{in}\ \mathit{let}\ f = \mathbf{BellMeasure}\, p \\ \quad \mathit{in}\ \mathit{let}\ g = \mathbf{U}\, p' \\ \quad \mathit{in}\ f(g\langle 0,1\rangle) \end{array}\right] \\
\to_1{}^{*} & \left[\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle),\ \begin{array}{l} \mathit{let}\ f = \mathbf{BellMeasure}\, p_0 \\ \quad \mathit{in}\ \mathit{let}\ g = \mathbf{U}\, p_1 \\ \quad \mathit{in}\ f(g\langle 0,1\rangle) \end{array}\right] \\
\to_1{}^{*} & \left[\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle), B_{p_0}(U_{p_1}\langle 0,1\rangle)\right] \\
\to_1{}^{*} & \left[\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle), B_{p_0}(U_{01}p_1)\right] \\
\to_1 & \left[\frac{1}{\sqrt{2}}(|01\rangle+|10\rangle), B_{p_0}p_1\right] \\
\to_1 & \left[\frac{1}{\sqrt{2}}(|01\rangle+|10\rangle), \mathit{let}\ \langle p,p'\rangle = \mathit{CNOT}\langle p_1,p_0\rangle\ \mathit{in}\ \langle\, \mathit{meas}(Hp), \mathit{meas}\, p'\rangle\right] \\
\to_1 & \left[\frac{1}{\sqrt{2}}(|11\rangle+|10\rangle), \mathit{let}\ \langle p,p'\rangle = \langle p_1,p_0\rangle\ \mathit{in}\ \langle\, \mathit{meas}(Hp), \mathit{meas}\, p'\rangle\right] \\
\to_1 & \left[\frac{1}{\sqrt{2}}(|11\rangle+|10\rangle), \langle\, \mathit{meas}(Hp_1), \mathit{meas}\, p_0\rangle\right] \\
\to_1 & [|10\rangle, \langle\, \mathit{meas}\, p_1, \mathit{meas}\, p_0\rangle] \\
\to_1{}^{*} & [|10\rangle, \langle 0,1\rangle]
\end{array}$$

References

T. Altenkirch and J. Grattage. A functional quantum programming language. Available from arXiv:quant-ph/0409065, 2004.

H. P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundation of Mathematics*. North Holland, second edition, 1984.

P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22:563–591, 1980.

S. Bettelli, T. Calarco, and L. Serafini. Toward an architecture for quantum programming. *The European Physical Journal D*, 25(2):181–200, August 2003.

V. Danos, J.-B. Joinet, and H. Schellinx. On the linear decoration of intuitionistic derivations. *Archive for Mathematical Logic*, 33:387–412, 1995.

D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 400(1818):97–117, July 1985.

J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

E. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.

M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

J. Preskill. Lecture notes for Physics 229, Quantum Computation. Available from http://www.theory.caltech.edu/people/preskill/ph229/#lecture, 1999.

J. W. Sanders and P. Zuliani. Quantum programming. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction: 5th International Conference*, volume 1837 of *Lecture Notes in Computer Science*, pages 80–99, Ponte de Lima, Portugal, July 2000. Springer-Verlag.

P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

P. W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. Institute of Electrical and Electronic Engineers Computer Society Press, November 1994.

B. Valiron. A functional programming language for quantum computation with classical control. Master's thesis, University of Ottawa, September 2004.

A. van Tonder. A lambda calculus for quantum computation. *SIAM Journal of Computing*, 33(5):1109–1135, 2004. Available from arXiv:quant-ph/0307150.