# A New Algebraic Foundation for Quantum Programming Languages

Andrew Petersen and Mark Oskin
Department of Computer Science and Engineering
University of Washington

## ABSTRACT

*Quantum computation was first proposed two decades ago. Since then, we have chased the intrinsic parallelism that nature employs and have dreamed of bringing it to bear on our most intractable computational problems. However, despite our best efforts, methods for efficiently employing the power of quantum mechanics continue to elude us even as physicists make great strides toward quantum hardware. If this continues, we may well develop powerful quantum computers only to find that we have no way to use them.*

*When developing algorithms, quantum theorists use a formalism first used in the early twentieth century to describe physical systems. However, this formalism, a combination of linear algebra and a notation for describing individual states, is not sufficient for reasoning about quantum effects. When describing quantum computation, we are interested in the properties exhibited by specific states and the changes induced by applying some operation. Hence, we present a new algebraic formalism that supports abstractions for reasoning about quantum effects and indicates important quantum properties explicitly rather than focusing solely on describing a physical system.*

*The new quantum algebra has several attractive properties including explicit representation of important quantum properties, mechanisms for compactly describing and efficiently reasoning about large numbers of bits in a highly entangled state, and a descriptive representation of quantum operations. It is hardware independent and can be used as a notation for quantum computation or as the basis for a programming language.*

## 1. Introduction

Quantum computing offers both the opportunity to explore a new computational model and the hope that exponential computing power may be harnessed for use on currently intractable problems. So far, the exploration of quantum hardware technology has had several successes, including the entanglement of solid state qubits [1] and demonstrated 7-bit computers [2]. However, with few exceptions [3, 4], the dream of harnessing quantum computing for use on today's difficult problems has remained unfulfilled. It remains extremely difficult to reason about quantum effects and to develop quantum algorithms.

Why, then, are so few quantum algorithms known? There may be no more left to discover, but as we are still discovering new classical algorithms, this seems unlikely. It seems more plausible that we lack the tools necessary to effectively reason about quantum effects. After all, we have continued to use a notation developed nearly eighty years ago to represent physical systems in quantum mechanics. Quantum computation has different concerns, and in addition, it involves the manipulation of distinctly unfamiliar properties like entanglement, phase, and superposition. The current formalisms lack the necessary abstractions to manipulate these properties intuitively. Our goal is to develop a reasoning system that provides the tools necessary to reason effectively about quantum effects.

In this paper, we focus on first motivating and then introducing the elements of our algebra. We also provide a simple example that contrasts the traditional notation with our algebra and longer examples that highlight the algebra's expressiveness and demonstrate its ability to describe all major known algorithms. The major contributions of our work are fourfold. First, superposition and entanglement are represented explicitly. These two properties have no classical analog and provide part of the power of quantum computation. Second, we introduce a fundamental unit of phase. Phase has traditionally been treated as continuous, but a basic unit of phase simplifies reasoning about interference without sacrificing expressiveness. Third, we have developed a representation of quantum operations that indicate the properties the operation will introduce in the input. Finally, we include two mechanisms for compactly describing and reasoning about highly entangled states.

We begin by exploring the traditional quantum algebraic formalism and uncover some of its shortcomings as a reasoning system and basis for a language. In Section 3, we describe our algebra and contrast it to the traditional formalism. Section 4 contains a few examples including Grover's algorithm and the quantum Fourier transform. Section 5 explores related work, and finally, in Section 6, we summarize our findings and describe plans for future work.

## 2. The Traditional Formalism

Currently, a combination of linear algebra and Dirac notation [5] is used to reason about quantum algorithms and computations. This formalism can describe any quantum state or transformation, but we are more interested in its ability to facilitate reasoning about quantum states and effects. In this section, we describe the traditional formalism and explore its strengths and weaknesses as a quantum reasoning system.

### 2.1 Quantum States

A single quantum bit $\psi$ is represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ or, in the vector notation:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

The amplitudes $\alpha$ and $\beta$ are complex numbers such that $\alpha \cdot \alpha^* + \beta \cdot \beta^* = 1$, where $x^*$ is the complex conjugate of $x$. If either $\alpha$ or $\beta$ are zero, then $\psi$ is in a pure state. Otherwise, $\psi$ is said to be in a superposition of the two states. This use of amplitudes is problematic, since it combines the concepts of state and phase. When a system consists of two quantum bits, it is described by their cross product. For example, in the Dirac notation, a two-bit system can be described by $|\psi_1\psi_2\rangle = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle$. The number of possible states in the system continues to expand exponentially as additional bits are included. This leads to a problem when entanglement is encountered.

Entanglement is a key property that must be understood and used by algorithm designers and programmers, and when it is present, the system cannot be decomposed into a series of small, component vectors. For example, consider the following two systems:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

In the two vectors above, the system described by the vector on the left is not entangled, while that on the right is entangled. The vector on the left can be decomposed into component vectors which can be used to describe the system. However, the vector on the right cannot be decomposed into component vectors, so the system must be described by the larger vector. In big systems, this leads to a state explosion problem that hinders representation and reasoning, since a system of $n$ entangled bits must be described by a single vector of size $2^n$ rather than $n$ vectors of size 2.

The example above contains another subtle representation problem. The two vectors differ by a single swap of two values, yet only one of them represents a system that is entangled. While the entangled and unentangled states can be distinguished by decomposing the vector, this is not a trivial task as the size of the system increases. Thus, distinguishing whether a system contains entanglement can be difficult, which also hinders reasoning about the system.

## 2.2 Quantum Operations

In the traditional formalism, quantum operations are expressed as matrices. Multiplying an operator matrix by a state vector applies the operation to the system. For example, the application of a Hadamard ($H$) gate to the system $|\psi\rangle = |0\rangle$ is represented by:

$$H|\psi\rangle = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Unfortunately, in this approach, the representation of a gate changes as the number of bits entangled with the target bit increases. For example, the application of a Hadamard gate to the first bit of the system $|\psi_1\psi_2\rangle = |00\rangle + |11\rangle$ is represented by:

$$H|\psi_1\psi_2\rangle = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}\left(\tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}\right) = \tfrac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

A different representation of the gate is required to apply the Hadamard to the second bit of the same system. The transformation from the basic gate matrix to the one required for a larger state space is well-known, but it is, nevertheless, unintuitive even for small extensions. In addition, the space required to represent and store the extended operators is prohibitive for large state spaces. The combination of these two factors makes expressing computation on entangled states a non-trivial problem.

## 2.3 Example: EPR Generation

In practice, linear algebra is used to perform computations and describe state transformations, and the resulting state vectors are translated into Dirac notation as necessary. Alternatively, successive states, each in Dirac notation, can be presented with commentary describing the transformations performed at each step. In this section, we use the traditional formalism to present a procedure for creating EPR pairs.

The EPR pair, also known as the Bell state, is represented by the expression $|00\rangle + |11\rangle$ [6, 7]. It is a fundamental component of several important quantum effects including teleportation, which implements secure communication, and superdense coding, which allows two bits of information to be transmitted with one bit. The procedure for creating such pairs is as follows:

// Start with two bits in the zero state

$$|\phi_1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \; |\phi_2\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

// Apply a Hadamard gate to the first bit

$$H|\phi_1\rangle = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

// Apply a CNot on the second bit using the first bit as the control

$$|\phi_1\rangle \otimes |\phi_2\rangle = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$CNot|\phi_1, \phi_2\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}\left(\tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}\right) = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

This example highlights a few of the problems discussed earlier. For example, the final state is entangled and, in fact, has some very nice properties, but nothing about the state vector explicitly signals this. In addition, this procedure indicates how important the concept of order is, since the $CNot$ applies to a state vector that must be carefully constructed to perform the desired operation.

## 3. The Quantum Algebra

The quantum algebra is an alternative to the traditional formalism and seeks to remedy the problems encountered in Section 2. Those problems fit into two rough categories: clarity issues and problems dealing with entangled state spaces. The quantum algebra seeks to resolve these issues by making properties of qubits explicit and introducing mechanisms for reasoning about entangled states. Table 1 summarizes some of the interesting features of the algebra.

This section is organized into four parts. First, Section 3.1 describes how the algebra represents qubits and the states of systems. Section 3.2 details the declaration and application of quantum operations. Next, Section 3.3 describes the mechanisms included to deal with highly entangled states, and finally, we conclude in Sections 3.4 and 3.5 with an example and summary.

## 3.1 Qubits and System State

In the algebra, the basic unit is the bit. Unlike the traditional notation, where position or order is used to differentiate between bits, the algebra uses names, and the state of the bit is denoted by a superscript. Hence, a bit named $q$ in state $|0\rangle$ is denoted by $q^0$. Multi-bit variables can be represented similarly, and individual bits within that variable can be designated with subscripts. For example, an eight bit unsigned integer $j$ with value twenty-one is denoted by $j^{00010101}$, and $j$'s third bit is $j_3^0$.

### *3.1.1 Superposition and Entanglement*

Quantum bits are represented as the superposition of two classical bits, where superposition is denoted by the plus operator. Thus, a qubit $q$ in state $|0\rangle + |1\rangle$ is represented by $q^0 + q^1$. Multiple qubits can be linked via concatenation, so a two qubit system in state $|00\rangle + |01\rangle$ is written as $p^0q^0 + p^0q^1$. Commutative and associative laws apply to both the superposition and the concatenation operators and, unlike Dirac notation, require no extra comment since variables are named. Furthermore, the distributive property holds over these two operators, and its inapplicability indicates that the qubits involved are entangled. Hence, since the expression above can be rewritten as $p^0(q^0 + q^1)$, $p$ and $q$ are not entangled.

### *3.1.2 Weights*

So far, the quantum states presented have featured bits with equal probabilities of being in $|0\rangle$ or $|1\rangle$. To represent bits with unequal probabilities of being in the two base states, the algebra uses weights. Unlike the amplitudes used in the conventional formalism, which are constantly normalized and include phase information, the weights used by the new algebra are purely relative. For example, the state $\frac{\sqrt{3}}{2}|0\rangle + \frac{1}{2}|1\rangle$, in which the qubit is three times as likely to be found in $|0\rangle$ as in $|1\rangle$, could be written $3q^0 + q^1$. Both weights and phases may

**Table 1: Examples of important features of the algebra.**

| Description | Format | Examples |
|---|---|---|
| A state | $q^x$ | $r^0$ |
| A qubit | $\alpha q^0 + \beta q^1$ | $3q^0 + (-1)q^1$ |
| A 2-bit vector | | $p^0q^0 + p^1q^1$ |
| Procedure declaration | $name(par1, ...) \rightarrow label:\ pattern \Vert ...$ | $CNot(p,q) \rightarrow p^0q^x : p^0q^x$<br>$\Vert\, p^1q^0 : p^1q^1 \Vert\, p^1q^1 : p^1q^0$<br>$H(q) \rightarrow q^0 : q^0 + q^1 \Vert\, q^1 : q^0 - q^1$<br>$T(q) \rightarrow q^0 : q^0 \Vert\, q^1 : \chi^{\frac{2^n}{4}} q^1$ |
| Procedure application | $name(arg1, ...) \Rightarrow newState$ | // Let $q$ be in state $2q^0 - q^1$<br>$H(q) \rightarrow q^0 : q^0 + q^1 \Vert\, q^1 : q^0 - q^1\ on\ 2q^0 - q^1$<br>$\Rightarrow 2(q^0 + q^1) - (q^0 - q^1) = q^0 + 3q^1$ |
| Measurement | $Measure(q) \rightarrow q^0 : q^0 \wr \wr\, q^1 : q^1$ | // Let $q$ be in state $2q^0 - 3q^1$<br>$Measure(q) \Rightarrow 2q^0 \wr -3q^1 = 2q^0 \wr 3q^1$ |
| Symmetric entanglement | $e^+(p,q) \equiv p^0q^0 + p^1q^1 \equiv p \vert q$<br>$e^-(p,q) \equiv p^0q^0 - p^1q^1 \equiv p \vert -q$<br>$Measure(p \vert q) \equiv p^0q^0 \wr p^1q^1 \equiv p \wr q$ | $e^+(p) = p^0 + p^1$<br>$p \vert q \vert r = p^0q^0r^0 + p^1q^1r^1$ |
| Negative reasoning | $\overline{x}$ | $\overline{p \vert q} = p^0q^1 + p^1q^0$<br>$\overline{p \vert -q} = p^0q^1 + p^1q^0 + 2p^1q^1$ |

be normalized when they are at the outermost level of an expression. Otherwise, the distributive property is used to simplify expressions.

### *3.1.3 Phase*

Finally, in addition to a weight, every bit can have a phase. In quantum computations, phase is an important attribute, since it makes constructive and destructive interference possible. Traditionally, phase is modeled as a continuous complex value. For example, the expression $|0\rangle + e^{i\theta}|1\rangle$ involves a phase of $e^{i\theta}$. However, in all of the quantum algorithms we have seen, phase is only used in discrete units. Therefore, we use an elementary unit of phase $\chi = e^{\frac{i\pi}{2^n}}$ which is defined as a function of the number of bits $n$ in the system.

In many cases, however, even this much detail is unnecessary. Some algorithms require that only $\chi^0 = \chi^{2*2^n} = 1$ and $\chi^{2^n} = -1$ be used. In these cases, the use of $\pm 1$ is preferred. Addition and subtraction of like variables model constructive and destructive interference, respectively, and work as expected.

## 3.2 Operations

In the quantum algebra, all operations have a similar two-part form:

$$operation(parameter1, ...) \rightarrow label1 : pattern1 \parallel ...$$

The information to the left of the $\rightarrow$ operator includes the name of the operation and the names of its parameters. The information to the right of the arrow is the definition of the operation, which is presented as a list of labeled cases separated by double bars. The definition above is the only one needed in the quantum algebra. Since variables are named and operations are defined as lists of patterns, operations may be applied without modification to arguments in any size state-space; operations do not have to be redefined depending on the situation.

### *3.2.1 Applying Operations*

When an operation is applied, its form changes slightly, as the initial state and result of the operation are appended:

$$operation(argument1, ...) \rightarrow defn\ on\ state \Rightarrow newState$$

As shown above, the operator $\Rightarrow$ is used to indicate that the result follows, and while the definition and initial state may be supplied, either or both are often dropped to be concise.

Computation consists of three steps: consolidation, matching, and simplification. During consolidation, all of the states containing arguments are gathered and concatenated into a single, target expression. Usually, the resulting expression, which we have called the initial state, is solved for the arguments. In the next step, matching, each label is compared to the states in the target expression, and matching states are replaced by that label's pattern. Finally, in the last step, the expression is simplified and condensed, if possible. For example, let $p = p^0 + p^1$ and $q = q^1$. Then:

$$\begin{aligned}&CNot(p,q) \rightarrow p^0q^x : p^0q^x \parallel p^1q^0 : p^1q^1 \parallel p^1q^1 : p^1q^0\\&(Consolidate)\ on\ (p^0+p^1)(q^1) = p^0q^1 + p^1q^1\\&(Match)\ \Rightarrow p^0q^1 + p^1q^0\\&(Simplify)\end{aligned}$$

In the example above, no simplification could be performed because distributivity could not be applied to the expression. Hence, we know the two qubits are entangled.

### *3.2.2 Measurement*

Measurement is a special operation, as all primitive quantum operations except measurement are reversible. When a qubit is measured, any superposition collapses, and it becomes a classical bit. Whether it is observed in the $|0\rangle$ or the $|1\rangle$ state is determined by its respective weights. For example, the state $3q^0 + q^1$ has a $\frac{3}{4}$ probability of being observed in $q^0$.

Although a qubit is placed in a purely classical state after being measured, it is useful to consider both possibilities after measurement. Hence, we introduce the $\wr$ operator. This operator implies that the expressions on both of its sides are disjoint possibilities. Computation can be performed on an expression containing a $\wr$ with the constraint that the two possibilities cannot interact with one another. For example, let $p = p^0 + p^1$ and $q = q^0 \wr q^1$. Then:

$$\begin{aligned}&H(p) \Rightarrow (p^0 + p^1) + (p^0 - p^1) = 2p^0 = p^0\\&H(q) \Rightarrow q^0 + q^1 \wr q^0 - q^1\end{aligned}$$

The state of $p$ was simplified since the two possibilities could interact, but the state of $q$ could not be further modified, since the two possibilities are disjoint.

## 3.3 Abstractions for Entanglement

One of the major limitations of the traditional formalism is the inherent difficulty of dealing with the state-space explosion problem. Our algebra provides two methods for performing computation on groups of related states: symmetric entanglement and negative reasoning. Symmetric entanglement exploits symmetry in state between two quantum variables, and negative reasoning is the act of performing computations on the complement of a set of states. In both cases, computation can be performed on the compact form.

### 3.3.1 *Symmetric Entanglement*

Symmetric entanglement can be used on an expression containing two or more variables that will always be observed to have identical states. This occurs fairly frequently, as EPR pairs are often used in communication and error correction. This compaction mechanism can also be used when some portion of the entire state is symmetric, since superposition is associative. The definitions for the two types of symmetric entanglement are presented in Table 1. Formally, we define these states to be the result of a function $e^{\pm}$, which is derived from the EPR function, but we use the $|$ operator as semantic sugar to designate symmetric entanglement. In addition, we use the $\wr$ operator to designate a symmetrically entangled state which has been measured.

Computation on symmetrically entangled states is supported by the use of constraints. We define $pp \equiv p$ and $p\neg p \equiv \emptyset$ for any state $p$, and elaborate on the matching phase of computation. Instead of simply comparing each label to the states in the target expression, the label is concatenated to each state as a constraint. Any surviving states are then replaced. For example, let $p, q = p|q$. Then:

$$\begin{aligned}
&CNot(p,q) \rightarrow p^0q^x : p^0q^x \mid\mid p^1q^0 : p^1q^1 \mid\mid p^1q^1 : p^1q^0\\
&(Consolidate)\ on\ (p|q)\\
&(Match)\ becomes\ p^0q^x(p|q) + p^1q^0(p|q) + p^1q^1(p|q)\\
&\quad = p^0q^0 + p^1q^1 \Rightarrow p^0q^0 + p^1q^0\\
&(Simplify) = q^0(p^0 + p^1)
\end{aligned}$$

In this case, this process is equivalent to completely expanding the condensed state, but in more highly entangled states, this does not always occur. In addition, the constraints act as helpful cues for unrolling the state.

### 3.3.2 *Negative Reasoning*

Negative reasoning is used when the negation of a set of states is much smaller than the set of states itself. Let $U^n$ be the equally-weighted superposition of all possible n-bit states. For example, $U^2$ is $q_0^0q_1^0 + q_0^0q_1^1 + q_0^1q_1^0 + q_0^1q_1^1$. Then, we denote the negation of a set of n-bit states $x$ with $\overline{x}$ and formally define it to be $\overline{x} = U^n - x$. This definition of negation is very useful, as $U^n$ can be constructed by introducing any missing terms to the expression. For example, $q^0p^0 + q^0p^1 + q^1p^0 = q^0p^0 + q^0p^1 + q^1p^0 + (q^1p^1 - q^1p^1) = U^{pq} - q^1p^1 = \overline{q^1p^1}$.

Computation on a negation is performed in three steps. First, the patterns in all cases of the gate being applied are added together and multiplied by the superposition of all unaffected bits in the state-space. Second, the gate is applied to the states under the bar. Finally, the negation is replaced by the result obtained in the second step subtracted from the result from the first step. Further simplification may be performed as necessary. For example, let $p, q = \overline{p^1q^1}$. Then:

$$\begin{aligned}
&H(p) \rightarrow p^0 : p^0 + p^1 \mid\mid p^1 : p^0 - p^1\ on\ \overline{p^1q^1}\\
&\quad \Rightarrow ((p^0 + p^1) + (p^0 - p^1))(q^0 + q^1) - ((p^0 - p^1)q^1)\\
&\quad = (2p^0q^0 + 2p^0q^1) - (p^0q^1 - p^1q^1) = 2p^0q^0 + p^0q^1 + p^1q^1\\
&\quad = p^0q^0 + \overline{p^1q^0}
\end{aligned}$$

The last line of the example above was obtained by introducing the term $p^1q^0 - p^1q^0$ and then simplifying. The resulting expression is more compact than the earlier form and can be interpreted as being a superposition over the entire state space with the exception of $p^1q^0$. The state $p^0q^0$ has extra weight.

## 3.4 Example: EPR Generation

In Section 2.3, we demonstrated how the traditional notation can be used to describe the process of generating an EPR pair. In this section, we describe the same process using the new algebra.

```
$EPR(qubit\ p,\ qubit\ q)\ \{$
  $Zero(p) \Rightarrow p^0$
  $Zero(q) \Rightarrow q^0$
  $H(p) \rightarrow p^0 : p^0 + p^1 \mid\mid p^1 : p^0 - p^1\ on\ p^0 \Rightarrow p^0 + p^1$
  $CNot(p,q) \rightarrow p^0q^x : p^0q^x \mid\mid p^1q^0 : p^1q^1 \mid\mid p^1q^1 : p^1q^0$
    $on\ (p^0 + p^1)q^0 = p^0q^0 + p^1q^0 \Rightarrow p^0q^0 + p^0q^1 = p|q$
$\}\ \rightarrow p^xq^y : p|q$
```

This time, the notation explicitly describes the final state as entangled and even symmetric. In addition, order is no longer a concern, since variables are named. Furthermore, the procedure has been encapsulated as a new procedure named $EPR$, so it can be reused without describing its implementation.

## 3.5 Summary

The quantum algebra described in this section features several advantages over the traditional formalism. Properties like superposition, entanglement, and weight have been made more explicit; operations do not have to be redefined as state sizes increase; and mechanisms have been built into the notation that reduce the size of the state representation. We have demonstrated some of these advantages by defining a reusable procedure for creating EPR pairs.

The algebra is also complete and expressive, as it is capable of describing any legal quantum state or operation. The $H$, $CNot$, and $T$ gates have been declared in Table 1 and form a universal set, so the algebra can express any quantum state or algorithm [8]. We need not restrict ourselves to the primitive gates, however. More advanced operations can be defined by showing that a sequence of primitive or previously defined gates leads implements the desired result, and these operations are guaranteed to be legal quantum operations if their implementations are composed only of legal operations. Similarly, we can guarantee that a quantum state is legal if it is reached after a sequence of initializations and legal quantum operations.

## 4. Examples

In Section 3, we introduced the quantum algebra and presented a short example that highlights a few of the advantages of the new algebra. That example offers a point of comparison, as we implemented the same procedure in the traditional notation earlier in the paper. In this section, we present larger examples to display more of the features of the algebra and to introduce new syntax that we have found useful. The examples presented include many of the current major quantum algorithms. We refer the interested reader to Nielsen's text [9] for further explanations of the procedures presented as well as implementations in the traditional formalism.

## 4.1 The Deutsch-Jozsa Algorithm

The Deutsch-Jozsa (DJ) algorithm [10, 11] is the simplest of the major quantum algorithms. It rapidly determines, for a function $f$ with domain $D$ and range $\{0, 1\}$, if $f$ is balanced or constant. A function is constant if its range contains only a single element and balanced if exactly half of its domain map to one element of the range.

This example has two interesting features. First, we introduce support for loops. Second, the DJ algorithm utilizes phase transfer. In the classical world, control bits are not changed when they are used. However, in the quantum world, phase may be transferred to a control bit when a procedure is applied. To indicate the phase transfer in DJ as clearly as possible, we have chosen the $CNot$ operation on a two-bit domain as the function to be tested.

```
// The domain and ancilla bits
$qubit\ x[2] \Rightarrow x_1^0 x_2^0$
$qubit\ a \Rightarrow a^0$

// Placing the domain and ancilla into superposition
$forall\ x_i\ \{$
  $H(x_i) \rightarrow q^0 : q^0 + q^1 \mid\mid q^1 : q^0 - q^1$
$\}\ on\ (x_1^0)(x_2^0) \Rightarrow (x_1^0 + x_1^1)(x_2^0 + x_2^1)$

$Not(a) \rightarrow q^0 : q^1 \mid\mid q^1 : q^0\ on\ a^0 \Rightarrow a^1$
$H(a) \rightarrow q^0 : q^0 + q^1 \mid\mid q^1 : q^0 - q^1\ on\ a^1 \Rightarrow a^0 - a^1$

// Apply the function $f$
$CNot(x_1, a) \rightarrow p^0 q^x : p^0 q^x \mid\mid p^1 q^0 : p^1 q^1 \mid\mid p^1 q^1 : p^1 q^0$
  $on\ (x_1^0 + x_1^1)(a^0 - a^1) = x_1^0(a^0 - a^1) + x_1^1(a^0 - a^1)$
  $\Rightarrow x_1^0(a^0 - a^1) + x_1^1(a^1 - a^0) = (x_1^0 - x_1^1)(a^0 - a^1)$

// Take the domain out of superposition and measure
$forall\ x_i\ \{$
  $H(x_i) \rightarrow q^0 : q^0 + q^1 \mid\mid q^1 : q^0 - q^1$
  $Measure(x_i) \rightarrow q^0 : q^0 \wr\wr q^1 : q^1$
$\}\ on\ (x_1^0 - x_1^1)(x_2^0 + x_2^1) \Rightarrow (2x_1^1)(2x_2^0) = x_1^1 x_2^0$
```

The key to this process is in the application of $f$. The ancilla bit is in superposition with a negative phase component, and this negative phase is transferred to the control bit. We represent this transfer by factoring $-1$ from $(a^1 - a^0)$ so that the distributivity property applies. This phase transfer causes $x_1$ to be in the state $x_1^1$ when measurement occurs, so we know that $CNot$ is a balanced function.

Loops, as used above, are an ordered set of procedures meant to be applied to a sequence of inputs. The entire structure is treated much like a single procedure. The inputs are consolidated, and then the matching and simplification steps for all of the listed procedures are performed on each input bit. Other loop structures can be developed using a similar form.

## 4.2 The Quantum Fourier Transform

The quantum Fourier transform (QFT) is the key component of Shor's prime-factorization algorithm [3, 12]. In addition, since the QFT performs the same function as its classical analogue, it has many other potential applications. Nielsen defines the QFT as follows [9]:

$$|j\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2i\pi \frac{jk}{2^n}} |k\rangle$$

This example contains several new features of the algebra. The implementation of the QFT relies heavily on the manipulation of discrete units of phase, so this example is the first in which we use $\chi$, the basic unit of phase. In addition, to keep the description of the phase as clean as possible, we use a notational convention for writing classical decimal integers. We define $(0.q_1 q_2 ... q_n)_2 = \frac{q_1}{2^1} + \frac{q_2}{2^2} + ... + \frac{q_n}{2^n}$, where $q_i$ has value one if it is in state $q_i^1$ and zero if it is in state $q_i^0$.

We also use the $QFT$ to introduce the idea of reference values. These values are constants that maintain some quantum property like state or phase. They cannot be used during computation; instead, reference values are used to describe the new state of a qubit with respect to its former state.

```
$QFT(qubit\ q[n])\ \{$
  // Keep the original state for reference
  $ref\ a \equiv q$

  $for\ i = 1\ to\ n\ \{$
    //The Hadamard maps state to phase and vice versa
    $H(q_i) \rightarrow q_i^0 : q_i^0 + q_i^1 \mid\mid q_i^0 - q_i^1 = q_i^0 + \chi^{2*(0.a_i)_2} q_i^1$
      $\Rightarrow q_i^0 + \chi^{2*(0.a_i)_2} q_i^1$

    // Store the current phase for reference in the inner loop
    $ref\ m = \chi^{2*0.\hat{q}[i]_2}$
    $for\ j = 2\ to\ n - i\ \{$
      // Map state into the phase of the current qubit
      $R_j(j, q_{j+i}, q_i) \rightarrow q_i^0 : q_i^0 \mid\mid q_i^1 : \chi^{2*\frac{q_{j+i}}{2^j}} q_i^1$
        $\Rightarrow q_i^0 + m\chi^{2*\frac{a_{j+i}}{2^j}} q_i^1 = q_i^0 + \chi^{2*(0.a_i...a_{j+i})_2} q_i^1$
      $m = \chi^{2*(0.a_i...a_{j+i})_2}$
    $\} \Rightarrow q_i^0 + \chi^{2*(0.a_i...a_n)_2} q_i^1$

  // The state of all later bits should be stored in each bit's phase
  $\} \Rightarrow (q_1^0 + \chi^{2*(0.a_1...a_n)_2} q_1^1)(q_n^0 + \chi^{2*(0.a_n)_2} q_n^1)$
$\} \rightarrow (q_1^0 + \chi^{2*(0.a_1...a_n)_2} q_1^1)(q_n^0 + \chi^{2*(0.a_n)_2} q_n^1)$
```

The $QFT$ is an excellent example for discussing reference values because its output states must refer back to its input states. Reference values are used in two different ways to accomplish this. First, a reference value $a$ is created that holds the original state of the input qubits. These values are used to show that the state of the original qubits has been encoded into the phase of the transformed qubits. Second, a reference value $m$ is created to hold the current phase of the bit being transformed. This is useful to show how the increasingly smaller units of phase are being added so that no information is lost and is a prime example of how a single qubit can contain an infinite amount of data.

Reference values allow the quantum algebra to express state in terms of history. As noted before, however, they cannot be used in computation, since they violate several postulates of quantum mechanics. For example, we cannot copy the state of one qubit to another nor can we determine the exact phase of a qubit. Nevertheless, reference values can exhibit quantum properties including entanglement and superposition. This is especially apparent in the redefinition of the $Hadamard$ used in the $QFT$. The definition of Hadamard contains two cases. Both of these cases have been folded into a single case by using a reference value to describe the phase. This reference value can be in superposition, which would cause two different states, with different phases, to exist. This complexity is handled cleanly and automatically by the algebra.

## 4.3 Quantum Teleportation

Quantum teleportation [13] is the transfer of information from one qubit to another over unlimited physical distance. Classical information must also be transferred for the information transmitted to be useful, which keeps quantum teleportation from providing instantaneous communication. The process relies on the creation of EPR pairs, which was demonstrated earlier.

Our implementation of quantum teleportation defines a pair of procedures, $Send$ and $Receive$. We introduce new syntax in the parameters to both functions. To prove that quantum teleportation actually transmits the correct information, we fully specify the bit to be transferred using variables that catalog the weights and phase component. These variables are a form of reference variable, as introduced in Section 4.2. This syntax is very useful for writing and verifying procedures that act on some input.

On this and subsequent examples, we will omit, for the sake of brevity, the definition of a procedure unless it has not yet been introduced.

```
// p[2] must be an EPR pair
$Send(qubit\ a\ as\ ma^0 + (-1)^k na^1,\ qubit\ p_1\ as\ (p_1 \mid p_2))$ {
  // Encode the bit to be transmitted in half of the EPR pair
  $CNot(a, p_1) \Rightarrow ma^0(p_1 \mid p_2) + (-1)^k na^1\overline{(p_1 \mid p_2)}$
  $H(a) \Rightarrow m(a^0 + a^1)(p_1 \mid p_2) + (-1)^k n(a^0 - a^1)\overline{(p_1 \mid p_2)}$

  // Obtain classical values to transmit to the $Receive$ procedure
  $Measure(a) \Rightarrow m(a^0 \wr a^1)(p_1 \mid p_2) + (-1)^k n(a^0 \wr -a^1)\overline{(p_1 \mid p_2)}$
  $Measure(p_1) \Rightarrow m(a^0 \wr a^1)(p_1 \wr p_2) + (-1)^k n(a^0 \wr -a^1)\overline{(p_1 \wr p_2)}$
} $\rightarrow m(a^0 \wr a^1)(p_1 \wr p_2) + (-1)^k n(a^0 \wr -a^1)\overline{(p_1 \wr p_2)}$

// The input to $Receive$ must be the output of $Send$
$Receive((qubit\ a,\ qubit\ p_1,\ qubit\ p_2)$
    $as\ m(a^0 \wr a^1)(p_1 \wr p_2) + (-1)^k n(a^0 \wr -a^1)\overline{(p_1 \wr p_2)})$ {
  // Decode the transmitted state
  $CNot(p_1, p_2)$
    $\Rightarrow m(a^0 \wr a^1)(p_1^0 \wr p_1^1)p_2^0 + (-1)^k n(a^0 \wr -a^1)(p_1^0 \wr p_1^1)p_2^1$
    $= (p_1^0 \wr p_1^1)(m(a^0 \wr a^1)p_2^0 + (-1)^k n(a^0 \wr -a^1)p_2^1)$

  // Decode the transmitted phase
  $CZ(a, p_2) \rightarrow p^0q^x : p^0q^x \mid\mid p^1q^0 : p^1q^0 \mid\mid p^1q^1 : -p^1q^1$
    $\Rightarrow m(a^0 \wr a^1)p_2^0 + ((-1)^k n(a^0p_2^1 \wr a^1p_2^1))$
    $\Rightarrow (a^0 \wr a^1)(mp_2^0 + (-1)^k np_2^1)$
} $\rightarrow (p_1^0 \wr p_1^1)(a^0 \wr a^1)(mp_2^0 + (-1)^k np_2^1)$
```

The inputs to each of the procedures are quite complex, but the use of the $as$ structure to fully describe the state of the input bit has two benefits. First, as noted above, the developer is able to verify the correct operation of the procedures. The result of the $Send$ procedure is quite complex, but by the end of the $Receive$ procedure, verifying that the correct data was teleported is a matter of confirming that the final state of the receiver bit is the same as the initial state of the source bit. Second, the description of the input parameters acts as a constraint. Later readers of these procedures are immediately informed of the required attributes of the inputs.

## 4.4 Grover's Algorithm

Grover's algorithm [4] is an efficient quantum search. Given an unordered database with $n$ entries, Grover's searches it in $\sqrt{n}$ time. While not an exponential speedup over classical methods, this algorithm is an example where quantum computation offers significant advantages over classical approaches. Unlike Shor's algorithm, which is deterministic, Grover's algorithm is probabilistic. If correctly implemented, Grover's algorithm returns the correct answer with probability greater than 50%.

Grover's relies on several fairly complex procedures, including a phase flip operation and an oracle function, which we will define but not implement. The $PhaseFlip$ operation takes an array of $n$ qubits and introduces an additional $-1$ phase component to all but the zero state. It is defined as:

```
$PhaseFlip(q[n]) \rightarrow (q_n^{x_n}...q_i^1...q_1^{x_1}) : -(q_n^{x_n}...q_i^1...q_1^{x_1})$
  $\mid\mid (q_n^0...q_1^0) : (q_n^0...q_1^0)$
```

The $Oracle$ function introduces a $-1$ phase component to the state for which Grover's is searching. Therefore, a new Oracle must be built for each new search. We define our oracle to be:

```
// $\hat{p}$ refers to the solution the $Oracle$ is to flag
$Oracle(p[\underline{n}]) \rightarrow (\hat{p_n}...\hat{p_1}) : \underline{-}(\hat{p_n}...\hat{p_1})$
  $\mid\mid (p_n^{x_n}...\overline{\hat{p}_i}...p_1^{x_1}) : (p_n^{x_n}...\overline{\hat{p}_i}...p_1^{x_1})$
```

In the definition above and our implementation of Grover's algorithm, we use $\hat{p}$ to refer to the item for which we are searching. We assume that one copy of this item exists in the database. If more than one copy exists, Grover's algorithm converges more quickly; if not at all, then Grover's returns an incorrect answer.

We will implement Grover's algorithm in two parts. First, we will define a single $GroverIteration$. Then, implementing Grover's is a matter of repeating an iteration the required number of times.

```
$GroverIteration(p[n]\ as\ 1\overline{\hat{p}} + r\hat{p})$ {
  // Flag the desired entry with a negative phase
  $Oracle(p) \Rightarrow \overline{\hat{p}} - r\hat{p}$
    $= U^p - (r+1)\hat{p}$

  // Because of the negative phase, $\hat{p}$ does not leave superposition
  $forall\ p_i$ {
    $H(p_i)$
  } $\Rightarrow 2^n(p_n^0...p_1^0) - (r+1)U^{\hat{p}}$
    $= (2^n - (r+1))(p_n^0...p_1^0) - (r+1)\overline{(p_n^0...p_1^0)}$

  // Build up the weight on the correct entry
  $PhaseFlip(p) \Rightarrow (2^n - (r+1))(p_n^0...p_1^0) + (r+1)\overline{(p_n^0...p_1^0)}$
    $= -(2^n - 2(r+1))(p_n^0...p_1^0) + (r+1)U^{\hat{p}}$

  // Put the vector back into superposition
  $forall\ p_i$ {
    $H(p_i)$
  } $\Rightarrow (2^n - 2(r+1))U^p + 2^n(r+1)\hat{p} = 2^n((1 - \frac{r+1}{2^{n-1}})\overline{\hat{p}} + (r+1)\hat{p})$
} $\rightarrow (1 - \frac{r+1}{2^{n-1}})\overline{\hat{p}} + (r+1)\hat{p}$
```

Grover's algorithm is searching for a single correct answer in a search space. This is a perfect example of the type of situation for which negative reasoning was included. Hence, the procedure defined above places emphasis on partitioning the state space into incorrect solutions ($\overline{\hat{p}}$) and the single correct value ($\hat{p}$). The sum of these two partitions is the entire space ($U^p$), and much of the procedure is concerned with adding and subtracting instances of that correct state from both sides to simplify computation. For example, the first loop containing an $H$ would have been near impossible to express generally without the ability to express $U^p$.

One subtle feature of performing computations on groups of states, as is done above, is the treatment of weights. It appears at first glance that the weight on the single correct value $r$ may be much greater, initially, than the weight on the incorrect solutions. However, because the set of incorrect solutions contains so many more elements, their total weight, $(1 * (2^n - 1))$, is likely to be much higher than $r$. This point becomes especially apparent when the first set of $H$ operations are applied. When the entire superimposed state space is collapsed to a zero state, that zero state must be given weight equal to the sum of the weights of all the superimposed states. To see this, apply a $H$ to the state $p^0 + p^1$. The result is $2p^0$.

Now that a single iteration of Grover's algorithm has been defined, defining Grover's algorithm is a matter of preparing the state space and iterating the required number of times.

```
// An $Oracle$ must be built (and passed) to the Grover's procedure
$Grover(int\ n)$ {
  $qubitp[n] \Rightarrow p_n^0...p_1^0$
  $forall\ p_i$ {
    $H(p_i)$
  } $\Rightarrow (p_n^0 + p_n^1)...(p_1^0 + p_1^1)$
```

```
    $= \overline{\hat{p}} + \hat{p}$

  $for\ i = 1\ to\ (\sqrt{n} + 1)\ \{$
    $GroverIteration(p) \rightarrow (1 - \frac{r+1}{2^n-1})\overline{\hat{p}} + (r+1)\hat{p}$
  $\} \Rightarrow \overline{\hat{p}} + r\hat{p},\ r > (2^n - 1)$

  $Measure(p) \Rightarrow \overline{\hat{p}} \wr r\hat{p},\ r > (2^n - 1)$
$\} \rightarrow \overline{\hat{p}} \wr r\hat{p},\ r > (2^n - 1)$
```

Although a qubit's name is mentioned in the specification of the output, the value produced is classical since the qubit has been measured. Hence, the procedure for Grover's algorithm takes an integer, requires an $Oracle$ procedure, and produces a classical value. This is an excellent example of interaction between the classical and quantum worlds. In addition to requiring and returning purely classical values, this, and almost all other, quantum procedure relies entirely on $classical\ control$ to manipulate $quantum\ data$ [14].

Unfortunately, while convenient, the interaction between classical control and quantum data is problematic. For example, how was it determined that $\sqrt{2^n}$ is the required number of iterations? The most elegant proof is, unfortunately, geometric [9], but it can also be computed by noting that the ratio between the weights of the incorrect and correct solutions is $\left(2^n - 1 : (r+1) + \frac{r+1}{2^n-1}\right)$ after each iteration, where $r$ is the initial weight on the correct solution. In addition, the loop could be placed under quantum, rather than classical, control using reference values:

```
$ref\ r \equiv 1$
$while\ r \leq (2^n - 1)\ \{$
  $GroverIteration(p) \rightarrow (1 - \frac{r+1}{2^n-1})\overline{\hat{p}} + (r+1)\hat{p}$
  $r \equiv (r+1) + \frac{r+1}{2^n-1}$
$\} \Rightarrow \overline{\hat{p}} \wr r\hat{p},\ r > (2^n - 1)$
```

Unfortunately, the first half of this solution is not at all intuitive and the second is not amenable for use in a programming language, so part of our future work will focus on relating classical control to quantum data.

## 5. Related Work

The study of quantum computation has attracted researchers with a diverse set of backgrounds and objectives. As a result, many systems for describing quantum states and effects have been proposed, and in this section, we discuss the contributions these approaches made and the differences that separate them. This section is organized loosely in chronological order, so we begin with Dirac notation and quantum circuits, mention rules proposed for writing pseudo-code, and conclude with quantum assembly and higher level programming languages.

Dirac notation [5] is marked by the use of the ket ($|\rangle$) and is the standard method for describing states in quantum mechanics. It is a concise and expressive notation for describing the quantum state of a system but, traditionally, is not used to describe transformations. Instead, a vector notation, taken directly from linear algebra, supplements it. This formalism was first used early in the twentieth century and was adopted for the purpose of describing and modeling physical systems. It does so admirably but was not developed to describe quantum computation. Hence, Dirac and vector notation suffers from an exponential increase in representation size, a dearth of useful abstractions, and a reliance on ordering. Since it was not designed for the purpose, we believe this notational convention limits our ability to effectively reason about quantum concepts.

In the late 80's, Deutsch [15] proposed a circuit-based model for quantum computation that was further developed by Yao [16] and which led to the circuit representation for algorithms often used today. This representation is graphical and easily understood, and it is very useful for communicating the structure of algorithms and describing control flow. However, it lacks the ability to represent interactions between qubits like entanglement and phase transfer and does not scale well due to its graphical nature. Despite these drawbacks, the combination of the circuit and vector notations is currently the most popular method for describing quantum algorithms, as the two methods complement one another well. The circuit notation gives the reader an idea of the algorithm's structure, and the vector notation describes the states and transformations involved.

The first steps toward developing a notation for quantum computation amenable to programmers was Knill's attempt to standardize pseudo-code [17]. His work differentiates between classical and quantum registers (values) and enforces a separation between the two that is bridged by operations and measurement. In addition, his proposed pseudo-code provides explicit support for reversible functions and conditional operations. Like the traditional formalism on which it is based, however, his proposed quantum pseudo-code does not provide any clear way to indicate that quantum variables are entangled. Furthermore, it does not convey any sense of the state of the variables being manipulated.

Bernhard Ömer developed the first high-level quantum programming language, QCL [18, 19]. QCL is an imperative language with a few of the usual programming constructs (procedures, arrays, etc.) and has been built into an interpreted environment. A universal set of basic gates is provided, and facilities exist for building new operators and procedures. In addition, commands are provided to access the current state of the simulated system, which aids development. Unfortunately however, QCL is built upon the traditional base of Dirac and vector notation, so the same caveats apply.

At about the same time as QCL was being developed, Stephen Blaha took an algebraic approach and developed an assembly and a C-like language [20]. The algebra he uses is based on harmonic oscillators to relate it to superstring theory, and the languages he develops are useful primarily as tools for investigating theories in physics. As such, they do not provide support for designing or reasoning about quantum algorithms.

Later, Sanders and Zuliani used a probabilistic guarded-command language to create a high-level, imperative quantum language called qGCL [21, 22]. Their language is based around the invocation of procedures that consist of three phases: initialization, evolution, and finalization. Quantum variables are declared and placed into a state of superposition in the initialization phase, and they are forced out of superposition (measured) during the finalization stage. Variables are represented as a mapping from n-bit vectors to complex numbers and, during the evolution phase, are manipulated by a series of unitary transformations, which are concisely represented as mathematical functions that alter variable mappings. A refinement calculus is provided to verify that a given program satisfies its specification. Because of its compact representation, qGCL avoids the state explosion problem encountered when matrices are used to represent states and operations. However, it is difficult to relate their procedures with a series of actual quantum operations, and during the evolution phase, when computation is actually being performed, it is difficult to access and reason about the states of variables. As designed, qGCL is more a method for formally verifying the correctness of algorithms than a notation to aid in understanding, building, or implementing them.

Most recently, Bettelli et al. developed an extension for C++ to support the ability to do quantum computations in a classical framework [23]. They limit their work by targeting a specific architecture in which a classical core has the ability to manipulate a "quantum co-processor," so one of the advantages of their system is a clear separation between classical and quantum operations. In addition, their language extension introduces the idea of operations as data objects, rather than as functions, which allows for easy composition and optimization of operators during run-time. Although we have not attempted it, we believe our system can provide a similar benefit, as an algebraic simplifier may be used for optimization.

Finally, Peter Selinger recently released his work on a simple, graph-based programming language for quantum computation called BlockQPL [14]. It is based on complete partial orders and, like previous language attempts, relies on a relaxed form of Dirac notation and the traditional matrix notation to describe quantum state. Selinger combines the graph-based approach with the traditional formalism to mitigate the weaknesses of both. However, since BlockQPL relies on flowchart-like graphs to describe control-flow, the approach is still sensitive to state explosion effects and scaling problems. Nevertheless, Selinger's language includes several advances not present in earlier efforts. BlockQPL is functional, in contrast to the imperative languages designed recently by other groups. In addition, it is typed and has a developed syntax and formal semantics. Most importantly, Selinger clearly separates the quantum and classical domains.

## 6. Conclusions

We have introduced a new algebraic foundation for describing and reasoning about quantum computations. The goal of our algebra is to simplify the process of reasoning about quantum systems by explicitly identifying quantum properties like superposition and entanglement and providing mechanisms for reasoning about groups of possible quantum states. Two such mechanisms are provided in this paper: symmetric entanglement and negative reasoning. Both of these methods provide the ability to compactly represent exponential state spaces and to perform computations on the compact representations. In addition, we introduce the notion of a base unit of phase, which eases reasoning about interference. Finally, our algebra represents quantum gates in a manner independent of the system to which it is being applied. This allows gates to be defined by a single expression and to be applied to highly entangled states without modification.

However, this algebra is merely the basis for future work in quantum programming languages. While it is an attempt to make the representation of quantum effects more intuitive, the algebra does not provide an obvious solution to the problem of creating the high-level abstractions necessary in a language. Therefore, we intend to extend this work in three different directions. First, we will explore languages based on this algebra and build an environment, including a compiler and simulated quantum device, for designing algorithms that uses it. Second, we intend to collaborate with groups designing quantum hardware to evaluate hardware architectures and language styles. Third, we will explore new quantum algorithms with an emphasis on error correction and will apply our formalisms to the task of developing more realistic quantum noise models.

Our overall goal is to stimulate progress in the nebulous area of "quantum software." We have chosen to contribute by developing an algebra that supports reasoning about quantum effects and transformations, rather than simply describing quantum states and operations. We hope further progress in this area will contribute to the creation of intuitive, high-level quantum programming languages and aid in increasing knowledge of quantum concepts and the development of new quantum algorithms.

**Acknowledgements** We would like to thank our sponsors, DARPA QuIST and the NSF nano-technology initiative. In addition, we would like to thank David Notkin and Craig Chambers for reading earlier versions of this paper.

## 7. REFERENCES

[1] "NEC and RIKEN realize world's first quantum entanglement in a 2-bit solid-state device." Press Release, February 2003.

[2] "Los Alamos scientists make seven bit quantum leap." Press Release, March 2000.

[3] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. $35^{th}$ Annual Symposium on Foundations of Computer Science*, (Los Alamitos, CA), p. 124, IEEE Press, 1994.

[4] L. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. $28^{th}$ Annual ACM Symposium on the Theory of Computation*, (New York), pp. 212–219, ACM Press, 1996.

[5] P. Dirac, *The Principles of Quantum Mechanics*. Clarendon Press, 3rd ed., 1947.

[6] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?," *Physical Review*, vol. 41, 1935.

[7] J. S. Bell, "On the Einstein-Podolsky-Rosen paradox," *Physics*, vol. 1, no. 3, 1964.

[8] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, "On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for Shor's basis," in *Proc. 40th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 1999.

[9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, UK: Cambridge University Press, 2000.

[10] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London A*, vol. 439, pp. 553–558, 1992.

[11] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, "Quantum algorithms revisited," *Proceedings of the Royal Society of London A*, vol. 454, p. 339=354, 1998.

[12] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comp.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[13] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. Wootters, "Teleporting an unknown quantum state via dual classical and EPR channels," *Phys. Rev. Lett.*, vol. 70, pp. 1895–1899, 1993.

[14] P. Selinger, "Towards a quantum programming language." http://quasar.mathstat.uottawa.ca/ selinger/papers.html, November 2002.

[15] D. Deutsch, "Quantum computational networks," *Proceedings of the Royal Society of London*, vol. 425, no. 1868, pp. 73–90, 1989.

[16] A. C. Yao, "Quantum circuit complexity," *Proc. of the 34th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 352–361, 1993.

[17] E. Knill, "Conventions for quantum pseudocode." Unpublished, June 1996.

[18] B. Ömer, "A procedural formalism for quantum computing," Master's thesis, Technical University of Vienna, 1998.

[19] B. Ömer, "Quantum programming in QCL," Master's thesis, Technical University of Vienna, 2000.

[20] S. Blaha, *Cosmos and consciousness*. 1stBooks Library, 1998.

[21] J. Sanders and P. Zuliani, "Quantum programming," tech. rep., Oxford University, 1999.

[22] P. Zuliani, *Quantum Programming*. PhD thesis, St. Cross College of the University of Oxford, 2001.

[23] S. Bettelli, L. Serafini, and T. Calarco, "Toward an architecture for quantum programming," *To appear in the European Physical Journal*, 2002.