Quantum Programming

J. W. Sanders and P. Zuliani

Programming Research Group,
Oxford University Computing Laboratory,
Oxford, OX1 3QD,
England
{jeff,pz}@comlab.ox.ac.uk

Abstract. In this paper a programming language, *qGCL*, is presented for the expression of quantum algorithms. It contains the features required to program a 'universal' quantum computer (including initialisation and observation), has a formal semantics and body of laws, and provides a refinement calculus supporting the verification and derivation of programs against their specifications. A representative selection of quantum algorithms are expressed in the language and one of them is derived from its specification.

1 Introduction

The purpose of this paper is to present a programming language, *qGCL*, for quantum computation.

Quantum algorithms are usually described in pseudo code. For semantic support there are two models of quantum computation: quantum networks [8, 2] and quantum Turing machines [7]. The former provides a data-flow view and so is relevant when considering implementation in terms of gates; whilst it expresses modularisation well, it fails to express (demonic) nondeterminism or probability (both features of quantum computation). The latter is appropriate for complexity analysis but as inappropriate for modularised description and reasoning about correctness of quantum algorithms as standard Turing machines are for that purpose for standard algorithms.

With *qGCL* we introduce an extension of the guarded-command language to express quantum algorithms. It contains both (demonic) nondeterminism and probability. The former arises in the specification of several quantum algorithms (and so in their derivations) and the latter is required in order to 'observe' a quantum system. *qGCL* has a rigorous semantics and body of laws as a result of other work (on probabilistic semantics; see for example [23]) and so benefits from an associated refinement calculus (exhibiting notions of program refinement, data refinement, containing high-level control structures and combining specification constructs with code). Moreover it abstracts implementation concerns like the representation of assignments as unitary transformations and the execution of those unitary transformations as gates.

After the invention of various efficient quantum algorithms there seems to have been a period of consolidation in which frameworks have been sought to relate those algorithms. The 'hidden subgroup problem' [25] has been seen as a conceptually unifying principle whilst 'multi-particle interference' [6] has been proposed as a unifying principle closer to implementation. More pragmatically, several simulations have been proposed [1, 26, 31] at various levels of applicability.

Not surprisingly we take the formal-methods or 'MPC' view that a derivation is worth a thousand simulations (or more!). Thus in an area containing subtle algorithms formal reasoning can be expected to come into its own. One approach would be to perform derivations of quantum program in a standard model and 'bolt on' reasoning to cover their probabilistic and quantum behaviour. A more elegant alternative would be a single formalism in which all aspects of a quantum program's functionality are reasoned about at once. It might be thought that such a formalism would be unwieldy. From our experience with probabilistic semantics we have found that not to be the case; it has led us to the present proposal.

There have been at least two previous attempts to treat quantum computation from a programming-language perspective: Greg Baker's Q-GOL [1] and Bernhard Ömer's Quantum Computation Language, QCL [26]. The former provides a graphical tool for building and simulating quantum circuits using the gate formalism for quantum computation. It does not offer a concise programming language and is not able to implement and simulate all known quantum algorithms.

Ömer's QCL is a high-level architecture-independent programming language for quantum computers, with a syntax very like that of C and an interpreter powerful enough to implement and simulate all known quantum algorithms. It incorporates neither probabilism nor nondeterminism, has no notion of program refinement (and so no refinement calculus) and no semantics; furthermore only standard observation is allowed. QCL is appropriate for numerical simulation of quantum algorithms, whilst *qGCL*'s abstraction, rigorous semantics and associated refinement calculus seem to make it more suitable for program derivation, correctness proof and teaching.

Only experience will show whether *qGCL* is pitched at the right level of abstraction. However to support that view we here express in it a representative selection of quantum algorithms and perform an exemplary, though simple, derivation.

2 Quantum types

In this section we study, for use in quantum computation, a transformation q that converts a classical type to its quantum analogue. With but one simple exception, in section 6.5, quantum algorithms require application of q only to registers and so here we restrict ourselves to that case.

Let $\mathbb{B}$ denote the type $\{0, 1\}$ treated either as booleans or bits, as convenience dictates. For natural number n let $0 \mathinner{..} n$ denote the interval of natural numbers at least 0 but less than n

$$0\,..\,n \;\widehat{=}\; \{i \mid 0 \le i < n\}\,.$$

A (classical) register of size n is a vector of n booleans. The type of all registers of size n is thus defined to be the set of boolean-valued functions on $0\,..\,n$

$$\mathbb{B}^n \;\widehat{=}\; 0\,..\,n \longrightarrow \mathbb{B}\,.$$

Naturally we are interested in n at least 1 and identify $\mathbb{B}^1$ with $\mathbb{B}$.

The state of a classical system can be expressed using registers. From quantum theory we learn [1] —for example from Young's double-slit experiment— that the state of a quantum system is modelled using 'phase' information associated with each standard state. We follow convention [13, 27] and represent phase as a complex number of modulus at most 1. The probability of observing a state is then the modulus squared of its phase; and all probabilities sum to 1.

That leads to the following definition.

The quantum analogue of $\mathbb{B}^n$ is

$$q(\mathbb{B}^n) \;\widehat{=}\; \{\chi : \mathbb{B}^n \longrightarrow \mathbb{C} \mid \sum_{x:\mathbb{B}^n} |\chi(x)|^2 = 1\}\,. \tag{1}$$

An element of $q(\mathbb{B})$ is called a *qubit* [28] and that of $q(\mathbb{B}^n)$ a *qureg*.

Classical state is embedded in its quantum analogue by the Dirac delta function

$$\delta : \mathbb{B}^n \longrightarrow q(\mathbb{B}^n)$$
$$\delta_x(y) \;=\; (y = x)\,.$$

The range of δ, $\{\delta_x \mid x \in \mathbb{B}^n\}$, forms a *basis* for quantum states in this sense:

> any qureg $\chi : q(\mathbb{B}^n)$ is a square-convex complex superposition of standard states
>
> $$\chi \;=\; \textstyle\sum_{x:\mathbb{B}^n} \chi(x)\delta_x, \qquad \sum_{x:\mathbb{B}^n} |\chi(x)|^2 = 1\,.$$

(In physics δ_x is denoted by the ket $|x\rangle$. Our choice of notation has been determined by audience background.)

The Hilbert space $\mathbb{B}^n \longrightarrow \mathbb{C}$ (with the structure making it isomorphic to $\mathbb{C}^{2^n}$) is called the *enveloping space* of $q(\mathbb{B}^n)$; it is the Hilbert space of lowest dimension containing $q(\mathbb{B}^n)$ as unit sphere. We shall see that, because the elements of the range of δ are pairwise orthogonal in the enveloping space, they are observably distinct with probability 1.

---

[1] In the talk which this paper accompanies the relevant features of quantum theory will be introduced in a tutorial manner.

3 Tensor products

In a standard programming language the state of a program having independent component program variables can be expressed, more for theoretical than practical convenience, as a single variable equal to the Cartesian product of the components. The quantum analogue is that quantum state is the tensor product of its independent state components (equation (2)). In describing algorithms we thus have a choice between using individual variables, combining them when required (for example by finalisation) using tensor product; and using a vector of variables but subjecting it to transformation by the tensor product of a particular function on a particular component with the identity function on the remaining components. To support both approaches we require the tensor product both of registers and of functions.

The tensor product of (standard) registers is defined

$$\begin{array}{l} \otimes : \mathbb{B}^m \times \mathbb{B}^n \longrightarrow \mathbb{B}^{m+n} \\ (x \otimes y)(i) \ \ \widehat{=} \ \ x(i \operatorname{div} n) \times y(i \bmod n) \end{array}$$

and readily shown to be surjective. That definition lifts, via δ and linearity, to quantum registers

$$\otimes : q(\mathbb{B}^m) \times q(\mathbb{B}^n) \longrightarrow q(\mathbb{B}^{m+n}) \,.$$

Well definedness (i.e. square-summability to 1) is immediate.

For sets E and F of qregs we write

$$E \otimes F \ \ \widehat{=} \ \ \{\chi \otimes \xi \mid \chi \in E \wedge \xi \in F\} \,.$$

Then the property of q alluded to above is the isomorphism

$$q(\mathbb{B}^m \times \mathbb{B}^n) \ \cong \ q(\mathbb{B}^m) \otimes q(\mathbb{B}^n) \,. \tag{2}$$

(Since both sides are finite-dimensional vector spaces the proof is a matter of counting dimension. The left-hand side evidently has basis

$$\{(\delta_x, \delta_y) \mid x \in \mathbb{B}^m \wedge y \in \mathbb{B}^n\}$$

whilst a basis for the right-hand side consists of the equinumerous set

$$\{\delta_x \otimes \delta_y \mid x \in \mathbb{B}^m \wedge y \in \mathbb{B}^n\} \,.)$$

Next tensor product of functions on registers is defined

$$\begin{array}{l} \otimes : (\mathbb{B}^m \longrightarrow \mathbb{B}^m) \times (\mathbb{B}^n \longrightarrow \mathbb{B}^n) \longrightarrow (\mathbb{B}^{m+n} \longrightarrow \mathbb{B}^{m+n}) \\ (A \otimes B)(x \otimes y) \ \ \widehat{=} \ \ A(x) \otimes B(y) \,. \end{array}$$

Finally $\otimes$ is extended by linearity to functions on quantum registers, for which we follow tradition and use the same symbol yet again

$$\otimes : q(\mathbb{B}^m \longrightarrow \mathbb{B}^m) \times q(\mathbb{B}^n \longrightarrow \mathbb{B}^n) \longrightarrow q(\mathbb{B}^{m+n} \longrightarrow \mathbb{B}^{m+n}) \,.$$

4 Probabilistic language pGCL

In the next section we introduce an imperative quantum-programming language. But first, in this section, we recall Dijsktra's guarded-command language [11], GCL, extended to include probabilism [21, 23] and called $pGCL$.

Syntax for the guarded-command language consists of all but the last of these constructs

var	variable declaration
skip	no op
abort	abortion
$x := e$	assignment
$P \,\fatsemi\, Q$	sequencing
if $[]\ b_i \longrightarrow S_i$ **fi**	conditional
do $[]\ b_i \longrightarrow S_i$ **od**	iteration
$P \sqcap Q$	(demonic) nondeterminism
$P\ {}_r\oplus\ Q$	probabilism.

Semantics can be given either in terms of predicate transformers [11] or binary relations [17]. In the former case each program is thought of as transforming a post-condition to the weakest precondition from which termination, in a state satisfying that postcondition, is guaranteed. In the latter case each program is thought of as transforming initial state to final state, with a virtual state encoding non-termination.

We require the language to be extended, as usual, to embrace procedure invocation; see for example [20].

$pGCL$ denotes the guarded-command language extended to contain probabilism. Program $P\ {}_r\oplus\ Q$ equals P with probability r and Q with probability $1-r$. Its semantics has been given in two forms, following the semantic styles of GCL. The transformer semantics [22] extends pre- and post-conditions to pre- and post-*expectations*: real-valued random variables; the relational semantics [16] relates each initial state to a set of final distributions. In either case refinement $P \sqsubseteq Q$ means that Q is at least as deterministic as P. The two models are related by a Galois connection embedding the relational in the transformer [22]. There is a family of sound laws [16, 23], including those for data refinement, so that the language $pGCL$ is embedded in a refinement calculus. It is that feature which we exploit.

In $pGCL$ (demonic) nondeterminism is expressed semantically as the combination of all possible probabilistic resolutions

$$P \sqcap Q \;=\; \sqcap\{P\ {}_r\oplus\ Q \mid 0 \le r \le 1\}\,. \tag{3}$$

Thus a (demonic) nondeterministic choice between two programs is refined by any probabilistic choice between them

$$\forall\, r : [0,1] \bullet P \sqcap Q \sqsubseteq P\ {}_r\oplus\ Q \qquad \text{(introduce probabilism)}\,.$$

Probabilism does not itself yield nondeterminism: if P and Q are deterministic (maximal with respect to the refinement order) then so is $P\ {}_r\!\oplus\ Q$. Unfortunately for most authors in the area of quantum computation *nondeterminism* means probabilism. One of the important (and technically difficult) features of *pGCL* is its combination of (demonic) nondeterminism and probabilism; the result seems to provide just the right expressive power for the treatment of quantum algorithms. Indeed of the examples to follow, those of Grover, Shor and Deutsch-Jozsa all feature both (demonic) nondeterminism and probabilism.

If a set E of expressions contains more than one element then in the guarded-command language the assignment $x :\in E$ means the nondeterministic choice over all individual assignments of elements of E to x. In *pGCL* that choice is interpreted to occur with *uniform* probability.

As we need them we introduce two pieces of derived syntax concerning probabilism: one a prefix combinator (display (6) to follow); the other weakening exact probability r in probabilistic choice to the interval $[r, 1]$ (definition (9) to follow).

5 Quantum language qGCL

A *quantum program* is a *pGCL* program invoking quantum procedures (described below); the resulting language is called *qGCL*. It is important for us that *qGCL*, being expressed in terms of *pGCL*, inherits its refinement calculus. That enables us to combine code and specifications (and, less of a problem, to benefit from the usual liberties in writing programs, like using as guard the predicate 'N times') since the result has a semantic denotation to which refinement applies.

There are three types of *quantum procedure*: *initialisation* (or state preparation) followed by *evolution* and finally *finalisation* (or observation or state reduction). We now explain each of those three terms.

5.1 Initialisation

Initialisation is a procedure which simply assigns to its qureg state the uniform square-convex combination of all standard states

$$\begin{array}{l} \chi : q(\mathbb{B}^n) \\ \mathit{In}\,(\chi) \;\;\widehat{=}\;\; \chi := 2^{-n/2} \sum_{x:\mathbb{B}^n} \delta_x \,. \end{array}$$

There χ is a result parameter.

Initialisation so defined is *feasible* in the sense that it is achievable in practice [8] by initialising the qureg to the classical state $\delta_{\mathbf{0}}$ (where $\mathbf{0}$ denotes the register identically false) and then subjecting that to evolution by the (unitary) Hadamard transform, defined as a tensor power:

$$\begin{array}{r} H_n : q(\mathbb{B}^n) \longrightarrow q(\mathbb{B}^n) \\ H_1(\chi)(x) \,\widehat{=}\, 2^{-1/2}(\chi(0) + (-1)^x \chi(1)) \\ H_{n+1} \,\widehat{=}\, H_n \otimes H_1 \end{array} \tag{4}$$

where exponentiation of bits is standard $(-1)^x \;=\; -1 \lhd x \rhd 1$.

5.2 Evolution

Evolution consists of iteration of unitary transformations on quantum state. (It is thought of, after initialisation, as achieving all superposed evolutions simultaneously, which provides much of the reason for quantum computation's efficiency.) Again, evolution is feasible: it may be implemented using universal quantum gates [3, 9].

For example on $\mathbb{B}$, after initialisation, evolution by the Hadamard transformation H_1 results in $\chi = \delta_0$ (because H_1 is not only unitary but equal to its own conjugate transpose and so self-inverse). Thus our definition of initialisation does not exclude setting state to equal δ_0 (or any other standard state for that matter). That fact is used in procedure Q in Shor's algorithm (and similarly in Simon's algorithm, not considered here).

Later we use this important example of evolution: for function $f : \mathbb{B}^n \longrightarrow \mathbb{B}^n$ between registers, transformation T_f between the corresponding quregs is defined *pointwise* to invert χ about 0 if f holds and otherwise to leave it unchanged

$$\begin{array}{l} T_f : q(\mathbb{B}^n) \longrightarrow q(\mathbb{B}^n) \\ (T_f\chi)(x) \ \widehat{=} \ (-1)^{f(x)}\chi(x) \ = \ -\chi(x) \lhd f(x) \rhd \chi(x) \,. \end{array} \tag{5}$$

Evidently T_f is unitary.

More complicated evolutions appear in section 6.

5.3 Finalisation

Finalisation is a little more difficult to define largely because of the notation required. We motivate it by considering first the simple qubit case (later called 'diagonal').

Simple observation of a qubit $\chi = \chi(0)\delta_0 + \chi(1)\delta_1$ reduces it, by the principles of quantum theory, to the standard state δ_x with probability $|\chi(x)|^2$, for $x : \mathbb{B}$. Thus it might be expressed, using probabilistic assignment, as a procedure with result parameter χ

$$\begin{array}{l} \chi : q(\mathbb{B}) \\ (\chi := \delta_0) \ {}_{|\chi(0)|^2}\!\oplus \ (\chi := \delta_1) \,. \end{array}$$

We find it convenient, for more general forms of observation, to conform to standard practice and return not just the reduced state (the eigenvector of the matrix corresponding to the observation) but also the eigenvalue, in this case 0 or 1. At the same time we note that the probability $|\chi(0)|^2$ equals the inner product of the vector χ in enveloping space with its projection on the one-dimensional subspace $\mathbb{C}\delta_0$

$$\langle \chi, P_{\mathbb{C}\delta_0}(\chi) \rangle \ = \ \chi(0)\overline{\chi(0)} \ = |\chi(0)|^2$$

where angle brackets denote inner product, $P_E(\chi)$ denotes projection of χ onto subspace E and overline denotes complex conjugate. The procedure above then becomes

$$x : \mathbb{B},\ \chi : q(\mathbb{B})$$
$$(x, \chi := 0, \delta_0)\ {}_{\langle \chi, P_{\mathbb{C}\,\delta_0}(\chi)\rangle}\!\oplus\ (x, \chi := 1, \delta_1)\,.$$

In that case enveloping space $\mathbb{B} \longrightarrow \mathbb{C}$ is the direct sum of the orthogonal subpsaces $\mathbb{C}\,\delta_x$ for $x : \mathbb{B}$.

We now extend that simple case from qubits to quregs and from the family of subspaces $\mathbb{C}\,\delta_x$ to a family of arbitrary pairwise orthogonal subspaces which span enveloping space. In order to do so it is convenient to use the following notation for the probabilistic combination of a list of more than two programs.

If $[\,(P_j, r_j) \mid 0 \le j < m\,]$ denotes a finite indexed family of (program, number) pairs with $\sum_{0 \le j < m} r_j = 1$, then the probabilistic choice in which P_j is chosen with probability r_j is written in prefix form

$$\oplus[\,P_j\ @\ r_j \mid 0 \le j < m\,] \tag{6}$$

(whose advantage is to avoid the normalising factors required by nested infix form).

Let $\mathcal{V} = [\,V_j \mid 0 \le j < m\,]$ be an indexed family of pairwise orthogonal subspaces which together span enveloping space,

$$\mathit{span}\,[\,V_j \mid 0 \le j < m\,] \ = \ \mathbb{B}^n \longrightarrow \mathbb{C}\,,$$

where $\mathit{span}\,E$ denotes the (complex) vector space generated by any subset E of enveloping space. Finalisation with respect to $\mathcal{V}$ is defined to consist of a procedure which reduces state to lie in one of the subspaces in $\mathcal{V}$, with probability determined as it was in the simple case above:

$$i : 0\,..\,m,\ \chi : q(\mathbb{B}^n\,)$$
$$\mathit{Fin}[\mathcal{V}]\,(i, \chi)\ \ \widehat{=}\ \ \oplus[\,(i, \chi :\in \{j\}, V_j)\ @\ \langle \chi, P_{V_j}(\chi)\rangle\ \mid 0 \le j < m\,]$$

wherein i is a result parameter determining the subspace to which state is reduced and χ is a value-result parameter giving that state. In most cases (and with good physical reason if the observation is not 'complete' —i.e. V_i is more than one-dimensional— χ is not used, in which case we simply suppress it. We include χ in the definition of finalisation, however, because one of the quantum algorithms requires it (the last example). That definition provides the law 'introduce finalisation'.

The simple form of finalisation introduced in the qubit case is sufficiently important to warrant its own notation. We write Δ for the indexed family of subspaces $[\mathbb{C}\,\delta_x \mid x \in \mathbb{B}^n\,]$. Then finalisation with respect to Δ is called *diagonal* finalisation and abbreviated

$$x : \mathbb{B}^n$$
$$\mathit{Fin}[\Delta]\,(x)\,.$$

Its definition reduces to

$$\oplus[\,x\ @\ |\chi(x)|^2\ \ |\ \ x \in \mathbb{B}^n\,]$$

and the suppressed value of χ is determined by that of x since $q(\mathbb{B}^n) \cap \mathbb{C}\delta_x$ is a singleton. When an output number $i : 0 \mathinner{..} 2^n$ is required, it is produced by applying to $x : \mathbb{B}^n$ the function which yields a number, $num(x)$, whose binary representation equals its argument

$$
\begin{aligned}
& num : \mathbb{B}^n \longrightarrow 0 \mathinner{..} 2^n \\
& num(x) \ \ \widehat{=} \ \ \textstyle\sum_{j:0..n} x(j)2^j \,.
\end{aligned}
\tag{7}
$$

The definition of finalisation accords with general principles of quantum theory (e.g. [13, 18, 27]), which permit *simultaneous* finalisation (or observation) —i.e. in either order with the same result— since the subspaces in $\mathcal{V}$ are orthogonal. Thus feasibility of that definition is assured by general principles and in particular by Jozsa's characterisation [19] of quantum-observable functions.

It is interesting to note that finalisation is no more restrictive than probabilistic choice. Indeed a simple trigonometric argument shows that $P \,{}_r\!\oplus\, Q$ can be achieved by a quantum program which uses $\oplus$ only in the form defined by finalisation.

For examples of finalisation we proceed to the next section.

6 Example programs

In this section we demonstrate the expressive power of *qGCL* by casting in it a representative selection of quantum algorithms and their specifications. Although it is their efficiency which validates these algorithms, we are interested here in formalising functionality. With each algorithm we state the feature of *qGCL* it illustrates.

6.1 Fair coin

The first example is chosen to illustrate initialisation and diagonal finalisation without any evolution, and is included as a consistency check. It shows how the formalism is able to capture genuine probabilistic behaviour (i.e. not merely that of a finite automaton satisfying some fairness condition).

The example finds serious application in formalisation of the 'Mach-Zehnder interferometer' and, in particular, so-called 'interference-free measurement' [12]. In that setting the following program models a beam-splitter (a half-silvered mirror which either transmits or reflects incident photons with equal probability) and the Hadamard transform (4) is used for evolution.

The toss of a fair coin is modelled by specifying the result to be a uniformly-distributed boolean:

$$
\begin{aligned}
& \mathbf{var}\ i : \mathbb{B} \bullet \\
& \quad i :\in \mathbb{B}
\end{aligned}
$$

A quantum implementation is

$$
\begin{array}{l}
\textbf{var } \chi : q(\mathbb{B}),\ \ i : \mathbb{B} \bullet \\
\quad \mathit{In}\,(\chi) \,\fatsemi \\
\quad \mathit{Fin}[\Delta]\,(i)
\end{array}
$$

which may be checked to satisfy its specification since the probability with which $i = 0$ is of course

$$|\,\chi(0)\,|^2 \ = \ (2^{-1/2})^2 \ = \ \tfrac{1}{2}\,.$$

A formal proof is immediate from the definitions of initialisation and finalisation.

6.2 Grover's point search

The previous program can be proved to meet its probabilistic specification. The next example provides a more typical quantum algorithm which achieves its naïve specification only to within a margin of error. This example thus shows how *pGCL* (and hence *qGCL*) captures this important type of behaviour.

The point search problem is: given an array f of 2^n bits containing a single 1, locate it. A program which is correct on every execution is specified without any recourse to probability:

$$
\begin{array}{l}
\textbf{var } j : 0\,..\,2^n \bullet \\
\quad j := f^{-1}(1)\,.
\end{array}
\tag{8}
$$

A standard algorithm is at best $O(2^n)$ in both the worst and average cases. However Grover's quantum algorithm [14], although correct only to within a margin of error ε (dependent on the number of loop iterations), is $O(2^{n/2})$ in both those cases. It is conveniently specified by introducing some derived syntax: $P_{\geq r}{\oplus}\, Q$ equals P with probability at least r and otherwise equals Q . It is defined (cf. equation (3))

$$P\ {}_{\geq r}{\oplus}\ Q \ \ \widehat{=}\ \ \sqcap\{P\ {}_{s}{\oplus}\ Q \mid r \leq s \leq 1\} \tag{9}$$

which by a semantic convexity argument [23] simplifies to $(P\ {}_{r}{\oplus}\ Q) \sqcap P$.

The error-prone point-search problem is thus specified to behave, with probability at least ε, like the naïve behaviour (8) and otherwise to terminate with an arbitrary value for j

$$
\begin{array}{l}
\textbf{var } j : 0\,..\,2^n \bullet \\
\quad (j := f^{-1}(1))\ \ {}_{\geq\varepsilon}{\oplus}\ (j \in 0\,..\,2^n)\,.
\end{array}
$$

Grover's implementation contains evolution expressed as a loop and uses the function *num* (see (7)).

$$
\begin{array}{l}
\textbf{var } \chi : q(\mathbb{B}^n),\ i : \mathbb{B}^n,\ j : 0 \mathinner{..} 2^n \bullet \\
\quad In\,(\chi) \,⨾ \\
\quad \textbf{do}\ \ N \text{ times } \longrightarrow \\
\qquad \chi := T_f(\chi) \,⨾ \\
\qquad \chi := M(\chi) \\
\quad \textbf{od} \,⨾ \\
\quad Fin[\Delta]\,(i) \,⨾ \\
\quad j := num(i)
\end{array}
$$

There transform T_f is defined by (5) and transform M inverts χ (pointwise) about its average

$$
\begin{array}{l}
M : q(\mathbb{B}^n) \longrightarrow q(\mathbb{B}^n) \\
(M\chi)(x) \ \ \widehat{=}\ \ 2\,[2^{-n} \sum_{y:\mathbb{B}^n}\ \chi(y)] - \chi(x)\,.
\end{array}
$$

We are not here concerned with the choice of N which determines the number of iterations of the loop. The function $\varepsilon = \varepsilon(N)$ is investigated in [4] and its place in a semantic (expectation-transformer) proof of correctness is explored in [5].

6.3 Deutsch-Jozsa classification

So far the specifications have been (demonically) deterministic (though probabilistic) and we have used only diagonal finalisation. The next example meets a nondeterministic specification, exhibits non-diagonal finalisation and requires no margin for error.

A truth function $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ is *constant* iff it takes only a single value. It is *balanced* iff it takes values 0 and 1 equally often

$$\# f^{-1}(0) = \# f^{-1}(1)\,.$$

For use in the next section we note:

$$
\begin{array}{l}
f \text{ is constant iff } \# f^{-1}(1) \in \{0, 2^n\}, \\
f \text{ is balanced iff } \# f^{-1}(1) = 2^{n-1}, \\
\text{and } \# f^{-1}(1) \ = \ \sum_{x:\mathbb{B}^n} f(x)\,.
\end{array}
\tag{10}
$$

Any constant truth function f is not balanced. So any truth function is either not balanced or not constant, usually both. The Deutsch-Jozsa classification problem is to decide, for a given truth function, which holds; if both hold then either answer is correct.

Letting the result be encoded by variable $i : \mathbb{B}$, the problem is specified

$$
\begin{array}{l}
\textbf{var } i : \mathbb{B} \bullet \\
\quad \textbf{if} \quad i \longrightarrow f \text{ not balanced} \\
\quad [\,] \ \neg\, i \longrightarrow f \text{ not constant} \\
\quad \textbf{fi}
\end{array}
$$

A standard algorithm for the Deutsch-Jozsa classification problem is at least $O(2^n)$ in the worst case and on average evaulates f thrice. Deutsch and Jozsa's quantum algorithm [10] contains just *one* evolution step using the transformation T_f defined by equation (5). It is expressed in our notation:

$$
\begin{array}{l}
\textbf{var } \chi : q(\mathbb{B}^n),\ i : \mathbb{B} \bullet \\
\quad In\,(\chi)\ ⨾ \\
\quad \chi := T_f(\chi)\ ⨾ \\
\quad Fin[\mathcal{V}]\,(i)
\end{array}
$$

where finalisation is non-diagonal

$$
\begin{aligned}
\mathcal{V} &\mathrel{\widehat{=}} [\, V, V^{\perp} \,] \\
V &\mathrel{\widehat{=}} \mathbb{C} \textstyle\sum_{y:\mathbb{B}^n} \delta_y \\
V^{\perp} &\mathrel{\widehat{=}} \text{the orthogonal complement of } V\,.
\end{aligned}
$$

A derivation of that algorithm (and hence its correctness) is exhibited in the next section.

6.4 Shor's factorisation algorithm

Shor's quantum algorithms [29] for factorisation and for discrete logarithm are at once the most mathematically sophisticated and relatively efficient practical quantum algorithms known. We consider the former algorithm which, as has been widely advertised, makes factorisation feasible by achieving an average-case polynomial efficiency instead of the standard exponential. Although it demonstrates no new features of *qGCL* we include it as the most important quantum algorithm to date.

The factorisation problem is: given a natural number $n > 1$ find a prime divisor d of n. It is thus naturally nondeterministic (as was Deutsch-Jozsa classification):

$$
\begin{array}{l}
\textbf{var } d : 1 \mathrel{..} (n{+}1) \bullet \\
\quad d \text{ is a prime divisor of } n\,.
\end{array}
$$

For natural numbers x and y, we write $x \sqcup y$ for their maximum and $gcd(x, y)$ for their greatest-common divisor. Shor's algorithm is

$$
\begin{array}{l}
\textbf{var}\ t : \mathbb{B},\ \ a, d, p : 0\,..\,(n{+}1) \bullet \\
\quad t := 0\ ⨾ \\
\quad \textbf{do}\ \ \neg t \ \ \longrightarrow \\
\qquad\quad a :\in 2\,..\,n\ ⨾ \\
\qquad\quad d := gcd(a, n)\ ⨾ \\
\qquad\quad \textbf{if}\ d \neq 1 \ \ \longrightarrow \ \ t := 1 \\
\qquad\quad [\,]\ \ d = 1 \ \ \longrightarrow \ \ Q(a, n; p)\ ⨾ \\
\qquad\qquad\qquad\qquad\quad \textbf{if}\ p \text{ odd} \ \ \longrightarrow \ \ t := 1 \\
\qquad\qquad\qquad\qquad\quad [\,]\ p \text{ even} \ \ \longrightarrow \\
\qquad\qquad\qquad\qquad\quad d := gcd(a^{p/2}{-}1, n) \sqcup gcd(a^{p/2}{+}1, n)\ ⨾ \\
\qquad\qquad\qquad\qquad\quad t := (d \neq 1) \\
\qquad\qquad\qquad\qquad\quad \textbf{fi} \\
\qquad\quad \textbf{fi} \\
\quad \textbf{od}
\end{array}
$$

The quantum content lies in procedure Q. It is our first example to use quantum state after finalisation, though it does so for only standard purposes.

$$
\begin{array}{l}
\textbf{var}\ \chi : q(\mathbb{B}^m \times \mathbb{B}^m),\ \ x : \mathbb{B}^{2m},\ \ c : \mathbb{B}^m \ \bullet \\
\quad In\,(\chi)\ ⨾ \\
\quad \chi := (I_m \otimes H_m)(\chi)\ ⨾ \\
\quad \chi := E(\chi)\ ⨾ \\
\quad \chi := (F_m \otimes I_m)(\chi)\ ⨾ \\
\quad Fin[\Delta]\,(x, \chi)\ ⨾ \\
\quad c := P_m(\chi)\ ⨾ \\
\quad p := \text{post processing of } c
\end{array}
$$

where:

m satisfies $n^2 \leq 2^m \leq 2n^2$;

H_m denotes the Hadamard transform defined by equation (4);

unitary transformation $E : q(\mathbb{B}^m \times \mathbb{B}^m) \longrightarrow q(\mathbb{B}^m \times \mathbb{B}^m)$ is defined in terms of modular exponentiation

$$E(\chi)(x, y) \ \ \widehat{=} \ \ (x, y \oplus num^{-1}(a^{num(x)} \bmod n))\,;$$

$F_m : q(\mathbb{B}^m) \longrightarrow q(\mathbb{B}^m)$ is the quantum Fourier transform (see [15], section 3.2.2);

diagonal finalisation has been extended to return also state χ ;

$P_m : \delta(\mathbb{B}^m \times \mathbb{B}^m) \longrightarrow \mathbb{B}^m$ denotes a kind of projection

$$P_m(\delta_x \otimes \delta_y) \ \widehat{=}\ x \ ; \text{ and}$$

the post processing of c is standard, using continued fractions to find efficiently the unique p for which

$$|\ num(c)/2^m - d/p\ | \leq 2^{-(m+1)} .$$

The first two lines of procedure Q are equivalent (see section 5.2) to

$$\chi := (H_m \otimes I_n)(\delta_{\mathbf{0}})$$

(where $\delta_{\mathbf{0}}$ denotes the qureg containing $m+n$ zeroes); however our insistence that quantum programs begin with a standard initialisation obliges us to take the longer version.

Simon's quantum algorithm for his masking problem [30] is similar in structure, from our current point of view, to Shor's factorisation algorithm.

6.5 Finite automaton

The previous algorithm uses quantum state *after* finalisation for the purpose of (standard) post processing. However since finalisation was diagonal, the quantum state could have been inferred from the eigenvalue returned by finalisation. The next example uses non-diagonal finalisation and makes genuine use of quantum state after finalisation. It thus justifies our inclusion of state in finalisation.

Recall that (standard) finite automata, whether deterministic or not and one-way or two-way, accept just regular languages. For quantum finite automata enough is already known to demonstrate that the picture is quite different (see [15], chapter 4).

A *one-measurement one-way quantum finite automaton* employs finalisation after reading its input string and so is readily expressed in our notation. So instead we consider the *many-measurement* version which employs finalisation after reading each symbol on its input string. It turns out (Kondacs and Watrous; see, for example, [15] p. 159) that a many-measurement one-way quantum finite automaton accepts a proper subset of regular expressions. Here we give sufficient of the definition to permit its translation into our programming language.

For any set Σ, let Σ^* denote the set of all finite sequences of elements of Σ. Suppose that set $\mathcal{S} = \{S_a, S_r, S_n\}$ of subsets of Σ^* *partitions* Σ^*. A sequence $s : \Sigma^*$ is said to be *accepted* by $\mathcal{S}$ if $s \in S_a$, to be *rejected* by it if $s \in S_r$ and to *fail classification* if $s \in S_n$. Evaluation of that is specified:

var $i : \{a, r, n\} \bullet$
  $s \in S_i$.

But here we are interested in computing whether a prefix of a given sequence is accepted or rejected, since that gives rise to an automaton which continues to use its quantum state after finalisation. Its specification thus extends the previous one. In it $t \leq s$ means that sequence t is a prefix of sequence s.

$$\begin{array}{l}\textbf{var}\ i : \{a, r, n\} \bullet \\ \left(\begin{array}{l} i = a \Rightarrow \exists\, t \leq s \bullet t \in S_a \\ i = r \Rightarrow \exists\, t \leq s \bullet t \in S_r \\ i = n \Rightarrow s \in S_n \end{array}\right)\end{array}$$

(A stronger specification might reflect the fact that computation proceeds from left to right and so ensure that sequence t there is smallest.)

A *many-measurement one-way quantum finite automaton* is designed to achieve such a computation with efficient quantum evolution. It has a finite set Q of states, with distinguished acceptance states Q_a and rejection states Q_r

$$\begin{array}{l} Q_a \subseteq Q \\ Q_r \subseteq Q \\ Q_a \cap Q_r = \{\,\}\,. \end{array}$$

Thus $Q_a \cup Q_r$ need not exhaust Q.

On input sequence $s = [\sigma_0, \ldots, \sigma_{n-1}] : \Sigma^*$ the automaton evolves successively under unitary transformations

$$U_{init}, U_{\sigma_0}, \ldots, U_{\sigma_{n-1}}$$

subject to finalisation after each. If a finalisation leaves the automaton in an acceptance state then computation terminates with output value $i = a$; if it leaves the automaton in a rejection state then computation terminates with output value $i = r$; but otherwise the automaton reiterates. If it has not accepted or rejected a prefix of the input sequence, it terminates when the entire sequence has been read, with value $i = n$.

We thus let quantum state belong to $q(Q)$, defined as was qureg state by (1). Initialisation over $q(Q)$ is defined as for registers; its feasibility is assured by solubility of the appropriate simultaneous equations describing a unitary transformation that yields a uniform image of $\delta_{\mathbf{0}}$. For finalisation we take

$$\begin{array}{l} \mathcal{V} \mathrel{\widehat{=}} [\, V_a, V_r, V_n \,] \\ V_a \mathrel{\widehat{=}} span\{\delta_x \mid x \in Q_a\} \\ V_r \mathrel{\widehat{=}} span\{\delta_x \mid x \in Q_r\} \\ V_n \mathrel{\widehat{=}} (V_a \oplus V_r)^{\perp}\,. \end{array}$$

A program for such an automaton is

$$
\begin{array}{l}
\textbf{var}\ \chi : q(Q),\ \ b : \mathbb{B} \bullet \\
\quad In(\chi)\ ; \\
\quad \chi, b := U_{init}(\chi), 0\ ; \\
\quad \textbf{do}\ \ \neg(b \vee s = [\,])\ \longrightarrow \\
\qquad\quad Fin[\mathcal{V}]\,(i, \chi)\ ; \\
\qquad\quad \textbf{if}\ \ i \in \{a, r\}\ \ \longrightarrow\ \ b := 1 \\
\qquad\quad [\,]\ i \notin \{a, r\}\ \ \longrightarrow\ \ \chi, s := U_{head(s)}(\chi), tail(s) \\
\qquad\quad \textbf{fi} \\
\quad \textbf{od}
\end{array}
$$

That can be expressed only because we allow quantum state to be returned by finalisation.

7 Example derivation

We conclude by outlining an algebraic derivation of the Deutsch-Jozsa classification algorithm. It is to be emphasised that derivations (or verifications) using the refinement calculus (i.e. the laws concerning refinement between programs —see for example [20]) are quite different in style from those phrased in terms of semantics (c.f. [5]). We are interested primarily in the shape of the derivation; and we shall see that it is largely standard. This example demonstrates how the refinement calculus which *qGCL* inherits from *pGCL* permits 'homogeneous' reasoning about the functionality of quantum algorithms, without recourse to arguments outside the formalism (pertaining for example to probabilism or 'quantism').

The following derivation can be followed intuitively as well as rigorously; the steps involved correspond largely to steps in an informal explanation of why the algorithm works. At one point $\sqsubseteq$ is extended to mean also data refinement, of which care must (in general) be taken in the probabilistic setting; but here the refinement is unproblematic. Interesting features of the derivation are the appearance of quantum state and of the three quantum procedures.

$$
\begin{array}{l}
\textbf{var}\ i : \mathbb{B} \bullet \\
\quad \textbf{if}\ \ \ \ i \longrightarrow f \text{ not balanced} \\
\quad [\,]\ \neg\ i \longrightarrow f \text{ not constant} \\
\quad \textbf{fi}
\end{array}
$$

$\sqsubseteq$ (10) and standard reasoning

$$
\begin{array}{l}
\textbf{var}\ i : \mathbb{B},\ j : 0\,..\,1{+}2^n \bullet \\
\quad j := \sum_{x:\mathbb{B}^n} f(x)\ ; \\
\quad \textbf{if}\ \ \ j \neq 2^{n-1}\ \ \longrightarrow\ \ i := 1 \\
\quad [\,]\ \ j \notin \{0, 2^n\}\ \ \longrightarrow\ \ i := 0 \\
\quad \textbf{fi}
\end{array}
$$

$\sqsubseteq$ standard reasoning

$$
\begin{array}{l}
\textbf{var } i:\mathbb{B},\ j:0\,..\,1{+}2^n \bullet \\
\quad j := \sum_{x:\mathbb{B}^n} f(x) \ ⨟ \\
\quad \begin{array}{lrcl}
\textbf{if} & j \in \{0, 2^n\} & \longrightarrow & i := 1 \\
[] & j = 2^{n-1} & \longrightarrow & i := 0 \\
[] & j \notin \{0, 2^{n-1}, 2^n\} & \longrightarrow & (i := 1) \sqcap (i := 0) \\
\textbf{fi} & & &
\end{array}
\end{array}
$$

$\sqsubseteq$ arithmetic and standard and probabilistic reasoning

$$
\begin{array}{l}
\textbf{var } i:\mathbb{B},\ j:0\,..\,1{+}2^n \bullet \\
\quad j := \sum_{x:\mathbb{B}^n} f(x) \ ⨟ \\
\quad \begin{array}{llcl}
\textbf{if} & j \in \{0, 2^{n-1}, 2^n\} & \longrightarrow & (i := 1) \ {}_{|1-j/2^{n-1}|}\!\oplus (i := 0) \\
[] & j \notin \{0, 2^{n-1}, 2^n\} & \longrightarrow & (i := 1) \sqcap (i := 0) \\
\textbf{fi} & & &
\end{array}
\end{array}
$$

$\sqsubseteq$ injective data refinement $k = 1{-}j/2^{n-1}$

$$
\begin{array}{l}
\textbf{var } i:\mathbb{B},\ k:[-1,1] \bullet \\
\quad k := 2^{-n} \sum_{x:\mathbb{B}^n} (-1)^{f(x)} \ ⨟ \\
\quad \begin{array}{llcl}
\textbf{if} & k \in \{-1, 0, 1\} & \longrightarrow & (i := 1) \ {}_{|k|}\!\oplus (i := 0) \\
[] & k \notin \{-1, 0, 1\} & \longrightarrow & (i := 1) \sqcap (i := 0) \\
\textbf{fi} & & &
\end{array}
\end{array}
$$

$\sqsubseteq$ 'introduce probabilism'

$$
\begin{array}{l}
\textbf{var } i:\mathbb{B},\ k:[-1,1] \bullet \\
\quad k := 2^{-n} \sum_{x:\mathbb{B}^n} (-1)^{f(x)} \ ⨟ \\
\quad (i := 1) \ {}_{|k|}\!\oplus (i := 0)
\end{array}
$$

$\sqsubseteq$ 'sequential composition'

$$
\begin{array}{l}
\textbf{var } i:\mathbb{B},\ k:[-1,1],\ \chi : q(\mathbb{B}^n) \bullet \\
\quad \chi := 2^{-n/2} \sum_{x:\mathbb{B}^n} (-1)^{f(x)} \delta_x \ ⨟ \\
\quad k := 2^{-n/2} \sum_{x:\mathbb{B}^n} \ \chi(x) \ ⨟ \\
\quad (i := 1) \ {}_{|k|}\!\oplus (i := 0)
\end{array}
$$

$\sqsubseteq$ 'sequential composition' and definition of T_f

$$
\begin{array}{l}
\textbf{var } i:\mathbb{B},\ k:[-1,1],\ \chi : q(\mathbb{B}^n) \bullet \\
\quad \chi := 2^{-n/2} \sum_{x:\mathbb{B}^n} \delta_x \ ⨟ \\
\quad \chi := T_f(\chi) \ ⨟ \\
\quad k := 2^{-n/2} \sum_{x:\mathbb{B}^n} \ \chi(x) \ ⨟ \\
\quad (i := 1) \ {}_{|k|}\!\oplus (i := 0)
\end{array}
$$

$\sqsubseteq$ definitions of In, Fin and diminish by $k = \langle \chi, 2^{-n/2} \sum_{x:\mathbb{B}^n} (-1)^{f(x)} \delta_x \rangle$

var $i : \mathbb{B},\ \chi : q(\mathbb{B}^n) \bullet$
  $In\,(\chi)\ ⨟$
  $\chi := T_f(\chi)\ ⨟$
  $Fin[\mathcal{V}]\,(i)$

with family $\mathcal{V} = [\ V,\ V^{\perp}\,]$, where $V = \mathbb{C}\ \sum_{y:\mathbb{B}^n} \delta_y$.

8 Conclusions

We have proposed a language, *qGCL*, for the expression of quantum algorithms and their derivations. It exhibits several features, many as a result of the work on *pGCL*:

1. expressivity: the language is sufficiently expressive to capture existing quantum algorithms
2. simplicity: the language seems to be as simple as possible (by law (3)) whilst containing (demonic) nondeterminism and probability (the latter either in a form restricted to 'observation' or for general use)
3. abstraction: the language contains control structures and data structures at the level of abstraction of today's imperative languages whilst abstracting implementation concerns (like the representation of a function f on the underlying standard types by its Lecerf-Bennett form on their quantum analogues)
4. calculation: the language has a formal semantics, sound laws and provides a refinement calculus supporting verification and derivation of quantum programs
5. the language provides a uniform treatment of 'observation'.

We conclude that it does seem possible to treat quantum programs in a refinement calculus with the same degree of elegance and rigour as standard algorithms. Starting from a specification in which standard and probabilistic (but not quantum) expressions appear it is possible to derive quantum algorithms by introducing algorithmic and quantum structure (in the guise of quantum state and the three quantum proceedures).

We have still to learn whether there are reuseable data refinements, or other derivation clichés, appropriate to the derivation of quantum programs. But abstraction from implementation concerns seems to make quantum algorithms easier to express, understand and reason about. Unmentioned here are more general properties of the functor q on types and work on the compilation of the programs expressed in *qGCL* ([32]).

9 Acknowledgements

The authors are pleased to acknowledge the quantity and quality of refereeing which resulted in an improved paper. In particular one referee is responsible for pointing out [1, 26].

References

[1] Greg Baker. `http://www.ics.mq.edu.au/ gregb/q-gol`.

[2] Adriano Barenco et al. Elementary gates of quantum computation. *Physical Review A*, **52**(5):3457–3467, 1995.

[3] Adriano Barenco. A universal two-bit gate for quantum computation. *Proc. R. Soc. Lond.* A, **449**:679–683, 1995.

[4] Michel Boyer, Gilles Brassard, Peter Hoyer and Alain Tapp. Tight bounds on quantum searching. In *Fourth Workshop on Physics and Computation*, editors T. Toffoli, M. Biaford and J. Lean, pages 36–43. New England Complex System Institute, 1996.

[5] Michael Butler and Pieter Hartel. Reasoning about Grover's quantum search algorithm using probabilistic wp. University of Southampton technical report DSSE-TR-98-10, 1998.

[6] R. Cleve, A. Ekert, C. Macchiavello and M. Mosca. Quantum algorithms revisited. *Proc. R. Soc. Lond.*, A. **454**:339–354, 1998.

[7] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.* A, **400**:97–117, 1985.

[8] D. Deutsch. Quantum computational networks. *Proc. R. Soc. Lond.* A, **425**:73–90, 1989.

[9] David Deutsch, Adriano Barenco and Artur Ekert. Universality in quantum computation. *Proc. R. Soc. Lond.* A, **449**:669–677, 1995.

[10] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond.* A, **439**:553–558, 1992.

[11] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall International, 1976.

[12] Avshalom C. Elitzur and Lev Vaidman. Quantum mechanical interaction-free measurements. *Foundations of Physics*, **32**(7):987–997, 1993.

[13] R. P. Feynman. *The Feynman Lectures on Physics*, volume 3. Addison-Wesley, 1964.

[14] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM STOC*, pages 212–219, 1996.

[15] Jozef Gruska. *Quantum Computing.* McGraw-Hill International (UK), Advanced Topics in Computer Science, 1999.

[16] He Jifeng, K. Seidel and A. K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, **28**:171–192, 1997.

[17] C. A. R. Hoare He Jifeng. The weakest prespecification, parts I and II. Fundamenta Informatica, IX, 51–84, 217–252, 1986.

[18] Chris J. Isham. *Lectures on Quantum Theory.* Imperial College Press, 1995.

[19] Richard Josza. Characterising classes of functions computable by quantum parallelism. *Proc. R. Soc. Lond.* A, **435**:563–574, 1991.

[20] Carroll Morgan. *Programming from Specifications*, second edition. Prentice-Hall International, 1994.

[21] K. Seidel, C. C. Morgan and A. K. McIver. Probabilistic imperative programming: a rigorous approach. 1996. Available at `http//www.comlab.ox.ac.uk/oucl/research/areas/probs/bibliography.html`.

[22] Carroll Morgan, Annabelle McIver and Karen Seidel. Probabilistic predicate transformers. *TOPLAS*, **18**(3):325–353, 1996.

[23] Carroll Morgan and Annabelle McIver. *pGCL*: formal reasoning for random algorithms. *South African Computer Journal*, **22**:14–27, 1999.

[24] Carroll Morgan and A. K. McIver. Demonic, angelic and unbounded probabilistic choices in sequential programs. To appear in *Acta Informatica*; see the site at [23].

[25] Michele Mosca and Artur Ekert. The hidden subgroup problem and eigenvalue estimation on a quantum computer. In *Proceedings of the 1st NASA Internation Conference on Quantum Compuing and Quantum Communication*. LNCS 1509, Springer-Verlag, 1998.

[26] Bernhard Ömer. `http://tph.tuwien.ac.at/ oemer`.

[27] Asher Peres. *Quantum Theory: Concepts and Methods*. Kluwer Academic Publishers, 1998.

[28] Benjamin Schumacher. Quantum coding. *Physical Review* A, **51**(4):2738–2747, 1995.

[29] Peter W. Shor. Algorithms for quantum computation: discrete log and factoring. In *Proceedings of the 35th IEEE FOCS*, pages 124–134, 1994.

[30] Daniel R. Simon. On the power of quantum computation. In *Proceedings of the 35th IEEE FOCS*, pages 116–123, 1994.

[31] Colin P. Williams and Scott H. Clearwater. *Explorations in Quantum Computing*. Springer-Verlag, New York, 1998.

[32] Paolo Zuliani. *DPhil Thesis*. Oxford University. In preparation.