A Formal Derivation of Grover's Quantum Search Algorithm

Paolo Zuliani
Oxford University Computing Laboratory
Oxford, OX1 3QD, UK
paolo.zuliani@comlab.ox.ac.uk

Abstract

In this paper we aim at applying established formal methods techniques to a recent software area: quantum programming. In particular, we aim at providing a stepwise derivation of Grover's quantum search algorithm. Our work shows that, in principle, traditional software engineering techniques such as specification and refinement can be applied to quantum programs. We have chosen Grover's algorithm as an example because it is one of the two main quantum algorithms. The algorithm can find with high probability an element in an unordered array of length L in just $O(\sqrt{L})$ steps (while any classical probabilistic algorithm needs $\Omega(L)$ steps). The derivation starts from a rigorous probabilistic specification of the search problem, then we stepwise refine that specification via standard refinement laws and quantum laws, until we arrive at a quantum program. The final program will thus be correct by construction.

1. Introduction

Quantum Mechanics has been recently applied to Computation with much benefit. In 1994 Shor [11] developed a polynomial-time quantum algorithm for integer factorisation, while no classical algorithm is known to require less than exponential time. In 1996 Grover [5] developed a quantum algorithm for finding with high probability an element in a unsorted array of length L with only $O(\sqrt{L})$ accesses to the array. In contrast, any classical algorithm needs $\Omega(L)$ accesses, even in the probabilistic case.

In this paper we aim at applying formal methods techniques to quantum programming, a model of quantum computation in which algorithms are expressed as imperative programs. In particular, we aim at providing a stepwise derivation of Grover's quantum search algorithm.

Quantum algorithms make use of highly non-trivial and counterintuitive concepts (*e.g.* superposition and measurement) which are completely new to the standard case. It is therefore essential, from a software engineering point of view, to understand whether standard techniques can cope with the quantum case. Furthermore, it is important that those techniques support a seamless integration of standard code and quantum code. That is because quantum algorithms are not just "pure quantum", rather, they may also employ standard processing. For example, Shor's factoring algorithm makes substantial use of standard number-theoretic processing, while the quantum part is in fact an efficient subroutine for computing the order of an integer modulo m (the order r of x modulo m is the least integer r such that $x^r = 1 \mod m$).

In this paper we have chosen to analyse Grover's algorithm because it uses a central concept of standard programming which is not used in other quantum algorithms: iteration. We start from a formal probabilistic specification of the search problem: finding the required element with probability at least $1 - \frac{1}{L}$ where L is the length of the array. Next, we stepwise refine such specification via standard refinement laws and quantum laws until we reach a feasible quantum program, which will therefore be correct by construction. Our work shows that, in principle, traditional software engineering techniques such as specification and refinement can be applied to the quantum case, thus providing an unique environment for both standard and quantum programming.

The framework is provided by qGCL [10], a programming language for quantum computation developed as a superset of the *probabilistic* guarded-command language pGCL [9]. qGCL inherits a rigorous semantics and an associated refinement calculus (see for example [8, 7]), which include program refinement, data refinement and combination of specifications with code. qGCL has been successfully used to describe all known quantum algorithms and also to derive Deutsch-Jozsa's algorithm from its specification. That algorithm is simpler than Grover's, and it does not feature iteration (although it employs a more complicated final step). Also, its specification is just (demonically) nondeterministic, while Grover problem' specification combines nondeterminism and probabilism.

As far as previous work is concerned, that of Butler and Hartel [3] is particularly relevant: they showed how to use the semantics of pGCL to derive the success probability of Grover's algorithm. In particular, that is computed as the pre-expectation of a suitable post-expectation (*i.e.* finding the required element) applied to Grover's algorithm. Such an approach is of course tailored for *proving* the correctness of an algorithm, rather than developing a correct implementation from a specification. For a survey and bibliography of quantum programming languages see [4].

We begin by giving a short presentation of the features of qGCL (a full introduction can be found in [10]). Then, we present Grover's algorithm [5] using qGCL. Finally, we outline our derivation of the algorithm.

2. Quantum programming language qGCL

2.1. Quantum types

Transformation q maps a classical bit register data type to its quantum analogue. We define the type $\mathbb{B} \mathrel{\widehat{=}} \{0,1\}$, which we will treat as booleans or bits, depending on convenience. A classical register of size $n{:}\mathbb{N}$ is a vector of n booleans. The type of all registers of size n is then defined to be the set of boolean-valued functions on $\{0,1,\ldots,n-1\}$:

$$\mathbb{B}^n \mathrel{\widehat{=}} \{0,1,\ldots,n-1\} \longrightarrow \mathbb{B}\,.$$

The quantum analogue of $\mathbb{B}^n$ is the set of complex-valued functions on $\mathbb{B}^n$ whose squared modulus sum to 1:

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

An element of $q(\mathbb{B})$ is called a *qubit* and that of $q(\mathbb{B}^n)$ a *qureg*. The numbers $\chi(x)$ are called *amplitudes*. Classical state is embedded in its quantum analogue by the Dirac delta function:

$$\begin{aligned}&\delta{:}\mathbb{B}^n \longrightarrow q(\mathbb{B}^n)\\ &\delta_x(y) \mathrel{\widehat{=}} (y = x)\,.\end{aligned}$$

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

$$\forall \chi{:}q(\mathbb{B}^n) \bullet \chi = \sum_{x:\mathbb{B}^n} \chi(x)\delta_x\,.$$

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 $q(\mathbb{B}^n)$. The usual scalar product becomes the application $\langle\cdot,\cdot\rangle{:}q(\mathbb{B}^n) \times q(\mathbb{B}^n) \to \mathbb{C}$ defined by:

$$\langle\psi,\phi\rangle \mathrel{\widehat{=}} \sum_{x:\mathbb{B}^n} \psi(x)^*\phi(x)$$

where z^* is the complex conjugate of $z{:}\mathbb{C}$. The *length* of ψ is defined $\|\psi\| \mathrel{\widehat{=}} \langle\psi,\psi\rangle^{\frac{1}{2}}$; all quregs have thus length 1.

The state of a composite quantum system is given by the tensor product of the states of the single systems, but since we shall not need it here, we do not define it. A complete definition can be found in [10].

2.2. qGCL

qGCL is an extension of pGCL, which in turn extends Dijkstra's guarded-command language with a probabilistic choice constructor in order to address probabilism. A guarded-command language program is a sequence of assignments, **skip** and **abort** manipulated by the standard constructors of sequential composition, conditional selection, repetition and nondeterministic choice. The syntax of pGCL is:

$prg \mathrel{\widehat{=}}$ **skip**	do nothing
$\mid$ **abort**	abortion
$\mid x := E$	assignment
$\mid prg \,\text{;}\, prg$	sequential composition
$\mid$ **do** $b \to prg$ **od**	iteration
$\mid prg \lhd b \rhd prg$	conditional selection
$\mid prg\ {}_p\!\oplus prg$	probabilistic choice
$\mid prg \sqcap prg$	demonic choice
$\mid$ **var** $v{:}D \bullet prg$ **rav**	variable declaration and local block

where b is a predicate and D a datatype; the probabilistic combinator ${}_p\!\oplus$ executes its LHS (RHS) with probability p $(1-p)$. Both probabilistic and nondeterministic choice may be written using a prefix notation. Let $[\ (P_j, r_j) \mid 0 \leqslant j < m\ ]$ be a finite indexed family of (program, number) pairs with $\sum_j r_j = 1$, then the probabilistic choice in which P_j is chosen with probability r_j is written in the form: $[\ P_j \ @\ r_j \mid 0 \leqslant j < m\ ]$. For nondeterministic choice the notation is similar.

A *quantum program* is a pGCL program invoking quantum procedures and the resulting language is called qGCL. Quantum procedures can be of three different kinds: *Initialisation* (or state preparation) followed by *Evolution* and finally by *Finalisation* (or observation).

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

$$\forall \chi{:}q(\mathbb{B}^n) \bullet \mathbf{In}(\chi) \mathrel{\widehat{=}} \left(\chi := \frac{1}{\sqrt{2^n}} \sum_{x:\mathbb{B}^n} \delta_x\right).$$

Initialisation is efficiently implemented by the Hadamard trasform (see [10] for more details).

Quantum-mechanical systems evolve over time under the action of *unitary* transformations:

$$U{:}q(\mathbb{B}^n) \rightarrow q(\mathbb{B}^n)\,,\text{linear}$$
$$U \text{ unitary iff } \ \|U\psi\| = \|\psi\| \text{ iff } \ UU^\dagger = U^\dagger U = I$$

where I is the identity transform and $U^\dagger$ is the conjugate transpose of U (in matrix representation). Unitary transformations thus preserve the length of the state. *Evolution* consists of iteration of unitary transformations on quantum state. In our formalism, evolution of qureg χ under unitary operator U is described by the assignment:

$$\chi := U\chi.$$

The content of a qureg can be read (measured) through quantum procedure *Finalisation* and suitable *observables*. The simplest observable - and the one we shall use - is the so-called *diagonal* observable. The effect of diagonal Finalisation on qureg $\chi{:}q(\mathbb{B}^n)$ is to "reduce" χ to state δ_j with probability $|\chi(j)|^2$. The measurement returns the value j, as well. In our notation we write:

$$\mathbf{Fin}\,(r,\chi) \mathrel{\widehat=} \left[\, (r, \chi := j, \delta_j) \;@\; |\chi(j)|^2 \mid 0 \leqslant j < m \,\right] \ .$$

A more general form of Finalisation is presented in [10].

2.3. Semantics

Semantics for pGCL (and in turn for qGCL) can be given either relationally [6] or in terms of expectation transformers [9]. The former relates each initial state to a set of final distributions. The latter extends pre- and post-conditions to pre- and post-*expectations*: real-valued random variables. 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 [9].

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

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

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

$$\forall r{:}[0,1] \bullet P \sqcap Q \sqsubseteq P\;{}_r{\oplus}\;Q\ . \tag{2}$$

We introduce another useful notation: $P\;{}_{\geqslant r}{\oplus}\;Q$ is equal to P with probability at least r and otherwise is equal to Q. Its definition is:

$$P\;{}_{\geqslant r}{\oplus}\;Q \mathrel{\widehat=} \sqcap\{(P\;{}_p{\oplus}\;Q) \mid r \leqslant p \leqslant 1\}\ . \tag{3}$$

It can be proved that $P\;{}_{\geqslant r}{\oplus}\;Q = (P\;{}_r{\oplus}\;Q) \sqcap P$.

3. Grover's quantum search algorithm

The search problem can be defined as: given an array f of 2^n bits containing a single 1, locate it - the solution is thus $f^{-1}(1)$. The complexity of the problem, measured in number of accesses to the array, is clearly $\Omega(2^n)$ for any classical algorithm in both the worst and average case. Grover's ingenious quantum algorithm [5] solves the problem with high probability and with only $O(\sqrt{2^n})$ accesses in both cases - a quadratic improvement over the classical method. However, the algorithm only succeeds with probability strictly less than 1, depending on the size of the array and number of iterations.

Grover's quantum algorithm can be expressed in qGCL as:

```
$\mathbf{var}\ \chi{:}q(\mathbb{B}^n),\ r{:}\mathbb{B}^n \bullet$
  $\mathbf{In}(\chi)⨾$
  $\mathbf{do}\ N$ times $\rightarrow$
      $\chi := T_f\chi⨾$
      $\chi := M\chi$
  $\mathbf{od}⨾$
  $\mathbf{Fin}(r,\chi)$
$\mathbf{rav}$
```

where function $f{:}\mathbb{B}^n \rightarrow \mathbb{B}$ is the array, transformation T_f between quregs is defined *pointwise* to invert χ about 0 if f holds and otherwise to leave it unchanged

$$\begin{aligned} &T_f{:}q(\mathbb{B}^n) \rightarrow q(\mathbb{B}^n) \\ &(T_f\chi)(x) \mathrel{\widehat=} (-1)^{f(x)}\chi(x) \ \ = \ \ -\chi(x) \lhd f(x) \rhd \chi(x)\,. \end{aligned} \tag{4}$$

Evidently T_f is unitary. Transform M inverts χ (pointwise) about its average

$$\begin{aligned} &M{:}q(\mathbb{B}^n) \rightarrow q(\mathbb{B}^n) \\ &(M\chi)(x) \mathrel{\widehat=} 2\,[\tfrac{1}{2^n}\textstyle\sum_{y:\mathbb{B}^n}\ \chi(y)] - \chi(x) \end{aligned} \tag{5}$$

and can be shown to be unitary and efficiently implementable [5]. The number of iterations N is a function of n of order $O(\sqrt{2^n})$.

Intuitively, the algorithm works in the following way: it starts by generating the uniform superposition of all the basis states (all the possible locations of the solution). At this point, a measurement of the quantum register would yield *any* basis state with uniform probability 2^{-n} - *i.e.* a uniformly random guess. The purpose of the loop is thus to stepwise increase the modulus of the amplitude of the solution state towards 1, so that a measurement will return the solution state with high probability. Remarkably, Grover's algorithm cannot be improved: $\Omega(2^n)$ accesses are provably needed by any quantum algorithm [1]. The algorithm works also in the case of multiple solutions: if f has s solutions then the algorithm's complexity is $O\left(\sqrt{\frac{2^n}{s}}\right)$. Boyer *et al.* [2] generalised the algorithm to the case that s is not known.

More formally, the working of the algorithm can be understood geometrically, as explained by Boyer *et al.* [2]. The initial uniform superposition $\nu \mathrel{\widehat{=}} \frac{1}{\sqrt{2^n}}\sum_i \delta_i$ can be written as:

$$\nu = \sqrt{\tfrac{2^n-s}{2^n}}\psi + \sqrt{\tfrac{s}{2^n}}\phi$$
$$\psi \mathrel{\widehat{=}} \tfrac{1}{\sqrt{2^n-s}}\sum_{j:\bar{S}}\delta_j \qquad \phi \mathrel{\widehat{=}} \tfrac{1}{\sqrt{s}}\sum_{i:S}\delta_i$$

where $(\bar{S})S$ is the set of the (non)solution indexes and $s \mathrel{\widehat{=}} \#S$. By choosing θ such that $\sin\frac{\theta}{2} = \sqrt{\frac{s}{2^n}}$ we have that $\nu = \psi\cos\left(\frac{\theta}{2}\right) + \phi\sin\left(\frac{\theta}{2}\right)$ and

$$(MT_f)\nu = \psi\cos\left(\frac{3}{2}\theta\right) + \phi\sin\left(\frac{3}{2}\theta\right)$$

(this can be easily proved noting that $T_f(a\psi + b\phi) = a\psi - b\phi$ and that $M = 2P_\nu - I$, where P_ν is the projector over the one-dimensional space spanned by ν). The effect of Grover's iteration body is thus a rotation by angle θ in the two-dimensional space spanned by ψ and ϕ. It can be shown that the state at the *k-th* iteration of the body is:

$$(MT_f)^k\nu = \psi\cos\left(\frac{2k+1}{2}\theta\right) + \phi\sin\left(\frac{2k+1}{2}\theta\right) \quad (6)$$

where by convention $(MT_f)^0 \mathrel{\widehat{=}} I$. In order to succeed with high probability we have to find a suitable k for which the state at the *k-th* iteration is close to ϕ. Without loss of generality we may suppose that $s \leqslant 2^{n-1}$, and therefore $\frac{\theta}{2} \geqslant \sin\frac{\theta}{2} = \sqrt{\frac{s}{2^n}}$. This implies that $k \leqslant \lfloor\frac{\pi}{4}\sqrt{\frac{2^n}{s}} - \frac{1}{2}\rfloor$, so we accordingly define:

$$N \mathrel{\widehat{=}} \left\lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{s}} - \frac{1}{2}\right\rfloor . \quad (7)$$

We note that the probability of success

$$p(k) \mathrel{\widehat{=}} \sin^2\left(\frac{2k+1}{2}\theta\right) \quad (8)$$

is a periodic function in the number of iterations k. This means that we will actually *decrease* the probability of success if we run too many iterations. Also, we note that the probability of success does not vary uniformly with k: when k is close to (multiples of) the optimal value N the probability varies very slowly. It can be shown that:

$$p(N) \geqslant \left(1 - \frac{s}{2^n}\right) .$$

In Appendix A we provide more details.

4. Derivation

Let $f{:}\mathbb{B}^n \to \mathbb{B}$ be our boolean array; we suppose that f contains at least one (possibly more) bit at 1. By defining the set of solutions $S \mathrel{\widehat{=}} f^{-1}(1)$, the search problem is just to return one element of S (the choice of which is arbitrary). To formally specify the problem we use Morgan's specification statement [8].

The specification $x : [pre, post]$ describes a computation which changes the (possibly empty) list of variables x in such a way that, if predicate *pre* holds on the initial state, termination is ensured in a state satisfying predicate *post* over the initial and final states; if *pre* does not hold, the computation aborts. If there is no value for x satisfying *post*, the computation will terminate achieving *post* miraculously (*i.e.* the validity of *post* is enforced). If $pre = true$ then we simply write $x : [post]$; the initial states in $post$ are denoted by a $_0$ subscript.

The problem is formally specified by the program G:

$$G \mathrel{\widehat{=}} \mathbf{var}\ r{:}\mathbb{B}^n \bullet\ r : [r{:}S] \ {}_{\geqslant\tilde{\epsilon}}{\oplus}\ r : [r{:}\bar{S}]\ \mathbf{rav} \quad (9)$$

where $\epsilon \mathrel{\widehat{=}} \frac{s}{2^n}$ and $\tilde{\epsilon} \mathrel{\widehat{=}} (1-\epsilon)$, S is a non-empty, proper subset of $\mathbb{B}^n$, $\bar{S} \mathrel{\widehat{=}} \mathbb{B}^n\backslash S$, and without loss of generality we suppose that $s \leqslant 2^{n-1}$. Informally, we are looking for an algorithm which returns a solution with probability at least $1-\epsilon$. We observe that G can be written in an equivalent way:

$$\begin{array}{lr}
G & \\
= & \text{definition of } {}_{\geqslant}{\oplus}\ (3) \\
(r : [r{:}S] \ {}_{\tilde{\epsilon}}{\oplus}\ r : [r{:}\bar{S}]) \sqcap r : [r{:}S] & \\
= & \text{Law P-2} \\
r : [r{:}S] \ {}_{\tilde{\epsilon}}{\oplus}\ (r : [r{:}\bar{S}] \sqcap r : [r{:}S]) & \\
= & \text{Law Spec5} \\
r : [r{:}S] \ {}_{\tilde{\epsilon}}{\oplus}\ r : [r{:}\bar{S} \vee r{:}S] & \\
= & \text{logic} \\
r : [r{:}S] \ {}_{\tilde{\epsilon}}{\oplus}\ r : [r{:}\mathbb{B}^n] &
\end{array}$$

where we omitted variable declarations for brevity. Informally, G requires a solution with probability exactly $\tilde{\epsilon}$, and with probability ϵ any outcome will do: we thus have again that G specifies an algorithm which is correct with probability at least $\tilde{\epsilon}$. The programming laws used are reported in Appendix B.

Before going into the derivation, we outline the main ideas and facts which we shall use. The following lemma shows how to implement a probabilistic choice by means of a specification on a qureg and a Finalisation (it is not a fully quantum program, hence the lemma's name).

Lemma 4.1. [Quantum semi-implementation] *For $A \subset \mathbb{B}^n$, $A \neq \emptyset$ and $p{:}[0,1]$ we have*

$$x : [x{:}A] \;{}_p\!\oplus\; x : [x{:}\bar{A}] \;\sqsubseteq\; \textbf{var}\ \chi{:}q(\mathbb{B}^n) \bullet \chi : [\|\chi\|_A^2 = p] \,;\, \textbf{Fin}(x, \chi)\ \textbf{rav}$$

where $\|\chi\|_A^2 \mathrel{\widehat=} \sum_{i:A} |\chi(i)|^2$.

Proof. We reason from the LHS:

$x : [x{:}A] \;{}_p\!\oplus\; x : [x{:}\bar{A}]$

$\sqsubseteq$ introduce nondeterminism (Law D-1)

$\sqcap[x := j \mid j \in A] \;{}_p\!\oplus\; \sqcap[x := k \mid k \in \bar{A}]$

$\sqsubseteq$ (2) and distributions u, v over $A, \bar{A}$

$[x := j \,@\, u_j \mid j \in A] \;{}_p\!\oplus\; [x := k \,@\, v_k \mid k \in \bar{A}]$

$=$ recombine probabilities (Law P-1)

$[x := i \,@\, p_i \mid i \in \mathbb{B}^n, p_i \mathrel{\widehat=} pu_i \lhd i \in A \rhd (1-p)v_i]$

$=$ u, v arbitrary, so p distribution over $\mathbb{B}^n | \sum_{i \in A} p_i = p$

$[x := i \,@\, p_i \mid \sum_{j \in A} p_j = p]$

$\sqsubseteq$ introduce and initialise qureg χ (Law D-2)

$[x, \chi := i, \delta_i \,@\, p_i \mid i \in \mathbb{B}^n, \sum_{j \in A} p_j = p]$

$\sqsubseteq$ data refinement with $p_i = |\chi(i)|^2$

$[x, \chi := i, \delta_i \,@\, |\chi(i)|^2 \mid \|\chi\|_A^2 = p]$

$=$ assumption as specification

$\chi : [\|\chi\|_A^2 = p] \,;\, [x, \chi := i, \delta_i \,@\, |\chi(i)|^2]$

$=$ definition of **Fin**

$\chi : [\|\chi\|_A^2 = p] \,;\, \textbf{Fin}(x, \chi)$

□

We now explain how to derive the iteration's body of the quantum search algorithm.

The general "trick" of a quantum algorithm is to unitarily (and efficiently) increasing the amplitude of the basis state corresponding to the solution of the problem, so that a measurement has a high probability of returning that state. Such a transformation must first of all be linear, so that we are looking for some operator $U{:}q(\mathbb{B}^n) \to q(\mathbb{B}^n)$ of the following shape:

$$U\chi(i) \mathrel{\widehat=} W\chi + c\chi(i) \qquad (10)$$

where $W{:}q(\mathbb{B}^n) \to \mathbb{C}$ and c a fixed complex such that $|c| = 1$. We now have to find out which W's are allowed by the unitarity constraint.

Lemma 4.2. *Suppose $U{:}q(\mathbb{B}^n) \to q(\mathbb{B}^n)$ is defined $U\chi(i) \mathrel{\widehat=} W\chi + c\chi(i)$, where $W{:}q(\mathbb{B}^n) \to \mathbb{C}$ is a linear application and $c \in \mathbb{C}$ fixed, $|c| = 1$. Then U is unitary iff all the coefficients of W are equal to w:*

$$\Im(w) \in [\tfrac{-\Im(c)-1}{d}, \tfrac{-\Im(c)+1}{d}]$$
$$\Re(w) = \frac{-\Re(c) \pm \sqrt{\Re(c)^2 - d^2\Im(w)^2 - 2d\Im(c)\Im(w)}}{d}$$

where $d = 2^n$.

Proof. Since $W\chi$ must be a scalar, then W must be a 1×2^n matrix, *i.e.* a row vector of elements w_i. By defining the operator $T\chi(i) \mathrel{\widehat=} W\chi$ we have that $U = T + cI$ and $T_{ij} = w_j$. We now reason:

U unitary

$\equiv$ definition of unitary and U

$(T + cI)^\dagger (T + cI) = I$

$\equiv$ linear algebra

$T^\dagger T + cT^\dagger + c^* T = 0$

$\equiv$ logic

$\forall i, j \bullet (T^\dagger T)_{ij} + cT^\dagger_{ij} + c^* T_{ij} = 0$

$\equiv$ linear algebra

$\forall i, j \bullet (\sum_k T^\dagger{}_{ik} T_{kj}) + cT^\dagger_{ij} + c^* T_{ij} = 0$

$\equiv$ definition of T

$\forall i, j \bullet dw_i^* w_j + cw_i^* + c^* w_j = 0$

$\equiv$ logic

$\forall i \bullet (\forall j \neq i \bullet dw_i^*(w_j - w_i) = -c^*(w_j - w_i) \wedge$
$cw_i^* = -c^* w_i - dw_i^* w_i)$

$\equiv$ logic (by contradiction)

$\forall i \bullet (\forall j \neq i \bullet w_i = w_j \wedge cw_i^* = -c^* w_i - dw_i^* w_i)$

Therefore we have proved that U is unitary iff all the coefficients w_i are equal (say w) and satisfy

$$cw^* = -c^* w - dw^* w\,.$$

We thus have to solve the equation

$$d(\Re(w)^2 + \Im(w)^2) + 2(\Re(c)\Re(w) + \Im(c)\Im(w)) = 0$$

where the unknowns are $\Re(w)$ and $\Im(w)$. Treating the latter as a constant we can solve the quadratic equation in the former:

$$\Re(w) = \frac{-\Re(c) \pm \sqrt{\Re(c)^2 - d^2\Im(w)^2 - 2d\Im(c)\Im(w)}}{d}.$$

However, it must be that

$$\Re(c)^2 - d^2\Im(w)^2 - 2d\Im(c)\Im(w) \geqslant 0$$

so $\Im(w) \in [\frac{-\Im(c)-1}{d}, \frac{-\Im(c)+1}{d}]$.

□

In the particular case of $\Im(w) = 0$ and $\Re(w) \neq 0$ (*i.e.* W a non-zero, real operator) we have that

$$w = -\Re(c)\frac{2}{d}.$$

If we also add the constraint that c must be real (*i.e.* $c = \pm 1$) we find that $w = \mp\frac{2}{2^n}$ and we can write $W = -\text{sign}(c)2m$ where $m{:}q(\mathbb{B}^n) \to \mathbb{C}$ is the average operator:

$$m\chi \mathrel{\widehat{=}} \frac{1}{2^n}\sum_{i:\mathbb{B}^n} \chi(i)\ .$$

We thus have proved the following corollary.

Corollary 4.3. *The only non-zero real operators W satisfying the unitarity of $U\chi(i) \mathrel{\widehat{=}} W\chi \pm \chi(i)$ are $\mp 2m$.*

The Grover's operator $M = 2m - I$ defined in (5) is therefore derived by imposing unitarity of $U\chi(i) \mathrel{\widehat{=}} W\chi - \chi(i)$ for W real. We observe that the solution $-M$ is also a valid Grover operator and could be used as well, since $M\chi$ is indistinguishable from $(-M)\chi = -M\chi$, *i.e.* no quantum measurement can distinguish them (in general, no measurement can distinguish between a state ξ and state $z\xi$, where z is a complex number of modulo one).

Note that for $M = 2m - I$ we have $MM = I$, and that is not much helpful for our purposes. In particular, we would like to increase the amplitude for states i such that $f(i)$ holds (because of unitarity that implies decreasing the amplitudes for the non-solution states). More specifically, we would like another transformation R such that

$$(MR\chi)(i) = \begin{cases} 2mR\chi + \chi(i) & \text{if } i \in S \\ 2mR\chi - \chi(i) & \text{if } i \notin S \end{cases}$$

but this is exactly the effect of transformation T_f defined by (4). So, we obtain in fact the operator

$$(2m - I)T_f = MT_f \tag{11}$$

which is of course the body of Grover's iteration.

We are ready to present the full derivation of the algorithm, starting from specification G defined in (9):

$$r : [r{:}S] \ {}_{\geqslant\tilde{\epsilon}}{\oplus} \ r : [r{:}\bar{S}]$$

$=$ definition of $P \ {}_{\geqslant r}{\oplus}\ Q$ (3)

$$\sqcap\{r : [r{:}S] \ {}_{p}{\oplus} \ r : [r{:}\bar{S}] \ \mid\ \tilde{\epsilon} \leqslant p \leqslant 1\}$$

$\sqsubseteq$ quantum semi-implementation (Lemma 4.1)

$$\sqcap\{\chi : [\|\chi\|_S^2 = p] \fatsemi \mathbf{Fin}(r,\chi) \ \mid\ \tilde{\epsilon} \leqslant p \leqslant 1\}$$

$=$ (1) and Law S-1

$$\sqcap\{\chi : [\ \|\chi\|_S^2 = p\ ] \ \mid\ \tilde{\epsilon} \leqslant p \leqslant 1\} \fatsemi \mathbf{Fin}(r,\chi)$$

$=$ semantics of specification

$$\chi : [\ \|\chi\|_S^2 \geqslant \tilde{\epsilon}\ ] \fatsemi \mathbf{Fin}(r,\chi)$$

$\sqsubseteq$ sequential composition (Law Spec2)

$$\begin{array}{l} \chi : [\ \chi = \nu\ ] \fatsemi \\ \chi : [\ \chi = \nu\ ,\ \|\chi\|_S^2 \geqslant \tilde{\epsilon}\ ] \fatsemi \\ \mathbf{Fin}(r,\chi) \end{array}$$

$\sqsubseteq$ implementation via **In** (Law Spec4)

$$\begin{array}{l} \mathbf{In}(\chi) \fatsemi \\ \chi : [\ \chi = \nu\ ,\ \|\chi\|_S^2 \geqslant \tilde{\epsilon}\ ] \fatsemi \\ \mathbf{Fin}(r,\chi) \end{array}$$

$\sqsubseteq$ $\chi = (MT_f)^N\nu \Rightarrow \|\chi\|_S^2 = p(N) \geqslant \tilde{\epsilon}$ (Lemma A.2)

$$\begin{array}{l} \mathbf{In}(\chi) \fatsemi \\ \chi : [\ \chi = \nu\ ,\ \chi = (MT_f)^N\nu\ ] \fatsemi \\ \mathbf{Fin}(r,\chi) \end{array}$$

$\sqsubseteq$ introduce k, sequential composition

$$\begin{array}{l} \mathbf{In}(\chi) \fatsemi \\ \chi, k : [\ \chi = \nu, k \leqslant N \wedge \chi = (MT_f)^k\nu\ ] \fatsemi \\ \chi, k : [\ k \leqslant N \wedge \chi = (MT_f)^k\nu, \\ \qquad\qquad \chi = (MT_f)^k\nu \wedge k = N\ ] \fatsemi \\ \mathbf{Fin}(r,\chi) \end{array}$$

$\sqsubseteq$ assignment (recall $(MT_f)^0 \mathrel{\widehat{=}} I$)

$$\begin{array}{l} \mathbf{In}(\chi) \fatsemi \\ k := 0 \fatsemi \\ \chi, k : [\ k \leqslant N \wedge \chi = (MT_f)^k\nu, \\ \qquad\qquad \chi = (MT_f)^k\nu \wedge k = N\ ] \fatsemi \\ \mathbf{Fin}(r,\chi) \end{array}$$

$\sqsubseteq$ introduce loop (Law Spec3)

$$\begin{array}{l} \mathbf{In}(\chi) \fatsemi \\ k := 0 \fatsemi \\ \mathbf{do}\ (k \neq N) \rightarrow \\ \quad \chi, k : [\ k < N \wedge \chi = (MT_f)^k\nu\ , \\ \qquad\qquad k \leqslant N \wedge \chi = (MT_f)^k\nu \wedge k > k_0] \\ \mathbf{od} \fatsemi \\ \mathbf{Fin}(r,\chi) \end{array}$$

$\sqsubseteq$ assignment (Law Spec4)

```
$\mathbf{In}(\chi)$;
$k := 0$;
$\mathbf{do}\ (k \neq N) \rightarrow$
    $k := k + 1$;
    $\chi := MT_f \chi$
$\mathbf{od}$;
$\mathbf{Fin}(r, \chi)$
=                                        notation
$\mathbf{In}(\chi)$;
$\mathbf{do}\ N$ times $\rightarrow$
    $\chi := MT_f \chi$
$\mathbf{od}$;
$\mathbf{Fin}(r, \chi)$
```

where in the last step we took the liberty to use an informal, though correct, argument.

5. Conclusions

The aim of this paper has been to try to apply standard formal methods techniques to quantum programs in an imperative language. In particular, we have used Grover's quantum search algorithm as a testbed. Starting from a rigorous specification of the search problem, we have refined it by means of standard programming laws and quantum laws until we have arrived at a feasible quantum program. The correctness of the final program is ensured by the correctness of the laws used.

Grover's algorithm works by iterating a fixed number of times (dependent on the problem size) a particular transformation. The implementation of the iteration's body as a quantum operator follows from very general assumptions on the operator' shape (see Lemma 4.2). Previous work showed that it is possible to derive the Deutsch-Jozsa's algorithm from its specification. However, that algorithm is simpler than Grover's, as it is essentially a two-step algorithm (no iteration is present).

We conclude that quantum algorithms can be successfully treated at a high level using standard formal methods. That means that we can benefit from of the extensive range of techniques already developed for standard programming, and we can reason at different levels of abstraction within the same framework. Furthermore, no verification of the resulting quantum implementation is necessary, since refinement preserve correctness.

6. Acknowledgements

The author wishes to thank Jeff Sanders for many useful discussions. This work has been supported by a *Marie Curie Outgoing International Fellowship* within the 6th European Community Framework Programme.

References

[1] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

[2] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46:493–505, 1998. arxiv.org/quant-ph/9605034.

[3] M. Butler and P. Hartel. Reasoning about Grover's quantum search algorithm using probabilistic *wp*. *ACM Transactions on Programming Languages and Systems*, 21(3):417–430, 1999.

[4] S. J. Gay. Quantum programming languages: survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006.

[5] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the 28th Annual Symposium on the Theory of Computing*, pages 212–219, 1996.

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

[7] A. McIver and C. C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[8] C. C. Morgan. *Programming from Specifications*. Prentice-Hall International, 1994.

[9] C. C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.

[10] J. W. Sanders and P. Zuliani. Quantum programming. In *MPC '00: Mathematics of Program Construction, Springer LNCS*, volume 1837, pages 80–99, 2000.

[11] P. W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *FOCS '94: Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, pages 20–22, 1994.

A. Grover's algorithm success probability

We recall that Grover's algorithm success probability after k iterations is $p(k) = \sin^2\left(\frac{2k+1}{2}\theta\right)$, with θ defined such that $\sin\frac{\theta}{2} = \sqrt{\frac{s}{2^n}}$, where s is the number of solutions (we assume $s \leqslant 2^{n-1}$).

Lemma A.1. For any k:$\mathbb{N}$

$$p(k) = (1 - \epsilon) - \cos k\theta \cos(k+1)\theta$$

where $\epsilon = \frac{s}{2^n}$.

Proof. The proof is simple algebra, but we report it for completeness. We first note that by definition we have that $\sin\frac{\theta}{2} = \sqrt{\epsilon}$ and therefore $\tilde{\epsilon} \mathrel{\widehat{=}} 1-\epsilon = \cos^2\frac{\theta}{2}$. We start reasoning from the definition of $p(k)$:

$$\begin{aligned}
&\sin^2(k\theta+\tfrac{\theta}{2})\\
=&\qquad\text{addition formula}\\
&(\sin k\theta\cos\tfrac{\theta}{2}+\sin\tfrac{\theta}{2}\cos k\theta)^2\\
=&\qquad\text{logic and definition of } \epsilon\\
&\tilde{\epsilon}\sin^2 k\theta+\epsilon\cos^2 k\theta+2\sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}\sin k\theta\cos k\theta\\
=&\qquad\text{basic trigonometry}\\
&\tilde{\epsilon}+\cos k\theta((2\epsilon-1)\cos k\theta+2\sin\tfrac{\theta}{2}\cos\tfrac{\theta}{2}\sin k\theta)\\
=&\qquad\text{definition of } \epsilon\text{, addition formula}\\
&\tilde{\epsilon}+\cos k\theta(\sin\theta\sin k\theta-\cos\theta\cos k\theta)\\
=&\qquad\text{addition formula}\\
&\tilde{\epsilon}-\cos k\theta\cos(k+1)\theta\,.
\end{aligned}$$

□

The following result was first presented by Boyer *et al.* [2], albeit via a slightly different argument.

Lemma A.2. We have that

$$p(N) \geqslant 1-\epsilon$$

where $\epsilon = \frac{s}{2^n}$ and $N = \lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{s}} - \frac{1}{2}\rfloor$.

Proof. By Lemma A.1 we have that

$$p(N) = (1-\epsilon) - \cos N\theta\cos(N+1)\theta\,.$$

We have to prove that $\cos N\theta\cos(N+1)\theta$ is negative. Since N was defined as the least integer such that $(N+\frac{1}{2})\theta = \frac{\pi}{2}$, then it must be that $(N+1)\theta \geqslant \frac{\pi}{2}$. Therefore $\cos(N+1)\theta$ is negative, while $\cos N\theta$ is positive. □

Remark. If we write $p(k) = (1-\epsilon) + V(k)$ where $V(k) \mathrel{\widehat{=}} -cosk\theta\cos(k+1)\theta$, we can study the behaviour of $p(k)$ by studying $V(k)$'s behaviour. In particular, $V(k)$ is negative and increasing for $k < N$, positive (possibly 0) for $k = N$, negative and decreasing for $k > N$ (and $k < 2N$). We also observe that $p(N) = 1-\epsilon = \frac{1}{2}$ when $s = 2^{n-1}$; $p(N) = 1$ for $s = 2^{n-2}$. A detailed study of Grover's algorithm performance can be found in [2].

B. Algebraic programming laws

We list a few programming and refinement laws which hold for pGCL programs; the semantic models adopted and proofs can be found in [6, 9]; more specification refinement laws can be found in [8], with an emphasis on probabilism in [7].

Law (D-1)**.** For datatype D we have

$$x : [x{:}D] \sqsubseteq \sqcap[x := i \mid i \in D]\,.$$

Law (D-2)**.** If e is any expression of type D then

$$\mathbf{var}\, x{:}D \sqsubseteq (\mathbf{var}\, x{:}D \bullet x := e)\,.$$

Law (P-1)**.** For $p, q, r{:}[0,1]$ we have

$$\begin{aligned}&(prg_1\ {}_p\!\oplus prg_2)\ {}_q\!\oplus\ (prg_3\ {}_r\!\oplus prg_4) =\\ &[prg_1 \mathbin{@} pq \mid prg_2 \mathbin{@} \tilde{p}q \mid prg_3 \mathbin{@} \tilde{q}r \mid prg_4 \mathbin{@} \tilde{q}\tilde{r}]\end{aligned}$$

where $\tilde{a} \mathrel{\widehat{=}} (1-a)$.

Law (P-2)**.** For $p{:}[0,1]$ we have

$$prg_1\,{}_p\!\oplus(prg_2 \sqcap prg_3) = (prg_1\,{}_p\!\oplus prg_2) \sqcap (prg_1\,{}_p\!\oplus prg_3).$$

Law (S-1)**.**

$$(prg_1 \sqcap prg_2) \mathbin{⨾} prg_3 = (prg_1 \mathbin{⨾} prg_3) \sqcap (prg_2 \mathbin{⨾} prg_3)\,.$$

Law (Spec1-strengthen postcondition)**.** If $post' \Rightarrow post$ then

$$x : [pre, post] \sqsubseteq x : [pre, post']\,.$$

Law (Spec2-sequential composition)**.** If neither mid nor $post$ contain initial variables then

$$x : [pre, post] \sqsubseteq x : [pre, mid] \mathbin{⨾} x : [mid, post]\,.$$

Law (Spec3-iteration)**.** If V is any integer-valued formula then

$$\begin{aligned}&x : [inv, inv \wedge \neg G] \sqsubseteq\\ &\quad \mathbf{do}\ G \to x : [inv \wedge G, inv \wedge (0 \leqslant V < V_0)]\ \mathbf{od}\end{aligned}$$

where neither inv nor G contain initial variables; V_0 is the expression obtained from V by substituting the initial state of x.

Law (Spec4-assignment)**.** *If* $(x = x_0) \wedge pre \Rightarrow post[x\backslash E]$ *then*

$$x : [pre, post] \sqsubseteq x := E\,.$$

The following law can be easily proved from the semantics of specification and nondeterministic choice (see Chapter 23 of [8]).

Law (Spec5-non determinism)**.**

$$\begin{aligned}&x : [pre, post] \sqcap x : [pre', post'] =\\ &\quad x : [pre \wedge pre', post \vee post']\,.\end{aligned}$$