Quantum programming made easy

Luca Paolini
Dipartimento di Informatica
Università di Torino
Italy
luca.paolini@unito.it

Luca Roversi
Dipartimento di Informatica
Università di Torino
Italy
luca.roversi@unito.it

Margherita Zorzi
Dipartimento di Informatica
Università di Verona
Italy
margherita.zorzi@univr.it

We introduce the functional language IQu which, under the paradigm "quantum data & classical control" and in accordance with the model QRAM, allows to define and manipulate quantum circuits and quantum states on which we can execute partial measurement. IQu tailors a lot of ideas from the design of Idealized Algol (roughly, PCF extended with local stores and assignment) and its side-effect management. These ideas play a crucial role in the language design: each quantum co-processor is formalized by means of a quantum register (storing a quantum state) that can be modified by quantum directives (lists of unitary gates). The linearity of quantum states is assured by a one-to-one correspondence between quantum states and quantum registers. We adapt the type system of Idealized Algol for typing both quantum-registers and quantum-directives. The types for quantum-registers are parametric on the number of qubits and their linearity is granted for free. IQu operates on quantum circuits as they were classical data so no restriction exists on their duplication.

1 Introduction

Linearity is an essential ingredient for quantum computing, since quantum data have to undergo restrictions such as non-cloning and non-erasing properties. This is evident from the care that quantum programming languages design puts on the management of quantum bits, especially in presence of higher-order features. Selinger's QPL [18] is a milestone in the development of programming quantum languages. It is the first accrued investigation about the design of quantum programming language in the mainstream "*quantum data & classical control* ". A programmer instructs a classical machine to generate "directives" for a hypothetical quantum device. This latter is thought of to apply the directives to quantum data and to work like a Quantum Random Access Machine (QRAM) [5] which is the low level computational model of reference for *quantum data & classical control* .

The introduction of QPL suggested the design of several quantum programming languages. Some of them are in [2, 1, 12, 22, 19, 20, **?**]. Pagani [12] and Zorzi [22] are more focused on the computational models behind the language, while other papers as [19] are more focused on pioneering prototypes of effective quantum programming languages. A very pragmatic proposal is `Quipper` [21]. Its mixed procedural and declarative approach allows to design, manipulate and evaluate circuits. `Quipper` has `ProtoQuipper` as its a core whose proof-theoretical properties and relationship with the λ-calculus are dealt with in [17, 9]. The languages in [19, 9, 12, 22] follow the direction suggested by Selinger in [18] for dealing with higher-order functions: exponential modalities are used to devise a suitable typing system for quantum data accommodating quantum state in classic programming languages.

Linear Logic-based type systems are indubitably useful and suitable to manage duplication and, consequently, quantum data. Notwithstanding, other solutions are possible, as shown in recent investigations [15, 14]. These solutions come up when one moves the focus from the data-perspective to the control perspective, i.e. from quantum data to classical control.

Submitted to:
Linearity-TLLA 2018

`QWire` [15] is proposed as a sort of extension of a classical language (Haskell or Coq have been considered as hosts) that provides an elegant and manageable language for the definition of quantum circuits. Roughly, `QWire` is a ”quantum plugin” for a host classical language. It provides a suitable meta-programming support to the management of quantum directives through a suitable boxing interface. This interface delimits the quantum typing and decouples it from the typing system of the host language. Further, `QWire` promotes the use of dependent types to improve its approach to quantum programming.

Dependent types are concretely used by qPCF [14, 16] which is an extension of PCF. qPCF is conceived to interact with a restricted QRAM model with relaxed hardware requirements. In particular, qPCF simplifies the interaction between PCF and the quantum co-processor by forbidding both any permanent storing of quantum states and any partial intermediate measure on quantum states.

In this paper, we introduce IQu (read “*Haiku”* as the japanese poetic form). It is higher-order functional programming language that strongly relies on Idealized Algol, which, roughly, is PCF extended with local stores and assignments. The philosophy for defining IQu is to keep programming simple. We show how a minimal enrichment of Idealized ALGOL provides all features needed to address higher-order quantum programming with a style that keeps programmers in a classical programming environment. This simplifies the programming of known quantum algorithms. The main differences of IQu as compared to the existing quantum programming languages follow.

- IQu *avoids the need for explicit linear types*. Linear Logic exponential modalities do not occur in the types of IQu. It turns out that IQu manages linear resources in a way which drastically differs from the Linear Logic-oriented approaches of [19, 6, 7, 22, 9].
- IQu *relies on quantum registers*. Classical control is decoupled from quantum computation by means of the type of quantum registers. We think that it will be interesting to explore variants of these types via linear-logic modalities or dependent types to improve their static analysis potential.
- IQu *provides a classical representation of quantum states in a classic programming setting*. Registers model quantum co-processors which store permanently quantum states, letting partial or general measurements available in the course of a computation. This improves [14] and makes IQu fully expressive w.r.t. all known quantum programming languages. Having based IQu on Idealized ALGOL, its operational semantics internalizes the manipulation of quantum states by generalizing the traditional approach of [19, 6, 22].

2 IQu: Idealized QUantum language

IQu encompasses the essence of Idealized Algol [11], namely an imperative extension of PCF that includes assignments and side-effects. IQu is a prototypical and minimal typed language that combines quantum states and commands with higher-order functional features by using registers.

The *ground types* are $\beta ::= \mathsf{Nat} \mid \mathsf{circ} \mid \mathsf{qCom} \mid \overset{E}{\mathsf{qReg}}$ such that:

- Nat is the type of numerical expressions which evaluate to natural numbers.
- circ is the type of quantum-circuit expressions, i.e. expressions evaluating to strings that describe quantum gates whose arguments are quantum states.
- qCom is the type of commands. The typical use of commands is to apply operations to quantum states being stored in quantum registers. So, commands can produce state modifications, i.e. side-effects.

- $\overset{E}{\mathsf{qReg}}$ is the type of a quantum-register that stores a quantum state and the evaluation of the expression E provides the number of qubits available in the register. We can look at registers as co-processors that permanently store a quantum state, i.e. states are not subject to any decoherence between register commands.

  The expression E labeling $\overset{E}{\mathsf{qReg}}$ ranges over numeric expressions and it can includes specific kind of variables $\varkappa, \varkappa_i, \varkappa^i, \ldots$ take values in $\mathbb{N}$. We do not assume E to can involve all expressions of IQu, we limit ourselves to consider total expressions (that can be evaluated in finite time). The expression E endows IQu with a type polymorphism that, let the quantum algorithms be parametric in the number of qbits they use as input and output. This approach is essentially inspired by elementary form of depend types (e.g. see [14]).

The *general types* are in the language of the grammar $\sigma, \tau, \theta ::= \beta \mid \theta \to \theta$. If $\overset{E}{\mathsf{qReg}}$ occurs as a subtype of θ and $\varkappa$ occurs in E then θ is *open*, by definition. Otherwise θ is *closed.* The *terms* of IQu are in the language of the grammar:

$$\begin{array}{rcl} \mathtt{M},\mathtt{N},\mathtt{P},\mathtt{Q} & ::= & \mathtt{x} \mid \underline{\mathtt{n}} \mid \mathtt{pred} \mid \mathtt{succ} \mid \mathtt{if} \mid \mu\mathtt{x}.\mathtt{M} \mid \lambda\mathtt{x}.\mathtt{M} \mid \mathtt{MN} \\ & \mid & \mathtt{skip} \mid \mathtt{M};\mathtt{N} \mid \mathtt{while}\,\mathtt{P}\;\mathtt{do}\;\mathtt{Q} \mid \overset{E}{\mathsf{qNew}}\;\mathtt{x}\;\mathtt{in}\;\mathtt{N} \mid \mathtt{M} \lhd \mathtt{N} \mid \not-^{\mathtt{N}}\,\mathtt{M} \\ & \mid & \mathtt{U}^{\underline{\mathtt{k}}} \mid \;::\; \mid \;\|\; \mid \mathtt{get} \mid r\mathtt{size} \end{array}$$

- The first line let the boolean-free (call-by-name) version of PCF [3] be part of IQu.
- In analogy with Idealized Algol [11], the second line adds imperative aspects to PCF, adapting them to our quantum setting. They are:
  - The "do-nothing" instruction is $\mathtt{skip}$, the sequential composition of instructions is $\mathtt{M};\mathtt{N}$, the iteration is $\mathtt{while}\,\mathtt{P}\;\mathtt{do}\;\mathtt{Q}$ which is syntactic sugar for the μ-recursion. If $\mathtt{x}$ is a variable of type $\overset{E}{\mathsf{qReg}}$, then $\overset{E}{\mathsf{qNew}}\;\mathtt{x}\;\mathtt{in}\;\mathtt{N}$ is the binder of $\mathtt{x}$ in N, which restricts the use of $\mathtt{x}$ to $\mathtt{N}$.
  - If $\mathtt{N}$ is a circuit expression and the circuit $\mathtt{C}$ is its evaluation, then $\mathtt{M} \lhd \mathtt{N}$ applies $\mathtt{C}$ to the state in the register $\mathtt{M}$.
  - If $\mathtt{N}$ is a number expression with value $\underline{\mathtt{k}}$ and $\mathtt{M}$ is typed $\overset{n}{\mathsf{qReg}}$, i.e. it denotes a n-qubits register, then $\not-^{\mathtt{N}}\,\mathtt{M}$ denotes the measurement of the first $n\%k$-qubits in $\mathtt{M}$ (where $\%$ denote the modulo operation).
- The third line adds gate-names (ranged over $\mathtt{U}$ and labeled by their arity), the sequential and parallel composition of circuits, the (syntactic-sugar) operator that extracts a bit from a number and the operator which returns the size of a register. These operations are typical of languages focusing on quantum directives as [14].

2.1 Typing system

A *base* is a finite list $\mathtt{x}_1 : \sigma_1, \ldots, \mathtt{x}_n : \sigma_n$ that we manage as a set such that $\mathtt{x}_i \neq \mathtt{x}_j$ for every $i \neq j$. If $B = \mathtt{x}_1 : \sigma_1, \ldots, \mathtt{x}_n : \sigma_n$, then $\mathrm{dom}(B) = \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$ and $\mathrm{ran}(B) = \{\sigma_1, \ldots, \sigma_n\}$. The extension of B with $\mathtt{x} : \sigma$ is $B \cup \{\mathtt{x} : \sigma\}$ where without loss of generality we can assume $\mathtt{x} \notin \mathrm{dom}(B)$.

Definition 1. *A term of* IQu *is* well-typed *if and only if it is the conclusion of a finite derivation built with the rules in Table 1. A well-typed term is* open *if either it contains a free variable or if its type depends on* $\overset{E}{\mathsf{qReg}}$ *where E is open. Otherwise, the term is* closed. *Table 1 presents only the rules concerning the quantum fragment of* IQu*. See [13] for the complete set of rules.* □

$$\frac{B \vdash \mathtt{P} : \mathsf{qCom} \quad B \vdash \mathtt{Q} : \beta \quad \beta \in \{\mathsf{Nat}, \mathsf{circ}, \mathsf{qCom}\}}{B \vdash \mathtt{P};\mathtt{Q} : \beta}\,(tc) \qquad \frac{B \vdash \mathtt{M} : \overset{E}{\mathsf{qReg}} \quad B \vdash \mathtt{N} : \mathsf{circ}}{B \vdash \mathtt{M} \lhd \mathtt{N} : \mathsf{qCom}}\,(tA)$$

$$\frac{B \cup \{\mathtt{x} : \overset{E}{\mathsf{qReg}}\} \vdash \mathtt{N} : \beta \quad \beta \in \{\mathsf{Nat}, \mathsf{circ}, \mathsf{qCom}\}}{B \vdash \overset{E}{\mathsf{qNew}}\ \mathtt{x}\ \mathtt{in}\ \mathtt{N} : \beta}\,(tnew) \qquad \frac{B \vdash \mathtt{P} : \overset{E}{\mathsf{qReg}} \quad B \vdash \mathtt{Q} : \mathsf{Nat}}{B \vdash \not\frown^{\mathtt{Q}}\mathtt{P} : \mathsf{Nat}}\,(tpM)$$

$$\frac{\mathtt{U}^{\underline{\mathtt{k}}} \in \mathscr{U}}{B \vdash \mathtt{U}^{\underline{\mathtt{k}}} : \mathsf{circ}}\,(tc_1) \qquad \frac{}{B \vdash :: : \mathsf{circ} \to \mathsf{circ} \to \mathsf{circ}}\,(tc_2) \qquad \frac{}{B \vdash \| : \mathsf{circ} \to \mathsf{circ} \to \mathsf{circ}}\,(tc_3)$$

$$\frac{B \vdash \mathtt{M} : \overset{E}{\mathsf{qReg}}}{B \vdash r\mathtt{size}(\mathtt{M}) : \mathsf{Nat}}\,(ts_0) \qquad \frac{}{B \vdash \mathsf{get} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}}\,(tg)$$

Table 1: Typing Rules.

An open term becomes closed after we instantiate all its variables $\varkappa, \varkappa', \ldots$ and we substitute all its free term-variables by closed well typed terms. It is worth to remark that an open term with variables in its type stands for a family of IQu programs. Since renaming of $\varkappa$ and $\varkappa'$ by, say, $\varkappa''$ in a term is possible the effect is to restrict the family of programs that the term represents. Moreover, the polymorphism induced by $\varkappa, \varkappa', \ldots$ can be straightforwardly (albeit not trivially) adapted to an extension based on explicit dependent types, following [14]. Last, we remark that $\varkappa$ is never part of the domain of our bases B (all such variables are typed implicitly Nat).

Some comments on the rules are worth doing.

The rule (tA) types the application of a circuit, i.e. of a sequence of gates, to a quantum-register. The rule $(tnew)$ declares a local register, i.e. it hides a register whose type is in the given base, like in Idealized Algol. The rule (tpM) gives type to a partial measurement executed on the register $\mathtt{P}$. I.e. if $\mathtt{Q}$ evaluates to k and E is n, then $n\%k$ qubits are measured and the resulting state is left in the register. Moreover, the result of the measurement is a number carrying the measure-information in its binary representation. The operator $r\mathtt{size}$ returns the number of qubits stored in a register by extracting them from its type.

Basic properties, among which a Generation Lemma, hold on the type system [13].

Example 1. *Let* $\mathtt{N} : \mathsf{Nat}$ *be a term whose unique free variable is* $\mathtt{r} : \overset{3}{\mathsf{qReg}}$. *Also, let* $\mathtt{Not}$ *be the not-gate (a.k.a. Pauli-X gate) and* $\mathtt{Id}$ *be the identity gate, both with arity* 1. *Let* $\mathtt{M}$ *to denote* $(\mathtt{r} \lhd (\mathtt{Not} \parallel \mathtt{Id} \parallel \mathtt{Not}));\mathtt{N}$ *(anticipating the semantics, it would initialise* $\mathtt{r}$ *with* $|101\rangle$*). The type of* $\mathtt{M}$ *can be* Nat *by means of* (tA). *So,* Nat *can be the type of* $\overset{3}{\mathsf{qNew}}\ \mathtt{r}\ \mathtt{in}\ \mathtt{M}$ *using* $(tnew)$. □

2.2 Evaluation Semantics

We focus on the evaluation of IQu programs, i.e. closed terms typed with a ground type. Notice that IQu is endowed with an infinite set of ground types: $\mathsf{Nat}, \mathsf{circ}, \mathsf{qCom}, \overset{0}{\mathsf{qReg}}, \overset{1}{\mathsf{qReg}}, \ldots$.

Following Idealized Algol, the operational semantics of IQu relies on a *store*. A store $\mathbb{s}$ is a finite set of pairs $\{(r_1, |\phi_1\rangle), \ldots, (r_n, |\phi_n\rangle)\}$ where every r_i is the name of a register and every $|\phi_i\rangle$ is a quantum state that contains n qbits in accordance with the type $\overset{n}{\mathsf{qReg}}$ of r_i. The finite set of names for registers in

$$\frac{\text{(if } B \vdash \mathtt{M} : \overset{n}{\mathsf{qReg}}\text{)}}{\bar{\mathtt{s}}, \mathit{r}\mathtt{size}\,\mathtt{M} \Downarrow_1 \bar{\mathtt{s}}, \underline{\mathtt{n}}}\ (\mathrm{sz}) \qquad \frac{\bar{\mathtt{s}}, \mathtt{M} \Downarrow_{\alpha} \bar{\mathtt{s}}', \underline{\mathtt{m}} \qquad \bar{\mathtt{s}}', \mathtt{N} \Downarrow_{\alpha'} \bar{\mathtt{s}}'', \underline{\mathtt{n}}}{\bar{\mathtt{s}}, \mathtt{get}\,\mathtt{M}\,\mathtt{N} \Downarrow_{\alpha\cdot\alpha'} \bar{\mathtt{s}}'', \lceil \underline{\mathtt{m}} \rceil^{\underline{\mathtt{n}}}}\ (\mathrm{gt})$$

$$\frac{}{\bar{\mathtt{s}}, \mathtt{U}^{\underline{\mathtt{k}}} \Downarrow_1 \bar{\mathtt{s}}, \mathtt{U}^{\underline{\mathtt{k}}}}\ (\mathrm{u}) \qquad \frac{\bar{\mathtt{s}}, \mathtt{M}_0 \Downarrow_{\alpha} \bar{\mathtt{s}}', \mathtt{C}_0 \quad \bar{\mathtt{s}}', \mathtt{M}_1 \Downarrow_{\alpha'} \bar{\mathtt{s}}'', \mathtt{C}_1}{\bar{\mathtt{s}}, \mathtt{M}_0 :: \mathtt{M}_1 \Downarrow_{\alpha\cdot\alpha'} \bar{\mathtt{s}}'', \mathtt{C}_0 :: \mathtt{C}_1}\ (\mathrm{u}')$$

$$\frac{\bar{\mathtt{s}}, \mathtt{M}_0 \Downarrow_{\alpha} \bar{\mathtt{s}}', \mathtt{C}_0 \qquad \bar{\mathtt{s}}', \mathtt{M}_1 \Downarrow_{\alpha'} \bar{\mathtt{s}}'', \mathtt{C}_1}{\bar{\mathtt{s}}, \mathtt{M}_0 \parallel \mathtt{M}_1 \Downarrow_{\alpha\cdot\alpha'} \bar{\mathtt{s}}'', \mathtt{C}_0 \parallel \mathtt{C}_1}\ (\mathrm{u}'')$$

$$\frac{\bar{\mathtt{s}} \cup \{\mathtt{r} := 0\}, \mathtt{M} \Downarrow_{\alpha} \bar{\mathtt{s}}', \mathtt{V}}{\bar{\mathtt{s}}, \overset{n}{\mathsf{qNew}}\ \mathtt{r}\ \mathtt{in}\ \mathtt{M} \Downarrow_{\alpha} \bar{\mathtt{s}}' \upharpoonright_{\mathtt{r}}, \mathtt{V}}\ (qn) \qquad \frac{\bar{\mathtt{s}}, \mathtt{N} \Downarrow_{\alpha} \bar{\mathtt{s}}', \mathtt{C} \qquad \text{(if } B \vdash \mathtt{r} : \overset{n}{\mathsf{qReg}}\text{)}}{\bar{\mathtt{s}}, \mathtt{r} \lhd \mathtt{N} \Downarrow_{\alpha} \bar{\mathtt{s}}'[\mathtt{r} := \mathsf{cEval}^n(\mathtt{C})(\bar{\mathtt{s}}'(\mathtt{r}))], \mathtt{skip}}\ (qa)$$

$$\frac{\bar{\mathtt{s}}, \mathtt{M} \Downarrow_{\alpha} \bar{\mathtt{s}}', \underline{\mathtt{k}} \qquad \bar{\mathtt{s}}', \mathtt{N} \Downarrow_{\alpha'} \bar{\mathtt{s}}'', \mathtt{r} \qquad (m, |\phi\rangle, \alpha'') \in \mathsf{pMea}^n(\bar{\mathtt{s}}''(\mathtt{r}), k) \qquad \text{(if } B \vdash \mathtt{N} : \overset{n}{\mathsf{qReg}}\text{)}}{\bar{\mathtt{s}}, \not\frown^{\mathtt{M}} \mathtt{N} \Downarrow_{\alpha\cdot\alpha'\cdot\alpha''} \bar{\mathtt{s}}''[\mathtt{r} := |\phi\rangle], \underline{\mathtt{m}}}\ (qm)$$

Table 2: Operational Semantics, quantum fragment

a program of IQu is dom($\bar{\mathtt{s}}$). The notation $\bar{\mathtt{s}}[\mathtt{x} := |\phi\rangle]$ builds a new store which behaves like $\bar{\mathtt{s}}$ everywhere except on $\mathtt{x}$; the new store associates the state $|\phi\rangle$ to the register $\mathtt{x}$. As a notation, $\mathtt{C}$ ranges over the strings that describe circuits, i.e. parallel and series composition of names for gates. Moreover, we can use $\mathtt{V}$ to range over numerals, strings that describe circuits, register names and `skip`.

Definition 2. *The evaluation semantics of IQu is a formal statement of the shape* $\bar{\mathtt{s}}, \mathtt{M} \Downarrow_{\alpha} \bar{\mathtt{s}}', \mathtt{V}$ *obtained as conclusion of a derivation built with the rules in Table 2, such that:* $\mathtt{M}$ *is a term whose typing judgment is* $\mathtt{x}_1 : \overset{n_1}{qReg}, \dots, \mathtt{x}_k : \overset{n_k}{qReg} \vdash M : \beta$*;* $\bar{\mathtt{s}}$ *is a store such that* $\{\mathtt{x}_1, \dots, \mathtt{x}_k\} \subseteq dom(\bar{\mathtt{s}})$*;* $0 < \alpha \leq 1$ *is the probability that, from the store* $\bar{\mathtt{s}}$*, the term* $\mathtt{M}$ *yields* $\mathtt{V}$ *and the store* $\bar{\mathtt{s}}'$*. As for the type system, in Table 2 we report only evaluation rules concerning the quantum fragment of the language. We assume the reamining part of the language evaluated iin accord with the standard call-by-name evaluation of PCF.* □

All the rules in Table 2, but (qa) and (qm), preserve the store that their judgments take as input. The rule (sz) returns the number of qbits of a register, reading that value from its type. The rule (gt) allows to get the $\underline{\mathtt{n}}$-th bit of $\underline{\mathtt{m}}$ resulting from $\mathtt{M}$. The rules $(\mathrm{u}), (\mathrm{u}'), (\mathrm{u}'')$ evaluate circuit expressions, i.e. strings we can build by series and parallel compositions of gate-names. It is worth to notice that term typed circuits have to be evaluated to become proper circuit that can be supplied on a quantum register. For instance, $(\lambda \mathtt{x}^{\mathsf{Nat}}.\mathtt{if}\,\mathtt{x}\,\mathtt{M}_0\,\mathtt{M}_1)\mathtt{N}$ has type circ whenever $\mathtt{M}_0, \mathtt{M}_1 : \mathsf{circ}$ and $\mathtt{N} : \mathsf{Nat}$, but it cannot used as a quantum transformation until its evaluation in a proper circuit is completed (note that the evaluation of $\mathtt{M}_0, \mathtt{M}_1$ can loop forever). Moreover, $\mathtt{M}_i : \mathsf{circ}$ can have shape $\mathtt{M}_i';\mathtt{M}_i'' : \mathsf{circ}$ $(i = 0, 1)$ where $\mathtt{M}_0', \mathtt{M}_0' : \mathsf{qCom}$ $(i = 0, 1)$ can apply some quantum transformations to registers producing side-effects. The evaluation of circuits is done in sequentially, also in presence of side-effect and when the generated circuit produce the parallel of two sub-circuits (c.f. rule (u'')).

The rules $(qn), (qa), (qm)$ are specific to IQu. A programmer can ask a new quantum register for manipulating a quantum by means of (qn) at run-time. We notice that no limit exists on the number of quantum registers that a program in IQu manipulates. The programmer is in charge to properly fix that number. In analogy with a standard management of computational resources, the lack of a resource needs to throw an exception. The rule (qa) interprets $\lhd$ as a sort of assignment which modifies a state by means of a circuit application. Te rule (qm) interprets $\not\frown$ as a sort of assignment which modifies a state by means of a measurement. Its result is the outcome of the corresponding (partial) observation. By the way, $\lhd$ and $\not\frown$ can occur hidden everywhere, for example also in the expression we need to evaluate for choosing which branch of a conditional to follow.

The function cEval^n occurs in (qa). It takes a circuit as its input for giving the corresponding unitary operator as output. Specifically, $\mathsf{cEval}^n : \textsc{Circ} \to \mathscr{H}^{2^n} \to \mathscr{H}^{2^n}$ is:

$$\mathsf{cEval}^n(x) = \begin{cases} \mathbf{Id}^{\underline{\mathtt{n}}} & x = \mathtt{U}^{\underline{\mathtt{k}}} \text{ and } n < k \\ \mathbf{U}^{\underline{\mathtt{k}}} & x = \mathtt{U}^{\underline{\mathtt{k}}} \text{ and } n = k \\ \mathbf{U}^{\underline{\mathtt{k}}} \otimes \mathbf{Id}^{\underline{\mathtt{n-k}}} & x = \mathtt{U}^{\underline{\mathtt{k}}} \text{ and } k \leq n \\ \mathsf{cEval}(\mathtt{C}_0) \otimes \mathsf{cEval}(\mathtt{C}_1) & x = \mathtt{C}_0 \parallel \mathtt{C}_1 \\ \mathsf{cEval}(\mathtt{C}_0) \circ \mathsf{cEval}(\mathtt{C}_1) & x = \mathtt{C}_0 :: \mathtt{C}_1 \end{cases}$$

which says that cEval^n relies on a family of functions on a n-dimension Hilbert space.

The relation pMea occurs in (qm). Following [4], $\mathsf{pMea}^n : \mathscr{H}^n \times \mathbb{N} \longrightarrow \wp\big(\mathbb{N} \times \mathscr{H}^n \times \mathbb{R}\big)$ formalizes a quantum measurement. Let us assume that $k \in \mathbb{N}$, $N = 2^n$, $K = 2^{k\%n}$ and that $j \cdot h$ denotes the number we obtain by juxtaposing the binary representations of j and h. Then:

$$\mathsf{pMea}^n(|\phi\rangle, k) = \left\{ (m, |\psi_m\rangle, p_m) \;\middle|\; \begin{array}{l} \sum_{j<K} \sum_{h<N-K} c_{j\cdot h} |j^\flat\rangle \otimes |h^\flat\rangle \text{ and,} \\ m \leq K \text{ s.t. } |\psi_m\rangle = \sum_{h<N-K} \frac{c_{m\cdot h}}{\sqrt{p_m}} |m^\flat\rangle \otimes |h^\flat\rangle \\ \qquad\qquad\qquad \text{where } p_m = \sum_{h<N-K} |c_{j\cdot h}|^2 \end{array} \right\}$$

where $|x^\flat\rangle$ represents the binary encoding of x. The first argument of pMea is a quantum state $|\phi\rangle$ of dimension N. The second argument is $k \in \mathbb{N}$, the number of qubits to measure, modulo n. The result of $\mathsf{pMea}^n(|\phi\rangle, k)$ is a set of triples. The first component of the triple is a partial measure executed on $|\phi\rangle$: its value $m \in \mathbb{N}$ is the (deterministic measure) of a sub-state of dimension $2^{k\%n}$. The second component is the (sometimes called) collapsing state which has dimension 2^n and it is obtained from $|\phi\rangle$ by collapsing to m its measured sub-state. The third component is the probability of measuring the value m.

We conclude by observing that (qa) is deterministic while (qm) has both non-deterministic and probabilistic nature. We mean that (qa) only modifies $\mathtt{r}$ in $\overline{\mathtt{s}}$ if $\mathtt{N}$ converges to $\mathtt{C}$. Instead, (qm) yields any of the possible measures on a state.

IQu enjoys standard properties such as Preservation and Progress [13].

Theorem 1 (Preservation)**.** *If* $\mathtt{M}$ *is a closed term such that* $\vdash \mathtt{M} : \beta$ *and* $\mathtt{M} \Downarrow \mathtt{N}$ *then* $\mathtt{N}$ *is a closed term such that* $\vdash \mathtt{N} : \beta$*.*

Theorem 2 (Progress)**.** *If* $\mathtt{M}$ *is a closed term such that* $\vdash \mathtt{M} : \beta$ *and* $\mathtt{M} \Downarrow \mathtt{N}$ *then N is either a numeral, a string representing a circuit,* `skip` *or a register.*

2.3 Examples

Let us provide two example of IQu programming.

Example 2 (Bell state circuit)**.** *The Bell states (or EPR states or EPR pairs) are the simplest examples of entanglement of quantum states [10]. The circuit in the left of Table 3 applies a Hadamard gate on the top wire followed by a controlled-not. It can be used to generate the Bell states by feeding it by* $|00\rangle$*,* $|01\rangle$*,* $|10\rangle$*,* $|11\rangle$*. For example, the circuit returns the state* $\beta_{00} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ *on input* $|00\rangle$*.*

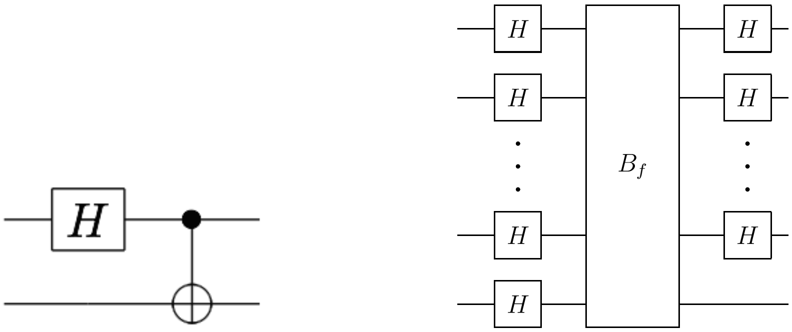

Table 3: Bell State circuit (left) and Deutsch-Jozsa circuit (right).

Let $H : \mathsf{circ}$ *be the Hadamart gate,* $\mathtt{Id} : \mathsf{circ}$ *be identity and* $\mathtt{CNOT} : \mathsf{circ}$ *be the controlled-not. Let* Bell *be the closed term* $(H \,\|\, \mathtt{Id}) :: \mathtt{CNOT}$ *that straigforwardly describes the above circuit. It is easy to see that* $\vdash (H \,\|\, \mathtt{Id}) :: \mathtt{CNOT} : \mathsf{circ}$ *.. We use the closed term* $\overset{2}{\mathsf{qNew}}\, \mathtt{r}\ \mathtt{in}\ (\mathtt{r} \lhd \mathsf{Bell}; \nprec^{1})$ *to simulate an EPR experiment: it requires that a fresh co-processor is made available for the computation of* $\mathtt{r} \lhd \mathsf{Bell}; \nprec^{1}$ *that applies the gates in* Bell *to the state stored in* $\mathtt{r}$ *and then does a measurement. For space reasons, we leave to the reader to check that* $\vdash \overset{2}{\mathsf{qNew}}\, \mathtt{r}\ \mathtt{in}\ (\mathtt{r} \lhd \mathsf{Bell}; \nprec^{1}) : \mathsf{Nat}$ *and* $\{(\mathtt{r}, |00\rangle)\}, \mathtt{r} \lhd \mathsf{Bell} \Downarrow_1 \{(\mathtt{r}, \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle))\}, \mathtt{skip}$*. A possible (probabilistic) conclusion of our EPR experiment is* $\{(\mathtt{r}, \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle))\} \Downarrow_{\frac{1}{2}} \nprec^{1} \{(\mathtt{r}, |00\rangle)\}, \underline{0}$*, because* $\mathsf{pMea}^2(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), 1) = \{(0, |00\rangle, \frac{1}{2}), (1, |11\rangle, \frac{1}{2})\}$.

Example 3 (Deutsch-Jozsa Circuit)**.** *In this example we show how a* IQu *term can represent a whole (infinite) family of quantum programs. We provide the* IQu *encoding of the circuit that implements the Deutsch-Jozsa algorithm [10]. It is a generalization of Deutsch algorithm: given a black-box* B_f *implementing some function* $f : \{0,1\} \to \{0,1\}$*, it determines whether* f *is constant or balanced (a function is balanced if exactly half of the inputs go to* 0 *and, the other half, go to* 1*). Deutsch showed how to achieve this result with a single call of* B_f*, in contrast with the classical case that requires to observe* f *on two inputs. The Deutsch-Josza algorithm solves the parametric problem that considers functions* $f : \{0,1\}^n \to \{0,1\}$ *with* n *inputs.*

The circuit on the right of Table 3 is a quantum solution for the Deutsch-Josza algorithm, where we neglect the final measurement phase. When fed with a classical input state $|\underbrace{0\ldots0}_{n}1\rangle$ *we can do a measurement of the first* n *bits to know if the function* f *is constant or not. If all* n *qubits of our (unique) measurement are 0 then we can conclude that* f *is constant. Otherwise, if at least one of the measurement outcomes is 1, then* f *is balanced. See [10] for further details.*

We denote $\mathtt{H} : \mathsf{circ}$ *the Hadamard gate and* $\mathtt{Id} : \mathsf{circ}$ *Identity gates. We program the circuit in the table by sequentializing the three sub-circuits* M_1*,* $\mathtt{x}$ *and* M_3*, where* $\mathtt{x} : \mathsf{circ}$ *is expected to be substituted by the black-box circuit that implements the function* f*.*

- *Let* M_{par} *be a term that applied to a circuit* $\mathtt{C} : \mathsf{circ}$ *and to a numeral* $\underline{\mathtt{n}}$ *puts in parallel* $n+1$ *copies of* $\mathtt{C}$*. It is defined as follows:* $\mathsf{M}_{par} = \lambda \mathtt{u}^{\mathsf{circ}}.\lambda k^{\mathsf{Nat}}.Y \mathsf{W}_1 \mathtt{u} k : \mathsf{circ} \to \mathsf{Nat} \to \mathsf{circ}$*, where* W_1 *is the term* $\lambda \mathtt{w}^{\sigma}.\lambda \mathtt{u}^{\mathsf{circ}}.\lambda k^{\mathsf{Nat}}.\mathtt{if}\ \mathtt{k}\ (\mathtt{u})\ (\mathtt{u}\ \|\ (\mathtt{wupred}(k)))$ *having type* $\sigma \to \sigma$ *with* $\sigma = \mathsf{circ} \to \mathsf{Nat} \to \mathsf{circ}$*.*
- *Let* M_1 *be* $\mathsf{M}_{par} H\, \mathtt{rsize}(\mathtt{r}) : \mathsf{circ}$ *where r is the co-processor register.*
- *Let* M_3 *be* $(\mathsf{M}_{par} H \mathtt{pred}(\mathtt{rsize}(\mathtt{r}))) \,\|\, \mathtt{Id} : \mathsf{circ}$ *where r is the co-processor register.*

We use register $\mathtt{r} : \overset{n+1}{\mathsf{qReg}}$ *to implement the n-instance of the Deutsch-Jozsa, for an arbitrary n. Since the expected starting state of our Deutsch-Jozsa algorithm is* $|\overbrace{0\ldots0}^{n}1\rangle$*, while* $\overset{n}{\mathsf{qNew}}\,\mathtt{r}$ *creates a quantum register fully initialized to 0, we use an initializing circuit* $\mathsf{M}_{\mathsf{init}} = (\mathsf{M}_{par}\,\mathtt{Id}(\mathtt{pred}(\mathtt{rsize}(r)))) \parallel \mathtt{Not}$ *that complements the last qbit, setting it to* 1*. Summing up, the parametric solution to the Deutsch-Jozsa algorithm can be defined in IQu by* $\lambda x^{\mathsf{circ}}.\overset{n+1}{\mathsf{qNew}}\,r\,\mathtt{in}\,((\mathtt{r} \lhd \mathsf{DJ}^{+}); \not\nearrow^{\underline{\mathtt{n}}}\mathtt{r})$ *where* DJ^{+} *is the circuit* $\mathsf{M}_{\mathsf{init}} :: \mathsf{M}_1 :: x :: \mathsf{M}_3 \ : \mathsf{circ}$*. The program can solve any instance of Deutsch-Jozsa we obtain by fixing the value of its type parameter n. We do not considered binders for variables in types only for sake of simplicity.*

Let M_{B_f} *be a black-box closed circuit implementing the function* f *that we want to check and let* $\mathsf{DJ}^{\star}$ *be* $\mathsf{DJ}^{+}[\mathsf{M}_{B_f}/x]$ *namely the circuit obtained by the substitution of* M_{B_f} *to* x *in* DJ^{+}*. The rule* (qa) *implies that* $\{(\mathtt{r}, |\underbrace{0\ldots0}_{n}\rangle)\}, \mathtt{r} \lhd \mathsf{DJ}^{\star} \Downarrow_1 \{(\mathtt{r}, |\phi\rangle)\}, \mathtt{skip}$ *where* $|\phi\rangle$ *is the computational state after the evaluation of* DJ^{+}*. To measure* $|\phi\rangle$ *we use* $\{\mathtt{r}, |\phi\rangle\}, \not\nearrow^{\underline{\mathtt{n}}}\mathtt{r} \Downarrow_{\alpha} \{\mathtt{r}, |\phi'\rangle\}, \underline{\mathtt{k}}$*, where* $(k, |\phi'\rangle, \alpha) \in \mathsf{pMea}^{n}(\overline{\mathtt{s}}'(r), n)$*, i.e.* $\underline{\mathtt{k}}$ *is one of the possible output of the measurement and* α *is the associated probability.*

3 Conclusions and future work

The language IQu is a higher-order programming language that manage quantum co-processors. We formalize co-processors as quantum registers that store quantum states. This approach is radically new. Its distinctive features are: (i) the linearity of quantum states is granted by the identity of registers (each register is identified linearly by a unique name), (ii) a natural internal approach to many co-processors; and, (iii) the classical fragment of the language is unaffected by the peculiarity of quantum data, so that it can be used in a natural way. Moreover, we carefully isolate the description of directives for quantum co-processors and the description of quantum states stored in quantum registers, because directives can be treated as classical data (duplicable/erasable).

Current ongoing work focuses on its semantics and its typing systems. First, we are adding dependent types for circuits and registers management (in analogy with [15, 14]). Second, we are studying a mature approach to circuits by providing an explicit status and a linear typing to circuit-wires (in analogy with [15]). Third, we are interested in the formalization of a call-by-value version of IQu in order to further ease the embedding of quantum programming in common programming frameworks. Fourth, we are interested in the development of denotational semantics for IQu, maybe a not complete one, but a semantic suitable to tackle the equivalence between programs involving (meaningful) quantum, non-determinism and probabilistic aspects.

References

[1] Pablo Arrighi & Alejandro Díaz-Caro (2012): *Scalar System F for Linear-Algebraic Lambda-Calculus: Towards a Quantum Physical Logic.* Logical Methods in Computer Science 8(1), doi:10.2168/LMCS-8(1:11)2012. Available at `http://dx.doi.org/10.2168/LMCS-8(1:11)2012`.

[2] Jonathan Grattage (2011): *An Overview of QML With a Concrete Implementation in Haskell.* Electr. Notes Theor. Comput. Sci. 270(1), pp. 165–174, doi:10.1016/j.entcs.2011.01.015. Available at `http://dx.doi.org/10.1016/j.entcs.2011.01.015`.

[3] Carl Adam Gunter (1992): *Semantics of Programming Languages. Structures and Techniques.* MIT Press.

[4] Phillip Kaye, Raymond Laflamme & Michele Mosca (2007): *An introduction to quantum computing*. Oxford University Press, Oxford.

[5] E. Knill (1996): *Conventions for quantum pseudocode*. Technical Report, Los Alamos National Laboratory. Technical Report.

[6] Ugo dal Lago, Andrea Masini & Margherita Zorzi (2009): *On a Measurement-free Quantum Lambda Calculus with Classical Control*. Mathematical. Structures in Comp. Sci. 19(2), pp. 297–335, doi:10.1017/S096012950800741X. Available at `http://dx.doi.org/10.1017/S096012950800741X`.

[7] Ugo Dal Lago, Andrea Masini & Margherita Zorzi (2010): *Quantum implicit computational complexity*. Theor. Comput. Sci. 411(2), pp. 377–409, doi:10.1016/j.tcs.2009.07.045. Available at `http://dx.doi.org/10.1016/j.tcs.2009.07.045`.

[8] Mohamed Mahmoud & Amy P Felty (2016): *Formalization of Metatheory of the Quipper Programming Language in a Linear Logic*. University of Ottawa, Canada.

[9] Michael A. Nielsen & Isaac L. Chuang (2000): *Quantum computation and quantum information*. Cambridge University Press, Cambridge.

[10] Peter W. O'Hearn (1997): *Algol-like Languages*, chapter Algol and Functional Programming. Progress in Theoretical Computer Science, Birkhäuser.

[11] Michele Pagani, Peter Selinger & Benoît Valiron (2014): *Applying quantitative semantics to higher-order quantum computing*. In: Proceedings of POPL '14, ACM, pp. 647–658.

[12] Luca Paolini, Luca Roversi & Margherita Zorzi (2017): *Quantum Programming Made Easy*. Technical Report, Universitá di Torino e Verona. Available at `http://lists.seas.upenn.edu/pipermail/types-announce/2016/006371.html`. Https://arxiv.org/abs/1711.00774.

[13] Luca Paolini & Margherita Zorzi (2017): *qPCF: a language for quantum circuit computations*. In: TAMC'17, LNCS 10185, Springer, Germany, pp. 455–469, doi:10.1007/978-3-319-55911-7_33. Available at `http://dx.doi.org/10.1007/978-3-319-55911-7_33`.

[14] Jennifer Paykin, Robert Rand & Steve Zdancewic (2017): *QWIRE: a core language for quantum circuits*. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, pp. 846–858.

[15] R Rand, J. Paykin & S. Zdancewic (2017): *QWIRE Practice: Formal Verification of Quantum Circuits in Coq*. In: Quantum Physic and Logic 2017, Lecture Notes in Computer Science 10185.

[16] Neil Ross (2015): *Algebraic and Logical Methods in Quantum Computation*. Ph.D. thesis, Dalhousie University Halifax, Nova Scotia.

[17] Peter Selinger (2004): *Towards a Quantum Programming Language*. Mathematical Structures in Computer Science 14(4), pp. 527–586, doi:10.1017/S0960129504004256. Available at `http://dx.doi.org/10.1017/S0960129504004256`.

[18] Peter Selinger & Benoit Valiron (2006): *A lambda calculus for quantum computation with classical control*. Mathematical Structures in Computer Science 16, pp. 527–552, doi:10.1017/S0960129506005238. Available at `http://journals.cambridge.org/article_S0960129506005238`.

[19] Peter Selinger & Benoît Valiron (2009): *Semantic Techniques in Quantum Computation*, chapter Quantum lambda calculus, pp. pp. 135–172. Cambridge University Press.

[20] Benoît Valiron, Neil J. Ross, Peter Selinger, D. Scott Alexander & Jonathan M. Smith (2015): *Programming the Quantum Future*. Commun. ACM 58(8), pp. 52–61, doi:10.1145/2699415.

[21] Margherita Zorzi (2016): *On quantum lambda calculi: a foundational perspective*. Mathematical Structures in Computer Science 26(7), pp. 1107–1195, doi:10.1017/S0960129514000425. Available at `http://dx.doi.org/10.1017/S0960129514000425`.