

QUANTUM PROGRAMMING

by

Anya Tafliovich

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Copyright © 2004 by Anya Tafliovich

Abstract

Quantum Programming

Anya Tafliovich

Master of Science

Graduate Department of Computer Science

University of Toronto

2004

The subject of this work is quantum programming — the study of developing of programs intended for execution on a quantum computer. We look at programming in the context of formal methods of program development, or programming methodology. Our work is based on probabilistic predicative programming, a recent generalisation of the well-established predicative programming. It supports the style of program development in which each programming step is proven correct as it is made. We inherit the advantages of the theory, such as its generality, simple treatment of recursive programs, and time and space complexity. Our theory of quantum programming provides tools to write both classical and quantum specifications, develop quantum programs that implement these specifications, and reason about their comparative time and space complexity all in the same framework.

Acknowledgements

I would like to thank my supervisor Eric C.R. Hehner for advice, help, and support during the past two years.

I thank the organisers of 2003 PIMS-MITACS Summer School on Quantum Information Science for introducing me to Quantum Computing.

My graduate studies at the University of Toronto were financially supported by Ontario Graduate Scholarship Program and Natural Sciences and Engineering Research Council of Canada.

Contents

1	Introduction	1
2	Related Work	4
3	Introduction to quantum computation	5
3.1	Basics from linear algebra	5
3.1.1	Dirac notation	6
3.1.2	Vector spaces	6
3.1.3	Operators	8
3.1.4	Tensor product	9
3.1.5	Computational basis	10
3.2	Postulates of quantum mechanics	11
3.3	A quantum algorithm	15
4	Introduction to probabilistic predicative programming	18
4.1	Predicative programming	18
4.1.1	Specifications	19
4.1.2	Specification notation	19
4.1.3	Refinement	21
4.1.4	Bunches	21
4.1.5	Functions	21

4.1.6	Quantifiers	22
4.1.7	Program development	22
4.1.8	Time and space	23
4.2	Probabilistic predicative programming	24
4.2.1	Probabilistic specifications	25
4.2.2	Developing probabilistic programs	27
4.2.3	Partial probabilistic specifications	31
5	The quantum system	32
5.1	The quantum state	32
5.2	Composing quantum systems	33
5.3	Unitary transformations	34
5.4	Measurement	36
6	Quantum programming	38
6.1	A quantum program	38
6.2	Fair coin	39
6.3	Deutsch’s problem	40
6.4	Deutsch-Josza problem	42
6.5	Grover’s search	46
6.6	Computing with Mixed States	54
7	Conclusion and Future Work	60
	Bibliography	63
	A Notations	65
	B Refinement Laws	68

C	Proofs omitted from Chapter 6	70
C.1	A classical solution for Deutsch-Jozsa problem	70
C.2	Deutsch-Jozsa problem with time restriction	73

Chapter 1

Introduction

Modern physics is dominated by concepts of quantum mechanics. Today, over seventy years after its recognition by the scientific community, quantum mechanics provides the most accurate known description of nature's behaviour. Surprisingly, the idea of using the quantum mechanical nature of the world to perform computational tasks is very new, less than thirty years old. Quantum computation and quantum information is the study of information processing and communication accomplished with quantum mechanical systems. In recent years the field has grown immensely. Scientists from various fields of computer science have discovered that thinking physically about computation yields new and exciting results in computation and communication. There has been extensive research in the areas of quantum algorithms, quantum communication and information, quantum cryptography, quantum error-correction, adiabatic computation, theoretical quantum optics, and the very new quantum game theory. Experimental quantum information and communication has also been a fruitful field. Experimental quantum optics, ion traps, solid state implementations and nuclear magnetic resonance (NMR) all add to the experimental successes of quantum computation.

The subject of this work is quantum programming — the study of developing programs intended for execution on a quantum computer. We assume a model of a quan-

tum computer proposed by Knill [10]: a classical computer with access to a quantum device that is capable of storing quantum bits, performing certain operations and measurements on these bits, and reporting the results of the measurements.

We look at programming in the context of formal methods of program development, or programming methodology. This is the field of computer science concerned with applications of mathematics and logic to software engineering tasks. In particular, the formal methods provide tools to formally express software specifications, prove correctness of implementations, and reason about various properties of specifications (e.g. implementability) and implementations (e.g. time and space complexity). Today formal methods are successfully employed in all stages of software development, such as requirements elicitation and analysis, software design, and software implementation.

In this work the theory of quantum programming is based on probabilistic predicative programming, a recent generalisation of the well-established predicative programming [7, 8], which we deem to be the simplest and the most elegant programming theory known today. It supports the style of program development in which each programming step is proven correct as it is made. We inherit the advantages of the theory, such as its generality, simple treatment of recursive programs, and time and space complexity. Our theory of quantum programming provides tools to write both classical and quantum specifications, develop quantum programs that implement these specifications, and reason about their comparative time and space complexity all in the same framework.

The rest of this work is organised as follows. Chapter 2 is a brief outline of related work. Chapter 3 is the introduction to quantum computation. It assumes that the reader has some basic knowledge of linear algebra and no knowledge of quantum computing. Chapter 4 contains the introduction to probabilistic predicative programming. The reader is assumed to have some background in logic, but no background in programming theory is necessary. The contribution of this work is Chapters 5 and 6.

Chapter 5 defines the quantum system and chapter 6 introduces programming with the quantum system. Chapter 6 contains several well-known problems, their classical and quantum solutions, and their formal comparative time complexity analyses. Chapter 7 states conclusions and outlines directions for future research.

Chapter 2

Related Work

The first quantum programming language, **QCL**, was introduced by Ömer [13]. It contains many useful high-level features and embeds a classical sub-language. Somewhat similar to **QCL** is the work of Bettelli *et al.* [2], who proposed a quantum extension of the language **C++**. They also proposed ways to optimise representations of quantum operators at run-time. Perhaps the closest to our work is the work of Sanders and Zuliani [14]. They propose a quantum programming language **qGCL**, which is the extension of probabilistic **pGCL**, which, in turn, builds upon Dijkstra's **GCL** (Guarded Command Language). This is the first attempt at systematic quantum program development and verification. Finally, Selinger [15] proposed a functional quantum programming language, which is guaranteed to be run-time error-free and represents programs with quantum flow charts.

Chapter 3

Introduction to quantum computation

This chapter explains the basic concepts of quantum computation. We start with a brief reminder of linear algebra, proceed with the introduction of basic principles of quantum computation, and finish with the analysis of a simple, but illustrative example of quantum algorithms. For a more thorough introduction to the field of quantum computation the reader is referred to [12]. This chapter does not discuss such interesting and important subjects as quantum communication or quantum cryptography. In an attempt to keep the introduction short, we exclude anything non-necessary for understanding the analyses of the several quantum algorithms presented in later chapters.

3.1 Basics from linear algebra

In this section we provide a brief overview of the notions from linear algebra necessary for understanding the basics of quantum computation. We assume the reader has some background in first and second-year linear algebra. The purpose of this section is to remind the reader of some basic facts about complex vector spaces and introduce the algebraic notations used in quantum mechanics. The reader with insufficient background in linear algebra is referred to [11] for a more detailed introduction.

The vector spaces considered will all be finite-dimensional complex vector spaces that are of interest in quantum computation and quantum information. In this section, whenever we refer to a vector space, finite dimensionality is implied.

3.1.1 Dirac notation

The *Dirac notation*, invented by Paul Dirac, is often used in quantum mechanics because of the advantages it offers. In this notation a vector v (a column vector by convention) is written inside a *ket*: $|v\rangle$. The dual vector (defined later) of $|v\rangle$ is $\langle v|$, written inside a *bra*. The inner products are then *bra-kets* $\langle v|w\rangle$. For n -dimensional vectors $|u\rangle$ and $|v\rangle$ and m -dimensional vector $|w\rangle$, the value of the inner product $\langle u|v\rangle$ is a scalar and the outer product operator $|v\rangle\langle w|$ corresponds to an m by n matrix. The Dirac notation clearly distinguishes vectors from operators and scalars, and makes it possible to write operators directly as combinations of bras and kets.

3.1.2 Vector spaces

Let \mathbb{C} be the set of complex numbers. The complex conjugate of $z \in \mathbb{C}$ is denoted by z^* . The i^{th} element of a vector $|v\rangle$ is denoted v_i . An *inner product* on a vector space V is a function $\langle \cdot | \cdot \rangle : V \times V \rightarrow \mathbb{C}$, linear in the second argument, such that for vectors $|v\rangle, |w\rangle \in V$

1. $\langle v|w\rangle = \langle w|v\rangle^*$
2. $\langle v|v\rangle \geq 0$ with equality if and only if v is a zero vector.

A *Hilbert space* (written \mathcal{H}) is defined as an inner product space (a vector space with an inner product defined on it) that is complete with respect to the norm defined by the inner product. In the case of finite dimensionality, a Hilbert space is equivalent to an inner product space.

The *dual* vector space of a Hilbert space \mathcal{H} , written \mathcal{H}^* , is defined as a set of linear maps $\mathcal{H} \rightarrow \mathbb{C}$, such that $\langle w| \in \mathcal{H}^*$ if $\langle w| : |v\rangle \rightarrow \langle w|v\rangle$, the inner product of $|w\rangle \in \mathcal{H}$ and $|v\rangle \in \mathcal{H}$. In the matrix representation, $\langle w|$, called a *dual vector* of a column vector $|w\rangle$, is obtained by taking the corresponding row vector and then taking the complex conjugate of its every element. In this setting, the inner product of two column vectors $|v\rangle$ and $|w\rangle$ is simply standard matrix multiplication of the row vector $\langle v|$ and the column vector $|w\rangle$: $\langle v||w\rangle$ or $\langle v|w\rangle$. There is no need to have different notations for the action of the dual operator and the corresponding matrix multiplication, since the two are equivalent.

If $|v\rangle$ and $|w\rangle$ are vectors in the inner product spaces V and W , with dimensions n and m , respectively, then the *outer product* $|w\rangle\langle v|$ is defined as the operator from V to W with the following action:

$$(|w\rangle\langle v|)(|u\rangle) = |w\rangle\langle v|u\rangle$$

The expression $|w\rangle\langle v||u\rangle$ has two potential meanings: the operator $|w\rangle\langle v|$ acting on $|u\rangle \in V$, and scalar multiplication of $\langle v|u\rangle$ and $|w\rangle$. These two meanings coincide, which is made clear by the notation. In matrix representation, the action of the operator $|w\rangle\langle v|$ corresponds to premultiplication by the m by n matrix obtained by vector multiplication of the column vector $|v\rangle$ and the row vector $\langle w|$.

The Hilbert space of our interest is the space \mathbb{C}^{2^n} of 2^n -dimensional complex vectors, with the inner product $\langle \cdot | \cdot \rangle : \mathbb{C}^{2^n} \times \mathbb{C}^{2^n} \rightarrow \mathbb{C}$, defined by

$$\langle v|w\rangle = \sum_{i: 0, \dots, 2^n} v_i^* \times w_i$$

For the sake of consistency, we use this quantifier notation throughout this work. For now the reader should understand this notation as “the sum as i varies from (including) 0 to (excluding) 2^n of $v_i^* \times w_i$ ”. Sometimes we will omit the values that i can take as it varies, if they can be understood from the context. The reasons for using this quantifier notation will become clear in Chapter 4.

The *norm* of $z \in \mathbb{C}$ is written $|z|$ and is defined by $|z| = \sqrt{\langle z|z \rangle}$. If $|z| = 1$, then z is called a *unit vector*. A *basis* is a set of linearly independent vectors that span the vector space. Vectors $|v\rangle$ and $|w\rangle$ are *orthogonal* if $\langle v|w \rangle = 0$. A set of vectors is *orthonormal* if each pair of distinct vectors are orthogonal and each vector is unit vector.

3.1.3 Operators

An operator A defined on a vector space V is *linear* if for any two vectors $|v\rangle, |w\rangle \in V$ and any scalar z ,

$$A(|v\rangle + |w\rangle) = A|v\rangle + A|w\rangle \quad \text{and} \quad A(z \times |v\rangle) = z \times (A|v\rangle)$$

If A is a linear operator on a Hilbert space \mathcal{H} , then there exists a unique linear operator A^\dagger on \mathcal{H} , such that for all vectors $|v\rangle, |w\rangle \in \mathcal{H}$,

$$(A^\dagger|w\rangle)^\dagger|v\rangle = \langle w|A|v\rangle, \quad \text{or simply} \quad \langle v|A^\dagger|w\rangle^* = \langle w|A|v\rangle$$

The operator A^\dagger is called the *adjoint* of A . In matrix representation, A^\dagger corresponds to the Hermitian conjugate of A . If $A^\dagger = A$, then A is called a *Hermitian* operator. If $A^\dagger A = A A^\dagger$, then A is called a *normal* operator. If $A^\dagger A = I$, where I is the identity operator on \mathcal{H} , then A is called a *unitary* operator. If W is a k -dimensional subspace of a d -dimensional vector space V , then there is an orthonormal basis $|v_0\rangle, \dots, |v_{d-1}\rangle$ for V , such that $|v_0\rangle, \dots, |v_{k-1}\rangle$ is an orthonormal basis for W . The operator $P = \sum_{i=0}^{k-1} |v_i\rangle\langle v_i|$ is called a *projector* on W .

The following several linear algebra results are important in quantum mechanics. Any projector is a Hermitian operator. Any Hermitian operator is normal. The Spectral Decomposition Theorem tells us that an operator is normal if and only if it is diagonalisable. This implies that a normal operator A on a Hilbert space \mathcal{H} can be written as $A = \sum_i \lambda_i \times P_i$, where λ_i are the eigenvalues of A and P_i is the projector onto the λ_i eigenspace of A . These projectors satisfy what are called the *completeness relation*,

$\sum_i P_i = I$, and the *orthonormality relation*, $P_i P_j = \delta_{ij} P_i$, where δ_{ij} is 1 if $i = j$, and 0 otherwise. Any unitary operator is normal, and therefore has a spectral decomposition. If A is normal, the eigenvalues of A are real. Unitary operators preserve inner products between vectors. If A is unitary, then all eigenvalues of A have modulus 1. A unitary operator has a unique inverse, which is also a unitary operator.

3.1.4 Tensor product

The *tensor product* is a way of combining vector spaces to form larger vector spaces. If V and W are Hilbert spaces of dimensions m and n , respectively, then $V \otimes W$ is a Hilbert space of dimension $m \times n$. The elements of the tensor product space are the linear combinations of the tensor products of the elements of the original vector spaces. In particular, if $|v_i\rangle$, $0 \leq i < n$ form an orthonormal basis of a vector space V and $|w_j\rangle$, $0 \leq j < m$ form an orthonormal basis of a vector space W , then $|v_i\rangle \otimes |w_j\rangle$, $0 \leq i < n$, $0 \leq j < m$ form an orthonormal basis of a vector space $V \otimes W$. If $|v\rangle, |v_1\rangle, |v_2\rangle \in V$ and $|w\rangle, |w_1\rangle, |w_2\rangle \in W$, and z is a scalar, then

1. $z \times (|v\rangle \otimes |w\rangle) = (z \times |v\rangle) \otimes |w\rangle = |v\rangle \otimes (z \times |w\rangle)$
2. $(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = (|v_1\rangle \otimes |w\rangle) + (|v_2\rangle \otimes |w\rangle)$
3. $|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = (|v\rangle \otimes |w_1\rangle) + (|v\rangle \otimes |w_2\rangle)$

If A is a linear operator defined on V , and B is a linear operator defined on W , then $A \otimes B$ is a linear operator defined on $V \otimes W$:

$$(A \otimes B)(|v\rangle \otimes |w\rangle) = (A|v\rangle) \otimes (B|w\rangle)$$

for $|v\rangle \in V$ and $|w\rangle \in W$. This definition extends naturally to all elements of $V \otimes W$:

$$(A \otimes B) \left(\sum_i k_i \times |v_i\rangle \otimes |w_i\rangle \right) = \sum_i k_i \times (A|v_i\rangle) \otimes (B|w_i\rangle)$$

The tensor product of operators is well-defined.

It may be more natural to think about tensor products in terms of matrices. Suppose A is an m by n matrix and B is a p by q matrix. The *Kronecker product*¹ of A and B , written $A \otimes B$, is the following $m \times p$ by $n \times q$ matrix:

$$A \otimes B = \begin{bmatrix} A_{0,0}B & A_{0,1}B & \dots & A_{0,n-1}B \\ A_{1,0}B & A_{1,1}B & \dots & A_{1,n-1}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{m,0}B & A_{m,1}B & \dots & A_{m-1,n-1}B \end{bmatrix}$$

The sub-matrices $A_{i,j}B$ are p by q matrices obtained by scalar multiplication of an entry $A_{i,j}$ of the matrix A and the entire matrix B .

Transpose, complex conjugation, and adjoint operations distribute over tensor product. The tensor product of two unitary operators is a unitary operator.

3.1.5 Computational basis

In quantum mechanics, the vector spaces of interest are the Hilbert spaces of dimension 2^n for some $n \in \mathbb{N}$. A convenient orthonormal basis is what is called a *computational basis*, in which we label 2^n basis vectors using binary strings of length n as follows:

$$\begin{aligned} \underbrace{|00\dots 00\rangle}_n &\text{ corresponds to } \underbrace{(1, 0, 0, \dots, 0, 0, 0)^T}_{2^n} \\ |00\dots 01\rangle &\text{ corresponds to } (0, 1, 0, \dots, 0, 0, 0)^T \\ |00\dots 10\rangle &\text{ corresponds to } (0, 0, 1, \dots, 0, 0, 0)^T \\ &\vdots \\ |11\dots 10\rangle &\text{ corresponds to } (0, 0, 0, \dots, 0, 1, 0)^T \\ |11\dots 11\rangle &\text{ corresponds to } (0, 0, 0, \dots, 0, 0, 1)^T \end{aligned}$$

¹The term *Kronecker product* is most often used in the context of matrices, while the term *tensor product* is used in the context of operators. Essentially, the two are equivalent.

It is important not to confuse $|0\rangle$, the zero vector in Dirac notation, with the vector $(0, 0)^T$. From now on, whenever we refer to a zero vector, we will mean the zero vector in Dirac notation. The tensor product $|i\rangle \otimes |j\rangle$ can be written simply as $|ij\rangle$. An arbitrary vector in \mathcal{H} can be written as a weighted sum of the computational basis vectors.

3.2 Postulates of quantum mechanics

In this section we introduce the basic concepts of quantum mechanics, as they pertain to the quantum systems that we will consider for quantum computation. The discussion of the underlying physical processes, spin- $\frac{1}{2}$ -particles, etc. is not of our interest. We are concerned with the model for quantum computation only.

Postulate 1 (state space): Associated to any isolated physical system is a Hilbert space, known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

The simplest quantum system is a quantum bit, or a *qubit*, which has a two-dimensional state space. Since $|0\rangle$ and $|1\rangle$ form an orthonormal basis for a two-dimensional Hilbert space, the state of a qubit is described by a vector

$$\alpha \times |0\rangle + \beta \times |1\rangle$$

where α and β are complex numbers. The condition that the state is a unit vector means that $|\alpha|^2 + |\beta|^2 = 1$. The coefficients α and β are called the *amplitudes* of $|0\rangle$ and $|1\rangle$, respectively. If both α and β are non-zero, then the state is in a *superposition* of the basis states $|0\rangle$ and $|1\rangle$.

If $|\psi\rangle$ and $|\phi\rangle$ are two states of an n -qubit system, such that $|\phi\rangle = e^{i\theta} \times |\psi\rangle$ for some θ , then $|\psi\rangle$ and $|\phi\rangle$ are *indistinguishable*. Very often in literature we see $|\psi\rangle = e^{i\theta} \times |\psi\rangle$, which is an unfortunate use of equality, since in general the vectors $|\psi\rangle$ and $e^{i\theta} \times |\psi\rangle$ are different.

Postulate 2 (evolution): The evolution of a closed quantum system is described by a *unitary transformation*.

That is, if the state of the system at time t_1 is described by a state vector $|\psi_1\rangle$, and the state of the same system at time t_2 is described by a state vector $|\psi_2\rangle$, then the states are related by a unitary operator U , which depends only on the times t_1 and t_2 : $|\psi_2\rangle = U|\psi_1\rangle$. Note that since unitary transformations preserve the norm, the resulting vector represents a valid system state. Another important remark is that unitarity of evolution operators implies *reversibility*, since every unitary operator has a unique inverse, which is unitary.

Postulate 3 (measurement): Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*, which act on the state space of the system being measured. The index m refers to the possible measurement outcomes. If the state of the system immediately prior to the measurement is described by a vector $|\psi\rangle$, then the probability of obtaining result m is $\langle\psi|M_m^\dagger M_m|\psi\rangle$, in which case the state of the system immediately after the measurement is described by the vector $\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$. The measurement operators satisfy the *completeness equation* $\sum m \cdot M_m^\dagger M_m = I$.

An important special class of measurements is known as *projective measurements*, which are equivalent to general measurements provided that we also have the ability to perform unitary transformations.

A projective measurement is described by an *observable* M , which is a Hermitian operator on the state space of the system being measured. This observable has a spectral decomposition $M = \sum m \cdot \lambda_m \times P_m$, where P_m is the projector onto the eigenspace of M with eigenvalue λ_m , which corresponds to the outcome of the measurement. The probability of measuring m is $\langle\psi|P_m|\psi\rangle$, in which case immediately after the measurement the system is found in the state $\frac{P_m|\psi\rangle}{\sqrt{\langle\psi|P_m|\psi\rangle}}$.

Given an orthonormal basis $|v_m\rangle$, $0 \leq m < 2^n$, measurement with respect to this basis is the corresponding projective measurement given by the observable $M = \sum m \cdot \lambda_m \times P_m$, where the projectors are $P_m = |v_m\rangle\langle v_m|$.

Measurement with respect to the computational basis is the simplest and the most commonly used class of measurements. In terms of the basis $|m\rangle$, $0 \leq m < 2^n$, the projectors are $P_m = |m\rangle\langle m|$ and $\langle\psi|P_m|\psi\rangle = |\psi_m|^2$. The state of the system immediately after measuring m is $|m\rangle$.

In the case of a single qubit, for example, measurement of the state $\alpha \times |0\rangle + \beta \times |1\rangle$ results in the outcome 0 with probability $|\alpha|^2$ and outcome 1 with probability $|\beta|^2$. The state of the system immediately after the measurement is $|0\rangle$ or $|1\rangle$, respectively.

Suppose the result of the measurement is ignored and we continue the computation. In this case the system is said to be in a *mixed state*. A mixed state is not the actual physical state of the system. Rather it describes our knowledge of the state the system is in. The conventional notation for mixed states is inconsistent. In the above example, the mixed state is expressed by the following equation:

$$|\psi\rangle = |\alpha|^2 \times \{|0\rangle\} + |\beta|^2 \times \{|1\rangle\}$$

The equation is meant to say that $|\psi\rangle$ is $|0\rangle$ with probability $|\alpha|^2$ and it is $|1\rangle$ with probability $|\beta|^2$. The brackets $\{\}$ cease to denote a set and are meant to denote the meta-state. The use of equality is unfortunate: $|\alpha|^2 \times \{|0\rangle\} + |\beta|^2 \times \{|1\rangle\}$ is not a valid quantum state. Furthermore, an application of operation U to the mixed state results in another mixed state — $U(|\alpha|^2 \times \{|0\rangle\} + |\beta|^2 \times \{|1\rangle\}) = |\alpha|^2 \times \{U|0\rangle\} + |\beta|^2 \times \{U|1\rangle\}$, although $|\alpha|^2 \times \{|0\rangle\} + |\beta|^2 \times \{|1\rangle\}$ is not in the domain of U . We will resolve this inconsistency in later chapters.

There are two important principles of quantum computation pertaining to measurements that are worth emphasising. We do not prove these in this work. The detailed proofs can be found in [12].

Principle of deferred measurement: Measurements can always be moved from an intermediate stage of quantum computation to the end of the computation. If any stage of the computation uses results of the measurement, then the classically controlled operations can always be replaced by conditional quantum operators.

Principle of implicit measurement: Without loss of generality, any qubits that are not measured at the end of the computation may be assumed to be measured.

Postulate 4 (composite systems): The state space of a composite physical system is the tensor product of the state spaces of the component systems. If we have systems numbered 0 up to and excluding n , and each system i , $0 \leq i < n$, is prepared in the state $|\psi_i\rangle$, then the joint state of the composite system is $|\psi_0\rangle \otimes |\psi_1\rangle \otimes \dots \otimes |\psi_{n-1}\rangle$.

It is important to understand that, while we can always describe a composite system given descriptions of the component systems, the reverse is not true. Indeed, given a state vector that describes a composite system, it may not be possible to factor it to obtain the state vectors of the component systems. A well-known example is the state $|\psi\rangle = |00\rangle/\sqrt{2} + |11\rangle/\sqrt{2}$. The reader may verify that there are no single qubit states $|\psi_1\rangle$ and $|\psi_2\rangle$, such that $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$. The state of the system that cannot be written as a product of states of its component systems is called an *entangled* state.

In a composite n -qubit system, to apply one-qubit operation U_i to qubit i , $0 \leq i < n$, we apply the operation $U_0 \otimes U_1 \otimes \dots \otimes U_{n-1}$ to the entire system. For example, applying A to the first and B to the second qubits of the two-qubit system in state $|\psi_0\rangle \otimes |\psi_1\rangle$ results in state $(A \otimes B)(|\psi_0\rangle \otimes |\psi_1\rangle) = (A|\psi_0\rangle) \otimes (B|\psi_1\rangle)$. Consequently, we can apply a one-qubit operation U to a particular qubit and leave the rest of the qubits unchanged. To do so, we apply U to the target qubit and the one-qubit identity operation I to every other qubit in the system. For example, if we apply U to the second qubit of the system in a

state $|\psi_0\rangle \otimes |\psi_1\rangle$, we get the state $(I \otimes U)(|\psi_0\rangle \otimes |\psi_1\rangle) = (I|\psi_0\rangle) \otimes (U|\psi_1\rangle) = |\psi_0\rangle \otimes (U|\psi_1\rangle)$. To apply U to every qubit in the n -qubit system, we apply $U^{\otimes n}$ (U tensored with itself n times) to the system. In a two-qubit system, $U^{\otimes 2}(|\psi_0\rangle \otimes |\psi_1\rangle) = (U \otimes U)(|\psi_0\rangle \otimes |\psi_1\rangle) = (U|\psi_0\rangle) \otimes (U|\psi_1\rangle)$.

Just as it may not be possible to represent the state of a multi-qubit system as tensor product of its component systems, it may not be possible to represent an operation on a composite system as a tensor product of single-qubit operations on the component systems. Consider, for example, “controlled-NOT” (CNOT) operation on two qubits defined by

$$CNOT(|0\rangle \otimes |x\rangle) = |0\rangle \otimes |x\rangle$$

$$CNOT(|1\rangle \otimes |x\rangle) = |1\rangle \otimes |1-x\rangle$$

where $x \in 0, 1$. It can be shown that there are no two single-qubit operations U_0 and U_1 , such that $CNOT = U_0 \otimes U_1$.

3.3 A quantum algorithm

In this section we look at one of the most famous quantum algorithms, Deutsch’s algorithm [4]. While Deutsch’s algorithm is simple and easy to understand (compared to most of the quantum algorithms), it illustrates two important aspects of quantum computing: *quantum parallelism* and *quantum superposition*. Moreover, it solves a problem that a classical computer cannot solve even in principle.

The problem is as follows. Given a function $f : 0, 1 \rightarrow 0, 1$, usually referred to as a (classical) *oracle*, compute $f(0) \oplus f(1)$, where \oplus denotes the “exclusive or” operator, provided that we are only allowed to query the oracle once. It is easy to convince oneself (by listing all possibilities) that there is no classical solution to this problem.

Before we present the quantum solution, we need to introduce two important op-

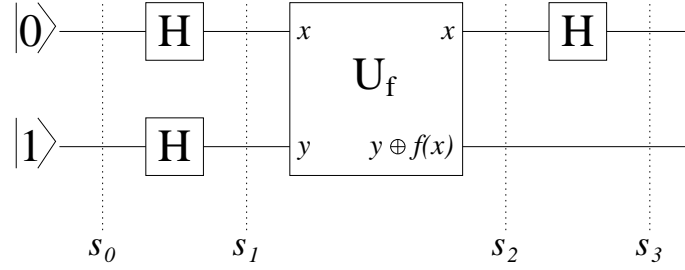


Figure 3.1: Deutsch's algorithm

erators. The first one is the *Hadamard transform*, defined by

$$H|0\rangle = \frac{1}{\sqrt{2}} \times (|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \times (|0\rangle - |1\rangle)$$

Hadamard is unitary and self-inverse. The second one is a quantum version of the oracle f . A standard way of defining a reversible version of f is to introduce an auxiliary bit as follows:

$$f_{rev}(x, y) = (x, y \oplus f(x))$$

The quantum oracle is a quantum version of f_{rev} :

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle$$

An important property of the oracle is that its application to state $|x\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}$, $x \in \{0, 1\}$, results in state $(-1)^{f(x)} \times |x\rangle \otimes (|0\rangle - |1\rangle)/\sqrt{2}$. The value of $f(x)$ gets pushed into the amplitude of $|x\rangle$.

Traditionally we use *quantum circuits* to describe and analyse quantum algorithms. Figure 3.1 shows a quantum circuit that implements Deutsch's algorithm.

Initially the system is in state $|0\rangle \otimes |1\rangle$. The first step of the algorithm is two applications of the Hadamard transform. After this step the system is in state

$$(H|0\rangle) \otimes (H|1\rangle) = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

The second step, the application of the oracle, results in state

$$U_f \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \frac{(-1)^{f(0)} \times |0\rangle + (-1)^{f(1)} \times |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Finally, applying Hadamard to the first qubit yields the state

$$\begin{aligned} & H \left(\frac{(-1)^{f(0)} \times |0\rangle + (-1)^{f(1)} \times |1\rangle}{\sqrt{2}} \right) \\ &= \frac{(-1)^{f(0)} \times (|0\rangle + |1\rangle) + (-1)^{f(1)} \times (|0\rangle - |1\rangle)}{2} \\ &= \pm |f(0) \oplus f(1)\rangle \end{aligned}$$

Measuring the first qubit in the computational basis gives $f(0) \oplus f(1)$ with probability 1.

It is convenient to think of the action of the quantum oracle simply as mapping $|x\rangle$ to $(-1)^{f(x)} \times |x\rangle$, for $x \in 0, 1$. Then use of an auxiliary qubit, prepared in state $(|0\rangle - |1\rangle)/\sqrt{2}$ and unchanged by the application of the oracle, is implicit.

Chapter 4

Introduction to probabilistic predicative programming

This chapter introduces the programming theory of our choice, on which our work on quantum programming is based — probabilistic predicative programming. We briefly introduce parts of the theory necessary for understanding chapters 5 and 6 of this work. For a course in predicative programming the reader is referred to [7]. Introduction to probabilistic predicative programming can be found in [8].

4.1 Predicative programming

In the theory of predicative programming specification is a boolean expression and refinement is logical implication. The theory applies to sequential and parallel, stand-alone and interactive, terminating and non-terminating, deterministic, non-deterministic, and probabilistic computation. It supports reasoning about computation time and space without introducing new concepts. While being remarkably general, it is also remarkably simple.

4.1.1 Specifications

A specification is a boolean expression. The variables in a specification represent the quantities of interest, such as prestate (inputs), poststate (outputs), and computation time and space. We use primed variables to describe outputs and unprimed variables to describe inputs. For example, specification $x' = x + 1$ in one integer variable x states that the final value of x is its initial value plus 1. A computation *satisfies* a specification if, given a prestate, it produces a poststate, such that the pair makes the specification true. A specification is *implementable* if for each input state there is at least one output state that satisfies the specification.

4.1.2 Specification notation

We use standard logical notation for writing specifications: \wedge (conjunction), \vee (disjunction), \Rightarrow (logical implication), $=$ (equality, boolean equivalence), \neq (non-equality, non-equivalence), and **if then else**. \equiv and \implies are the same as $=$ and \Rightarrow , but with lower precedence. We use standard mathematical notation, such as $+ - * / mod$. We use lowercase letters for variables of interest and uppercase letters for specifications.

In addition to the above, we use the following notations:

$$\begin{aligned} \sigma & \quad \text{prestate} \\ \sigma' & \quad \text{poststate} \\ ok & \quad \equiv \sigma' = \sigma \\ & \quad \equiv x' = x \wedge y' = y \wedge \dots \\ x := e & \quad \equiv x' = e \wedge y' = y \wedge \dots \end{aligned}$$

ok specifies that the values of all variables are unchanged. In the assignment $x := e$, x is a state variable (unprimed) and e is an expression (in unprimed variables) in the domain of x .

If R and S are specifications in variables x, y, \dots , R'' is obtained from R by substituting all occurrences of primed variables x', y', \dots with double-primed variables

x'', y'', \dots , and S'' is obtained from S by substituting all occurrences of unprimed variables x, y, \dots with double-primed variables x'', y'', \dots , then the *sequential composition* of R and S is defined by

$$R; S \equiv \exists x'', y'', \dots \cdot R'' \wedge S''$$

Here is the precedence order:

numbers, names, bracketed expressions
 function application (juxtaposition)
 superscript, subscript, exponentiation
 $\times /$
 \otimes
 $+ - \oplus$
 $, \dots$
 $= \neq < > \leq \geq :$
 \wedge
 \vee
 $\Leftarrow \Rightarrow$
if then else := measure
 $\lambda \cdot \forall \cdot \exists \cdot \Sigma \cdot ;$
 $\equiv \Leftarrow \Rightarrow$

Juxtaposition and the infix operator $-$ and $/$ associate from left to right. The infix operators $+ \times \wedge \vee ;$ associate in both directions. The operators $= < > \leq \geq$ are continuing. For example, $x = y = z$ means $x = y \wedge y = z$. The operators $\equiv \Leftarrow \Rightarrow$ are the same as $= \Leftarrow \Rightarrow$, but with lower precedence.

Various laws can be proved about sequential composition. One of the most important ones is the substitution law, which states that for any expression e of the prestate, state variable x , and specification P

$$x := e; P \equiv (\text{for } x \text{ substitute } e \text{ in } P)$$

According to the precedence rules listed above, this expression should be read as

follows:

$$((x := e); P) = (\text{for } x \text{ substitute } e \text{ in } P)$$

4.1.3 Refinement

Specification S is refined by specification P if and only if S is satisfied whenever P is satisfied:

$$\forall \sigma, \sigma' \cdot S \Leftarrow P$$

Specifications S and P are equal if and only if they are satisfied simultaneously:

$$\forall \sigma, \sigma' \cdot S = P$$

Given a specification, we are allowed to implement an equivalent specification or a stronger one.

4.1.4 Bunches

A *bunch* is a collection of objects. It is different from a set, which is a collection of objects in a package. Bunches are simpler than sets; they don't have a nesting structure. A bunch of one element is the element itself. We use upper-case to denote arbitrary bunches and lower-case to denote elements (an element is the same as a bunch of one element). A, B denotes the union of bunches A and B . $A : B$ denotes bunch inclusion — bunch A is included in bunch B . We use notation $x, ..y$ to mean from (including) x to (excluding) y .

4.1.5 Functions

If x is a fresh (previously unused) name, D is a bunch, and b is an arbitrary expression, then $\lambda x : D \cdot b$ is a *function* of a variable (parameter) x with domain D and body b . If

f is a function, then Δf denotes the domain of f . If $x : \Delta f$, then fx (f applied to x) is the corresponding element in the range. A function of n variables is a function of 1 variable, whose body is a function of $n - 1$ variables, for $n > 0$. A predicate is function whose body is a boolean expression. A relation is a function whose body is a predicate. A higher-order function is a function whose parameter is a function.

4.1.6 Quantifiers

We are now ready to explain what our quantifier notation really means. A *quantifier* is a unary prefix operator that applies to functions. If p is a predicate, then $\forall p$ is the boolean result, obtained by first applying p to all the elements in its domain and then taking the conjunction of those results. Taking the disjunction of the results produces $\exists p$. Similarly, if f is a numeric function, then $\sum f$ is the numeric result, obtained by first applying f to all the elements in its domain and then taking the sum of those results.

For example, applying the quantifier \sum to the function $\lambda i : 0, ..2^n \cdot |\psi_i|^2$, for some quantum state ψ , yields: $\sum \lambda i : 0, ..2^n \cdot |\psi_i|^2$, which for the sake of simplicity we abbreviate to $\sum i : 0, ..2^n \cdot |\psi_i|^2$. In addition, we allow a few other simplifications. For example, we can omit the domain of the free variable if it is clear from the context. We can also group variables from several quantifications. For example, $\sum i : 0, ..2^n \cdot \sum j : 0, ..2^n \cdot 2^{-m-n}$ can be abbreviated to $\sum i, j : 0, ..2^n \cdot 2^{-m-n}$.

4.1.7 Program development

A *program* is an implemented specification. For simplicity we only take the following to be implemented: *ok*, assignment, **if then else**, sequential composition, booleans, numbers, bunches, and functions.

Given a specification S , we proceed as follows. If S is a program, there is no work

to be done. If it is not, we build a program P , such that P refines S , i.e. $S \Leftarrow P$. The refinement can proceed in steps: $S \Leftarrow \dots \Leftarrow R \Leftarrow Q \Leftarrow P$.

One of the best features of Hehner's theory, is its simple treatment of recursion. In $S \Leftarrow P$ it is possible for S to appear in P . No additional rules are required to prove the refinement. For example,

$$x \geq 0 \Rightarrow x' = 0 \Leftarrow \mathbf{if } x = 0 \mathbf{ then } ok \mathbf{ else } (x := x - 1; x \geq 0 \Rightarrow x' = 0)$$

The specification says that if the initial value of x is non-negative, its final value must be 0. The solution is: if the value of x is zero, do nothing, otherwise decrement x and repeat. The refinement is proved as follows:

$$\begin{aligned} & \mathbf{if } x = 0 \mathbf{ then } ok \mathbf{ else } (x := x - 1; x \geq 0 \Rightarrow x' = 0) && \text{expand } ok \\ \equiv & \mathbf{if } x = 0 \mathbf{ then } x' = x \mathbf{ else } (x := x - 1; x \geq 0 \Rightarrow x' = 0) && \text{substitution} \\ \equiv & \mathbf{if } x = 0 \mathbf{ then } x' = x \mathbf{ else } (x - 1 \geq 0 \Rightarrow x' = 0) && \text{context} \\ \equiv & \mathbf{if } x = 0 \mathbf{ then } x' = 0 \mathbf{ else } (x - 1 \geq 0 \Rightarrow x' = 0) && \text{weaken} \\ \implies & \mathbf{if } x = 0 \mathbf{ then } (x \geq 0 \Rightarrow x' = 0) \mathbf{ else } (x \geq 0 \Rightarrow x' = 0) && \text{case idempotence} \\ \equiv & x \geq 0 \Rightarrow x' = 0 \end{aligned}$$

4.1.8 Time and space

To talk about computation time and space we don't need to expand the theory. We add special variables — time variable t and space variable s — and we treat them the same way that we treat any other variable. For example, consider the program in section 4.1.7.

$$P \Leftarrow \mathbf{if } x = 0 \mathbf{ then } ok \mathbf{ else } (x := x - 1; P)$$

$$P \equiv x \geq 0 \Rightarrow x' = 0$$

How long does the computation take? To account for time we add a time variable t . We use t to denote the time, at which the computation starts, and t' to denote the time,

at which the computation ends¹. We choose to use a *recursive time* measure, in which we charge 1 time unit for each time P is called. We replace each call to P to include the time increment as follows:

$$P \longleftarrow \mathbf{if } x = 0 \mathbf{ then } ok \mathbf{ else } (x := x - 1; t := t + 1; P)$$

It is easy to see that t is incremented the same number of times that x is decremented, i.e. $t' = t + x$, if $x \geq 0$, and $t' = \infty$, otherwise. We prove this as follows:

$$\begin{aligned} & \mathbf{if } x = 0 \mathbf{ then } ok \\ & \mathbf{else } (x := x - 1; t := t + 1; x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty) \quad \text{expand } ok \\ \equiv & \mathbf{if } x = 0 \mathbf{ then } x' = x \wedge t' = t \\ & \mathbf{else } (x := x - 1; t := t + 1; x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty) \quad \text{substitute} \\ \equiv & \mathbf{if } x = 0 \mathbf{ then } x' = x \wedge t' = t \\ & \mathbf{else } x - 1 \geq 0 \wedge t' = (t + 1) + (x - 1) \vee x - 1 < 0 \wedge t' = \infty \quad \text{context, math} \\ \equiv & \mathbf{if } x = 0 \mathbf{ then } x = 0 \wedge x' = 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty \\ & \mathbf{else } x \neq 0 \wedge x - 1 \geq 0 \wedge t' = t + x \vee x \neq 0 \wedge x - 1 < 0 \wedge t' = \infty \quad \text{weaken} \\ \implies & \mathbf{if } x = 0 \mathbf{ then } x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty \quad \text{case} \\ & \mathbf{else } x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty \quad \text{idempotence} \\ \equiv & x \geq 0 \wedge t' = t + x \vee x < 0 \wedge t' = \infty \end{aligned}$$

4.2 Probabilistic predicative programming

Probabilistic predicative programming was introduced in [7] and was further developed in [8]. It is a generalisation of predicative programming that allows reasoning about probability distributions of values of variables of interest. Although in this work

¹In case of non-termination, $t' = \infty$

we apply this reasoning to boolean and integer variables only, the theory does not change if we want to work with real numbers: we replace summations with integrals.

4.2.1 Probabilistic specifications

A *probability* is a real number between 0 and 1, inclusive. A *distribution* is an expression whose value is a probability and whose sum over all values of variables is 1. For example, if n is a positive natural variable, then 2^{-n} is a distribution, since for any n , 2^{-n} is a probability, and $\sum n \cdot 2^{-n} = 1$. In two positive natural variables m and n , 2^{-n-m} is also a distribution. If a distribution of several variables can be written as a product of distributions of the individual variables, then the variables are *independent*. For example, m and n in the previous example are independent. Given a distribution of several variables, we can sum out some of the variables to obtain a distribution of the rest of the variables. In our example, $\sum n \cdot 2^{-n-m} = 2^{-m}$, which is a distribution of m .

To generalise boolean specifications to probabilistic specifications, we use 1 and 0 for boolean *true* and *false*, respectively.² If S is an implementable deterministic specification and p is a distribution of the initial state x, y, \dots , then the distribution of the final state is

$$\sum_{x, y, \dots} S \times p$$

Consider the following example. Let x and y be two integer variables, and X and Y be distributions of x and y , respectively:

$$X = (x = 0)/2 + (x = 1)/2$$

$$Y = (y = 0)/3 + (y = 1) \times 2/3$$

²Readers familiar with \top and \perp notation can notice that we take the liberty to equate $\top = 1$ and $\perp = 0$.

These distributions say that initially x is 0 one half of the time and 1 one half of the time and initially y is 0 one third of the time and 1 two thirds of the time. The joint distribution is then

$$\begin{aligned} X \times Y &= (x = y = 0)/6 + (x = 0 \wedge y = 1)/3 \\ &+ (x = 1 \wedge y = 0)/6 + (x = y = 1)/3 \end{aligned}$$

Let the specification S be

$$y := y + x$$

Then the distribution of the final state is:

$$\begin{aligned} \sum x, y \cdot S \times X \times Y &= \sum x, y \cdot (y' = y + x \wedge x' = x) \times \\ &((x = y = 0)/6 + (x = 0 \wedge y = 1)/3 + \\ &(x = 1 \wedge y = 0)/6 + (x = y = 1)/3) \\ &= (x' = y' = 0)/6 + (x' = 0 \wedge y' = 1)/3 + \\ &(x' = y' = 1)/6 + (x' = 1 \wedge y' = 2)/3 \end{aligned}$$

The distribution says that after the computation the values of x and y , respectively, are both 0 one sixth of the time, 0 and 1 one third of the time, both 1 one sixth of the time, and 1 and 2 one third of the time. Note that we cannot express this distribution as a product of distributions of the final value of x and the final value of y . They are no longer independent.

A probability distribution of initial and final values of variables is a probabilistic specification. Sequential composition and **if then else** are generalised to apply to probabilistic specifications as follows.

If R and S are specifications in variables x, y, \dots , R' is obtained from R by substituting all occurrences of primed variables x', y', \dots with double-primed variables x'', y'', \dots , and S'' is obtained from S by substituting all occurrences of unprimed variables x, y, \dots with double-primed variables x'', y'', \dots , then the *sequential composition*

of R and S is defined by

$$R; S = \sum x'', y'', \dots \cdot R'' \times S''$$

If p is a probability and R and S are distributions, then

$$\mathbf{if } p \mathbf{ then } R \mathbf{ else } S = p \times R + (1 - p) \times S$$

Various laws can be proven about sequential composition. One of the most important ones, the substitution law, introduced in section 4.1, applies to probabilistic specifications as well.

4.2.2 Developing probabilistic programs

To implement a probabilistic specification we use a pseudo-random number generator. Since we cannot, even in theory, produce a real random number generator by means of traditional computing, we assume that a pseudo-random number generator generates truly random numbers and we simply refer to it as random number generator. For a positive natural variable n , we say that *rand* n produces a random natural number uniformly distributed in $0, ..n$. To reason about the values supplied by the random number generator consistently, we replace every occurrence of *rand* n with a fresh variable r whose value has probability $(r : 0, ..n)/n$. If *rand* occurs in a context such as $r = \text{rand } n$, we replace the equation by $r : (0, ..n)/n$. If *rand* occurs in the context of a loop, we parametrise the introduced variables by the execution time.

Recall the example from sections 4.1.7 and 4.1.8.

$$P \Leftarrow \mathbf{if } x = 0 \mathbf{ then } ok \mathbf{ else } (x := x - 1; t := t + 1; P)$$

$$P = (x \geq 0 \Rightarrow x' = 0) \wedge (t' = t + x)$$

Let us change the program slightly by introducing some probabilities:

$$P \Leftarrow \mathbf{if } x = 0 \mathbf{ then } ok \mathbf{ else } (x := x - \text{rand } 2; t := t + 1; P)$$

In the new program at each iteration x is either decremented by 1 or it is unchanged, with equal probability. Our intuition tells us that the revised program should still work, except it should take longer. Let us prove it. We replace $rand$ with $r : time \rightarrow (0, 1)$ with rt having probability $1/2$. Ignoring time:

$$\begin{aligned}
& \mathbf{if } x = 0 \mathbf{ then } ok \\
& \mathbf{else } (x := x - rand\ 2; x \geq 0 \Rightarrow x' = 0) \quad \text{replace } ok \text{ and } rand \\
= & \mathbf{if } x = 0 \mathbf{ then } x' = x \\
& \mathbf{else } (x := x - r\ t; x \geq 0 \Rightarrow x' = 0) \quad \text{substitute} \\
= & \mathbf{if } x = 0 \mathbf{ then } x' = x \mathbf{ else } (x - r\ t \geq 0 \Rightarrow x' = 0) \quad \mathbf{if\ then\ else} \\
= & x = 0 \wedge x' = x \vee x \neq 0 \wedge (x - r\ t \geq 0 \Rightarrow x' = 0) \quad \text{context, material implication} \\
= & x = 0 \wedge x' = 0 \vee x \neq 0 \wedge x < r\ t \vee x \neq 0 \wedge x' = 0 \quad \text{boolean laws, } r\ t : 0, 1 \\
= & x \geq 0 \Rightarrow x' = 0
\end{aligned}$$

As for the execution time, we can prove that it takes at least x time units to complete. That is, if we are lucky every time, we make progress toward $x' = 0$ at each iteration. On the other hand, if we are extremely unlucky, it is possible that $t' = \infty$.

$$\begin{aligned}
& \mathbf{if } x = 0 \mathbf{ then } ok \\
& \mathbf{else } (x := x - rand\ 2; t := t + 1; t' \geq t + x) \quad \text{replace } ok \text{ and } rand \\
= & \mathbf{if } x = 0 \mathbf{ then } x' = x \wedge t' = t \\
& \mathbf{else } (x := x - r\ t; t := t + 1; t' \geq t + x) \quad \text{substitute} \\
= & \mathbf{if } x = 0 \mathbf{ then } x' = x \wedge t' = t \\
& \mathbf{else } t' \geq t + 1 + x - r\ t \quad \text{context} \\
= & \mathbf{if } x = 0 \mathbf{ then } x' = x \wedge t' = t + x \\
& \mathbf{else } t' \geq t + x + (1 - r\ t) \quad \text{weaken} \\
\Rightarrow & \mathbf{if } x = 0 \mathbf{ then } t' \geq t + x \mathbf{ else } t' \geq t + x \quad \text{case idempotence}
\end{aligned}$$

$$= t' \geq t + x$$

How long should we expect to wait for the execution to complete? In other words, what is the distribution of t' ? We can either give a little thought to the problem or recall what we've learned about Negative Binomial Distribution in a statistics class some years ago, to arrive at the following distribution of the final states:

$$(0 = x' = x = t' - t) + (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t' - t}} \quad \text{where}$$

$$\binom{n}{m} = \frac{n!}{m! \times (n - m)!}$$

The proof is as follows:

$$\begin{aligned} & \sum rt \cdot \frac{1}{2} \times \left(\begin{array}{l} \mathbf{if} \ x = 0 \ \mathbf{then} \ ok \\ \mathbf{else} \ \left(\begin{array}{l} x := x - rt; \ t := t + 1; \\ (0 = x' = x = t' - t) + \\ (0 = x' < x \leq t' - t) \times \\ \left(\binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t' - t}} \right) \end{array} \right) \end{array} \right) \end{array} \quad \begin{array}{l} \text{replace } ok, \\ \text{substitute} \end{array} \\ = & \sum rt \cdot \frac{1}{2} \times \left(\begin{array}{l} \mathbf{if} \ x = 0 \ \mathbf{then} \ x' = x \wedge t' = t \\ \mathbf{else} \ \left(\begin{array}{l} (0 = x' = x - rt = t' - t - 1) + \\ (0 = x' < x - rt \leq t' - t - 1) \times \\ \left(\binom{t' - t - 2}{x - rt - 1} \times \frac{1}{2^{t' - t - 1}} \right) \end{array} \right) \end{array} \right) \quad \begin{array}{l} \text{expand,} \\ \text{distribute } \frac{1}{2} \end{array} \\ = & \left(\mathbf{if} \ x = 0 \ \mathbf{then} \ (x' = x \wedge t' = t) / 2 \right. \\ & \left. \mathbf{else} \ \left(\begin{array}{l} (0 = x' = x = t' - t - 1) / 2 + \\ (0 = x' < x \leq t' - t - 1) \times \left(\binom{t' - t - 2}{x - 1} \times \frac{1}{2^{t' - t}} \right) \end{array} \right) \right) + \end{array}$$

$$\begin{aligned}
& \left(\mathbf{if} \ x = 0 \ \mathbf{then} \ (x' = x \wedge t' = t)/2 \right. \\
& \quad \mathbf{else} \ \left((0 = x' = x - 1 = t' - t - 1)/2 + \right. \quad \text{collect} \\
& \quad \quad \left. (0 = x' < x - 1 \leq t' - t - 1) \times \binom{t' - t - 2}{x - 2} \times \frac{1}{2^{t' - t}} \right) \quad \text{terms} \\
= & \ \mathbf{if} \ x = 0 \ \mathbf{then} \ x' = x \wedge t' = t \\
& \quad \mathbf{else} \ \left((0 = x' = x = t' - t - 1) + \right. \\
& \quad \quad (0 = x' < x \leq t' - t - 1) \times \binom{t' - t - 2}{x - 1} + \\
& \quad \quad (0 = x' = x - 1 = t' - t - 1) + \quad \text{context,} \\
& \quad \quad \left. (0 = x' < x - 1 \leq t' - t - 1) \times \binom{t' - t - 2}{x - 2} \right) \times \frac{1}{2^{t' - t}} \quad \text{arithmetic} \\
= & \ \mathbf{if} \ x = 0 \ \mathbf{then} \ x' = x \wedge t' = t \\
& \quad \mathbf{else} \ \frac{1}{2^{t' - t}} \times (x' = 0) \times \\
& \quad \left((1 = x = t' - t) \times 1 + (1 = x < t' - t) \times 1 + \right. \\
& \quad \quad \left. (1 < x = t' - t) \times 1 + (1 < x < t' - t) \times \binom{t' - t - 1}{x - 1} \right) \\
= & \ \mathbf{if} \ x = 0 \ \mathbf{then} \ x' = x \wedge t' = t \\
& \quad \mathbf{else} \ \frac{1}{2^{t' - t}} \times (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \quad \text{expand if} \\
= & \ (0 = x' = x = t' = t) + \\
& \quad (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t' - t}}
\end{aligned}$$

Since for positive x , t' is distributed according to the negative binomial distribution with parameters x and $\frac{1}{2}$, its mean value is

$$\begin{aligned}
& \sum t' \cdot (t' - t) \times \left((0 = x = t' - t) + (0 < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t' - t}} \right) \\
= & \ 2 \times x + t
\end{aligned}$$

Therefore, we should expect to wait $2 \times x$ time units for the computation to complete.

4.2.3 Partial probabilistic specifications

Suppose we want the computation to produce $x' = 0$ with probability p and $x' \neq 0$ with probability $1 - p$. We do not care what the exact value of the non-zero x is, the specification is satisfied as long as x is not zero $1 - p$ of the time. The probabilistic specification is:

$$(x' = 0) \times p + (x' \neq 0) \times (1 - p)$$

and one way to implement it is:

$$\begin{aligned} & (x' = 0) \times p + (x' \neq 0) \times (1 - p) \\ & \geq (x' = 0) \times p + (x' = 1) \times (1 - p) \\ & = \mathbf{if } p \mathbf{ then } x := 0 \mathbf{ else } x := 1 \end{aligned}$$

Partial specifications often come up in quantum computation. The goal of a number of quantum algorithms is to successfully discriminate between their inputs with high probability. In such cases a (partial) probabilistic specification often looks like

$$(\text{function of the input}) = x' = 0$$

where x is the indicator variable. The implementation assigns values to x according to the distribution that refines the specification.

There is an intrinsic problem with formalising specifications as superdistributions — the convex closure problem, which we do not address in this work. See [8] for an example of the problem.

Chapter 5

The quantum system

This chapter introduces our formalisation of a quantum system. It defines pure and mixed quantum states, operations on a quantum state, composition of quantum systems, and a quantum measurement.

5.1 The quantum state

Let \mathbb{C} be the set of all complex numbers with the absolute value operator $|\cdot|$ and the complex conjugate operator $*$. Then a state of an n -qubit system is a function $\psi : 0, \dots, 2^n \rightarrow \mathbb{C}$, such that:

$$\sum_{x : 0, \dots, 2^n} |\psi x|^2 = 1$$

If ψ and ϕ are two states of an n -qubit system, then their *inner product*, $\langle \psi | \phi \rangle : \mathbb{C}$, is defined by:

$$\langle \psi | \phi \rangle = \sum_{x : 0, \dots, 2^n} (\psi x)^* \times (\phi x)$$

A *basis* of an n -qubit system is a collection of 2^n quantum states $b_{0, \dots, 2^n}$, such that:

$$\forall i, j : 0, \dots, 2^n \cdot \langle b_i | b_j \rangle = (i = j)$$

Most of the time we choose the computational basis, which is the simplest, to describe the quantum system. We adopt the following Dirac-like notation for the computational basis: if $x : 0, \dots, 2^n$, then \mathbf{x} denotes the corresponding n -bit binary encoding of x and $|\mathbf{x}\rangle : 0, \dots, 2^n \rightarrow \mathbb{C}$ is the following quantum state:

$$|\mathbf{x}\rangle = \lambda_i : 0, \dots, 2^n \cdot (i = x)$$

Given a not necessarily computational basis $b_{0, \dots, 2^n}$, an arbitrary state ψ of an n -qubit system can be written as:

$$\psi = \sum_{x : 0, \dots, 2^n} (\psi x) \times b_x$$

5.2 Composing quantum systems

If ψ is a state of an m -qubit system and ϕ is a state of an n -qubit system, then $\psi \otimes \phi$, the tensor product of ψ and ϕ , is the following state of a composite $m + n$ -qubit system:

$$\psi \otimes \phi = \lambda_i : 0, \dots, 2^{m+n} \cdot \psi(i \operatorname{div} 2^n) \times \phi(i \operatorname{mod} 2^n)$$

For example, a 3-qubit system composed from a 1-qubit system in state $|1\rangle$ and a 2-qubit system in state $|00\rangle$ is described by the following state:

$$\begin{aligned} |1\rangle \otimes |00\rangle &= \lambda_i : 0, \dots, 2^{1+2} \cdot |1\rangle(i \operatorname{div} 2^2) \times |00\rangle(i \operatorname{mod} 2^2) \\ &= \lambda_i : 0, \dots, 8 \cdot ((i \operatorname{div} 4) = 1) \times ((i \operatorname{mod} 4) = 0) \\ &= \lambda_i : 0, \dots, 8 \cdot (i = 4) \\ &= |100\rangle \end{aligned}$$

Here is another example. The composition of two 1-qubit quantum systems, the first one in state $|1\rangle$ and the second one in a superposition state $|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}$, results in

a 2-qubit system in state:

$$\begin{aligned} |1\rangle \otimes (|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) &= (\lambda_i : 0, 1 \cdot i = 1) \otimes (\lambda_i : 0, 1 \cdot 1/\sqrt{2}) \\ &= \lambda_i : 0, \dots, 4 \cdot ((i \operatorname{div} 2) = 1)/\sqrt{2} \\ &= |10\rangle/\sqrt{2} + |11\rangle/\sqrt{2} \end{aligned}$$

which is the equally weighted superposition of states $|10\rangle$ and $|11\rangle$ in a 2-qubit system.

We write \otimes^n to mean *tensored with itself n times*. For example,

$$|0\rangle^{\otimes n} = \underbrace{|0\rangle \otimes \dots \otimes |0\rangle}_n = \underbrace{|0\dots 0\rangle}_n$$

5.3 Unitary transformations

An operation defined on a n -qubit quantum system is a higher-order function, whose domain and range are maps from $0, \dots, 2^n$ to the complex numbers. An *identity* operation on a state of an n -qubit system is defined by

$$I^n = \lambda\psi : 0, \dots, 2^n \rightarrow \mathbb{C} \cdot \psi$$

For a linear operation A , the *adjoint* of A , written A^\dagger , is the (unique) operation, such that for any two states ψ and ϕ ,

$$\langle \psi | A\phi \rangle = \langle A^\dagger \psi | \phi \rangle$$

The *unitary transformations* that describe the evolution of a n -qubit quantum system are operations U defined on the system, such that:

$$U^\dagger U = I^n$$

In this setting, the *tensor product* of operators is defined in the usual way. If ψ is a state of an m -qubit system, ϕ is a state of an n -qubit system, and U and V are operations

defined on m and n -qubit systems, respectively, then the tensor product of U and V is defined on an $m + n$ qubit system by:

$$(U \otimes V)(\psi \otimes \phi) = (U\psi) \otimes (V\phi)$$

which, for bases $b_{0,..,2^m}$ and $c_{0,..,2^n}$, extends linearly to:

$$\begin{aligned} & (U \otimes V) \left(\sum_{j: 0, .., 2^m} \sum_{i: 0, .., 2^n} k_{ij} \times (b_i \otimes c_j) \right) \\ &= \sum_{j: 0, .., 2^m} \sum_{i: 0, .., 2^n} k_{ij} \times (U b_i \otimes V c_j) \end{aligned}$$

Just as with tensor products of states, we write $U^{\otimes n}$ to mean *operation U tensored with itself n times*:

$$U^{\otimes n} = \underbrace{U \otimes \dots \otimes U}_n$$

To apply an operation U defined on a 1-qubit system to qubit i in a composite n -qubit system, we apply the operation U_i^n to the entire system, where U_i^n is defined by:

$$U_i^n = \underbrace{I \otimes \dots \otimes I}_i \otimes U \otimes \underbrace{I \otimes \dots \otimes I}_{n-i-1}$$

Consider the following example of a unitary transformation. The *Hadamard* transform defined on a 1-qubit system is a higher-order function from $0, 1 \rightarrow \mathbb{C}$ to $0, 1 \rightarrow \mathbb{C}$, defined by:

$$H = \lambda \psi : 0, 1 \rightarrow \mathbb{C} \cdot i : 0, 1 \cdot (\psi 0 + (-1)^i \times \psi 1) / \sqrt{2} \quad (5.1)$$

The operation $H^{\otimes n}$ on a n -qubit system applies H to every qubit of the system. Its action on a zero state of an n -qubit system is:

$$H^{\otimes n} |0\rangle^{\otimes n} = \sum_{x: 0, .., 2^n} |x\rangle / \sqrt{2^n} \quad (5.2)$$

On a general state $|x\rangle$, the action of $H^{\otimes n}$ is:

$$H^{\otimes n} |x\rangle = \sum_{y: 0, .., 2^n} (-1)^{x \cdot y} \times |y\rangle / \sqrt{2^n} \quad (5.3)$$

where $x \cdot y$ is the bitwise inner product of x and y modulo 2 (bitwise XOR).

5.4 Measurement

Suppose we have a system of n qubits in state ψ and we measure it. Suppose also that we have a variable r from the domain $0, \dots, 2^n$, which we use to record the result of the measurement, and variables x, y, \dots , which are not affected by the measurement. Then the measurement corresponds to a probabilistic specification that gives the probability distribution of ψ' and r' (these depend on ψ and on the type of measurement) and states that the variables x, y, \dots are unchanged.

For a general quantum measurement described by a collection $M = M_{0, \dots, 2^n}$ of measurement operators, which satisfy the completeness equation $\sum_m M_m^\dagger M_m = I$, the specification is **measure** $_M \psi r$, where

$$\mathbf{measure}_M \psi r = \langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle \times \left(\psi' = \frac{M_{r'} \psi}{\sqrt{\langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle}} \right) \times (x' = x) \times (y' = y) \dots$$

To obtain the distribution of, say, r' we sum out the rest of the variables as follows:

$$\begin{aligned} & \sum_{\psi', x', y', \dots} \langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle \times \left(\psi' = \frac{M_{r'} \psi}{\sqrt{\langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle}} \right) \times (x' = x) \times (y' = y) \dots \\ &= \langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle \end{aligned}$$

For the projective measurement defined by an observable $O = \sum_m \lambda_m \times P_m$, where P_m is the projector on the eigenspace of O with eigenvalue λ_m :

$$\mathbf{measure}_O \psi r = \langle \psi | P_{r'} \psi \rangle \times \left(\psi' = \frac{P_{r'} \psi}{\sqrt{\langle \psi | P_{r'} \psi \rangle}} \right) \times (x' = x) \times (y' = y) \dots$$

Given an arbitrary orthonormal basis $B = b_{0, \dots, 2^n}$, measurement of ψ in basis B is:

$$\mathbf{measure}_B \psi r = |\langle b_{r'} | \psi \rangle|^2 \times (\psi' = b_{r'}) \times (x' = x) \times (y' = y) \dots$$

Finally, the simplest and the most commonly used measurement in the computational basis is:

$$\mathbf{measure} \psi r = |\psi_{r'}|^2 \times (\psi' = |r'\rangle) \times (x' = x) \times (y' = y) \dots$$

In this case the distribution of r' is $|\psi_{r'}|^2$ and the distribution of the quantum state is:

$$\sum r' \cdot |\psi_{r'}|^2 \times (\psi' = |\mathbf{r}'\rangle)$$

which is precisely the mixed quantum state that results from the measurement. We have recovered consistency in the definitions of pure and mixed states by providing the tools to compute directly with distributions and, thus, removing the necessity to introduce an awkward notation $\{\}$ for the mixed state.

Chapter 6

Quantum programming

We are now ready to introduce our theory of quantum programming. This chapter begins with an introduction of the basic building blocks — the simplest programs that we assume to be implemented. This is a reasonable assumption, since we adopt Knill’s model of a quantum computer [10], and this model guarantees precisely the operations we assume to be implemented. Then this chapter presents several well-known quantum algorithms together with their classical versions and their comparative time complexity analyses; these are intended to demonstrate the generality and the simplicity of our framework. The chapter concludes with the discussion of quantum computation with mixed states, where we show restoring consistency in the definition of the mixed state and the application of operators to it.

6.1 A quantum program

In order to develop quantum programs we need to add to our list of implemented things from section 4.1.7. We add variables of type quantum state as in section 5.1 and we allow the following three kinds of operations on these variables. If ψ is a state of an n -qubit quantum system, r is a natural variable, and M is a collection of measurement

operators that satisfy the completeness equation, then:

1. $\psi := |0\rangle^{\otimes n}$ is a program
2. $\psi := Uq$, where U is a unitary transformation on an n -qubit system, is a program
3. **measure** _{M} ψr is a program

The special cases of measurements, described in section 5.4, are therefore also allowed: for an observable O and an orthonormal basis B , **measure** _{O} qr , **measure** _{B} qr , and **measure** qr are programs.

Now we are ready to develop quantum programs. In the rest of this chapter we will describe a few well-known problems, develop both classical and quantum solutions to these problems, and analyse their complexity.

6.2 Fair coin

We begin with the following simple problem. The task is to simulate a flip of a fair coin. Let us use a variable c from the domain $0, 1$ to represent the coin so that 0 and 1 represent *head* and *tail*, respectively. The probabilistic specification is then:

$$(c' = 0)/2 + (c' = 1)/2$$

The specification says that we want to obtain *head* with probability $1/2$ and *tail* with probability $1/2$.

First, let us consider a classical solution. Assume we have a (pseudo-) random number generator $rand$, as shown in section 4.2.2. Then

$$\begin{aligned} & (c' = 0)/2 + (c' = 1)/2 \\ &= (c' : 0, 1)/2 \\ &= c := rand \ 2 \end{aligned}$$

There is only one problem. We cannot, even in theory, create *rand* by means of classical computing.

On the other hand, with a quantum computer, in theory we can implement the following. Let ψ be a state on a 1-qubit system. Then

$$\begin{aligned}
 & (c' = 0)/2 + (c' = 1)/2 \\
 \equiv & \left((|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \mid c' \right)^2 && \text{definition of measure} \\
 \equiv & \mathbf{measure} \left((|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \mid c \right) && \text{Hadamard} \\
 \equiv & \mathbf{measure} (H|0\rangle) \mid c && \text{substitution} \\
 \equiv & \psi := |0\rangle; \psi := H\psi; \mathbf{measure} \psi \mid c
 \end{aligned}$$

6.3 Deutsch's problem

Recall the Deutsch's problem discussed in section 3.3. The task is: given an oracle function $f : 0, 1 \rightarrow 0, 1$, compute $f0 \oplus f1$. For now, we ignore the restriction on the number of queries to the oracle. With natural x , the specification is:

$$x' = f0 \oplus f1$$

Here is a simple classical solution. With natural y :

$$\begin{aligned}
 & x' = f0 \oplus f1 && \text{strengthen} \\
 \Leftarrow & x' = f0 \oplus f1 \wedge y' = f1 && \text{substitution} \\
 \equiv & x := f0; y := f1; x := x \oplus y
 \end{aligned}$$

The only reason we do not implement the specification by simply assigning $f0 \oplus f1$ to x is that we want to be able to add time increments before each statement that contains a call to f (see time analysis later in this section). It is therefore convenient to have one call to f per statement.

Recall that the definition of the quantum analog of the above classical oracle is:

$$U_f = \lambda\psi : 0, 1 \rightarrow \mathbb{C} \cdot x : 0, 1 \cdot (-1)^{f^x} \times \psi x$$

Let us develop a quantum solution. With a state ψ of a 1-qubit system:

$$\begin{aligned} x' &= f0 \oplus f1 && \text{arithmetic} \\ &= |(((-1)^{f^0}/2 + (-1)^{f^1}/2) \times |0\rangle + ((-1)^{f^0}/2 - (-1)^{f^1}/2) \times |1\rangle) x'|^2 && \text{measure} \\ &= \text{measure } (((-1)^{f^0}/2 + (-1)^{f^1}/2) \times |0\rangle + \\ &\quad ((-1)^{f^0}/2 - (-1)^{f^1}/2) \times |1\rangle) x && \text{arithmetic} \\ &= \text{measure } ((-1)^{f^0}/2 \times (|0\rangle + |1\rangle) + (-1)^{f^1}/2 \times (|0\rangle - |1\rangle)) x && \text{Hadamard} \\ &= \text{measure } ((-1)^{f^0}/\sqrt{2} \times (H|0\rangle) + (-1)^{f^1}/\sqrt{2} \times (H|1\rangle)) x && \text{linearity} \\ &= \text{measure } H((-1)^{f^0}/\sqrt{2} \times |0\rangle + (-1)^{f^1}/\sqrt{2} \times |1\rangle) x && \text{Oracle} \\ &= \text{measure } H(U_f|0\rangle/\sqrt{2} + U_f|1\rangle/\sqrt{2}) x && \text{linearity} \\ &= \text{measure } H(U_f(|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2})) x && \text{Hadamard} \\ &= \text{measure } H(U_f(H|0\rangle)) x && \text{substitutions} \\ &= \psi := |0\rangle; \psi := H\psi; \psi := U_f\psi; \psi := H\psi; \text{measure } \psi x \end{aligned}$$

So far we have two solutions — a simple classical one and a complicated quantum one. Let us add the restriction on the number of allowed calls to the oracle. We add a time variable t and decide to charge 1 unit of time for a call to the oracle, leaving all other operations free. The new specification is:

$$x' = f0 \oplus f1 \wedge t' = t + 1$$

The above quantum solution still works:

$$\begin{aligned} x' &= f0 \oplus f1 \wedge t' = t + 1 && \text{as before} \\ &= |(((-1)^{f^0}/2 + (-1)^{f^1}/2) \times |0\rangle + ((-1)^{f^0}/2 - (-1)^{f^1}/2) \times |1\rangle) x'|^2 \times \\ &\quad (t' = t + 1) && \text{as before} \end{aligned}$$

$$\begin{aligned}
&= |H(U_f(H|0\rangle)) x'|^2 \times (t' = t + 1) && \text{substitutions} \\
&= \psi := |0\rangle; \psi := H\psi; t := t + 1; \psi := U_f\psi; \psi := H\psi; \\
&\quad |\psi x'|^2 \times (t' = t) && \text{measure} \\
&= \psi := |0\rangle; \psi := H\psi; t := t + 1; \psi := U_f\psi; \psi := H\psi; \mathbf{measure} \psi x
\end{aligned}$$

Readers can convince themselves that the new specification is unimplementable classically. Our classical solution fails with the time-constraint:

$$\begin{aligned}
&t := t + 1; x = f0; t := t + 1; y := f1; x := x \oplus y && \text{substitution} \\
&= x' = f0 \oplus f1 \wedge y' = f1 \wedge t' = t + 2 && \text{weaken} \\
&\implies x' = f0 \oplus f1 \wedge t' = t + 2
\end{aligned}$$

Classically, at least two calls to the oracle functions are necessary to compute $f0 \oplus f1$.

6.4 Deutsch-Josza problem

Deutsch-Josza problem generalises Deutsch's problem to n qubits [5]. It is an example of the broad class of quantum algorithms that are based on quantum Fourier transform [9]. The task is: given a function $f : 0, \dots, 2^n \rightarrow 0, 1$, such that f is either constant or balanced, determine which case it is. Without any restrictions on the number of calls to f , we can write the specification (let us call it S) as follows:

$$(f \text{ is constant} \vee f \text{ is balanced}) \implies b' = f \text{ is constant}$$

where b is a boolean variable and the informally stated properties of f are defined formally as follows:

$$\begin{aligned}
f \text{ is constant} &= \forall i : 0, \dots, 2^n \cdot f_i = f_0 \\
f \text{ is balanced} &= \left| \sum_{i : 0, \dots, 2^n} (-1)^{f_i} \right| = 0
\end{aligned}$$

It is easy to show that

$$(f \text{ is constant} \vee f \text{ is balanced}) \implies (f \text{ is constant} \iff \forall i : 0, \dots, 2^n - 1 \cdot f_i = f_0)$$

In other words, given that the function is either constant or it balanced, the function is constant if and only if half of its values plus one are the same.

Our task is now a little simpler: we need to implement the specification R defined as follows:

$$b' \iff \forall i : 0, \dots, 2^n - 1 \cdot f_i = f_0$$

The idea for a solution is simple. We query f with values of x from the domain $0, \dots, 2^n$, until we either get two different values of f , in which case we conclude that the function is balanced, or we have queried f with more than half of possible values for x and got the same value of f each time, in which case we conclude that f is constant. The program, in integer variables x and y , is:

$$R \longleftarrow y := f_0; x := 1; P$$

$$P \iff \forall i : 0, \dots, x \cdot (f_i = y) \implies R$$

$$P \longleftarrow \text{if } y \neq f_x \text{ then } b := \perp$$

$$\text{else if } x = 2^{n-1} \text{ then } b := \top \text{ else } (x := x + 1; P)$$

Interested readers can find the detailed proof in Appendix C.1.

The quantum solution is a direct generalisation of Deutsch's algorithm. We introduce a variable r from the domain $0, \dots, 2^n$, a quantum state $\psi : 0, \dots, 2^n \rightarrow \mathbb{C}$ and make use of the generalised quantum oracle:

$$U_f = \lambda \psi : 0, \dots, 2^n \rightarrow \mathbb{C} \cdot x : 0, \dots, 2^n \cdot (-1)^{f_x} \times \psi x$$

The idea is to create a suitable superposition for state ψ , so that a measurement of ψ produces 0 if and only if f is constant, so that:

$$S \longleftarrow Q; b := (r = 0) \quad , \text{ where}$$

$$Q \iff f \text{ is constant} \vee f \text{ is balanced} \implies f \text{ is constant} = (r' = 0)$$

To implement Q we notice that:

$$f \text{ is constant} = \left(\left| \sum x \cdot (-1)^{fx}/2^n \right| = 1 \right) \quad (6.1)$$

$$f \text{ is balanced} = \left(\left| \sum x \cdot (-1)^{fx}/2^n \right| = 0 \right) \quad (6.2)$$

Introducing variables x, y , and z from the domain $0, \dots, 2^n$ and assuming f is constant \vee f is balanced, we proceed:

$$\begin{aligned} f \text{ is constant} &= (r' = 0) \\ &= \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) 0 \right|^2 \times (r' = 0) + \\ &\quad \left(\sum y : 1, \dots, 2^n \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) y \right|^2 \right) \times (r' \neq 0) \end{aligned}$$

where $\mathbf{x} \cdot \mathbf{z}$ is the bitwise product of \mathbf{x} and \mathbf{z} . To understand this step we need to analyse the state $\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle$. Since $|\mathbf{z}\rangle$ applied to 0 is 1 for $z = 0$ and 0 for all other values of z , the application $\left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) 0$ can be simplified to $\sum x \cdot (-1)^{fx}/2^n$. Thus, $\left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) 0 \right|^2$ is 1, if f is constant, and it is 0, otherwise. On the other hand, $\left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) y \right|^2$ is non-zero only when $z = y$, so that the sum over non-zero values of y reduces to $\left| \sum y : 1, \dots, 2^n \sum x \cdot (-1)^{x \cdot y + fx}/2^n \right|^2$. Since $\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle$ is a valid quantum state, whenever f is constant (and thus $\left| \sum x \cdot (-1)^{fx}/2^n \right| = 1$), all other amplitudes are 0, i.e. $\left| \sum y : 1, \dots, 2^n \sum x \cdot (-1)^{x \cdot y + fx}/2^n \right|^2 = 0$. When f is balanced, the squares of these amplitudes must sum to 1, and so $\left| \sum y : 1, \dots, 2^n \sum x \cdot (-1)^{x \cdot y + fx}/2^n \right|^2 = 1$.

Now that we have some intuition about why the two expressions are equal, we can produce a formal proof of this step:

$$\begin{aligned} &\left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) 0 \right|^2 \times (r' = 0) + && \text{unitarity of} \\ &\left(\sum y : 1, \dots, 2^n \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) y \right|^2 \right) \times (r' \neq 0) && \text{quantum state} \\ &= \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) 0 \right|^2 \times (r' = 0) + \\ &\quad \left(1 - \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx}/2^n \times |\mathbf{z}\rangle \right) 0 \right|^2 \right) \times (r' \neq 0) && \text{definition of } |\mathbf{z}\rangle \end{aligned}$$

$$\begin{aligned}
&= \left| \sum x \cdot (-1)^{fx/2^n} \right|^2 \times (r' = 0) + \\
&\quad \left(1 - \left| \sum x \cdot (-1)^{fx/2^n} \right|^2 \right) \times (r' \neq 0) \quad \text{from (6.1)} \\
&= f \text{ is constant} = (r' = 0)
\end{aligned}$$

which concludes the proof of this step. We continue as follows:

$$\begin{aligned}
&\left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx/2^n} \times |\mathbf{z}\rangle \right) 0 \right|^2 \times (r' = 0) + \\
&\quad \left(\sum y : 1, \dots, 2^n \cdot \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx/2^n} \times |\mathbf{z}\rangle \right) y \right|^2 \right) \times (r' \neq 0) \quad \text{arithmetic} \\
&\geq \left| \left(\sum z, x \cdot (-1)^{x \cdot z + fx/2^n} \times |\mathbf{z}\rangle \right) r' \right|^2 \quad \text{measure} \\
&= \mathbf{measure} \left(\sum z, x \cdot (-1)^{x \cdot z + fx/2^n} \times |\mathbf{z}\rangle \right) r \quad \text{arithmetic} \\
&= \mathbf{measure} \left(\sum x \cdot (-1)^{fx/\sqrt{2^n}} \times \left(\sum z \cdot (-1)^{x \cdot z/\sqrt{2^n}} \times |\mathbf{z}\rangle \right) \right) r \quad \text{Hadamard} \\
&= \mathbf{measure} \left(\sum x \cdot (-1)^{fx/\sqrt{2^n}} \times (H^{\otimes n} |\mathbf{x}\rangle) \right) r \quad \text{linearity} \\
&= \mathbf{measure} \left(H^{\otimes n} \left(\sum x \cdot (-1)^{fx/\sqrt{2^n}} \times |\mathbf{x}\rangle \right) \right) r \quad \text{Oracle} \\
&= \mathbf{measure} \left(H^{\otimes n} \left(\sum x \cdot U_f |\mathbf{x}\rangle / \sqrt{2^n} \right) \right) r \quad \text{linearity} \\
&= \mathbf{measure} \left(H^{\otimes n} \left(U_f \left(\sum x \cdot |\mathbf{x}\rangle / \sqrt{2^n} \right) \right) \right) r \quad \text{Hadamard} \\
&= \mathbf{measure} (H^{\otimes n} (U_f (H^{\otimes n} |0\rangle^{\otimes n}))) r \quad \text{substitute} \\
&= q := |0\rangle^{\otimes n}; q := H^{\otimes n} q; q := U_f q; q := H^{\otimes n} q; \mathbf{measure} q r
\end{aligned}$$

The complete solution is:

$$q := |0\rangle^{\otimes n}; q := H^{\otimes n} q; q := U_f q; q := H^{\otimes n} q; \mathbf{measure} q r; b := (r' = 0)$$

Let us add to the specification a restriction on the number of calls to the oracle. Suppose the new specification is:

$$(f \text{ is constant} \vee f \text{ is balanced} \implies b' = f \text{ is constant}) \wedge (t' = t + 1)$$

where we charge 1 unit of time for each call to the oracle and all other operations are free. Clearly, the above quantum solution works. Classically the specification is

unimplementable. In fact, the strongest classically implementable specification is

$$(f \text{ is constant} \vee f \text{ is balanced} \implies b' = f \text{ is constant}) \wedge (t' = t + 2^{n-1} + 1)$$

which is implemented by the above classical program. The proof of refinement can be found in Appendix C.2. In other words, the quantum solution offers an exponential speed-up.

Deutsch-Josza algorithm is a powerful illustration of the potential of quantum computation.

6.5 Grover's search

Grover's quantum search algorithm [6] is well-known for the quadratic speed-up it offers in the solutions of NP-complete problems. The algorithm is optimal up to a multiplicative constant [3]. The task is: given a function $f : 0, \dots, 2^n \rightarrow 0, 1$, find $x : 0, \dots, 2^n$, such that $f x = 1$. For simplicity we assume that there is only a single solution, which we denote x_1 , i.e. $f x_1 = 1$ and $f x = 0$ for all $x \neq x_1$. The proofs are not very different for a general case of more than one solutions.

As before, we use a general quantum oracle, defined by

$$U_f |\mathbf{x}\rangle = (-1)^{f x} \times |\mathbf{x}\rangle$$

In addition, we define the *inversion about mean* operator as follows:

$$M : (0, \dots, N \rightarrow \mathbb{C}) \rightarrow (0, \dots, N \rightarrow \mathbb{C})$$

$$M \psi = \lambda x : 0, \dots, N \cdot 2 \times \left(\sum i : 0, \dots, N \cdot \psi i / N \right) - \psi x$$

where $N = 2^n$.

Grover's algorithm initialises the quantum system to an equally weighted superposition of all basis states $|\mathbf{x}\rangle$, $x : 0, \dots, N$. It then repeatedly applies U_f followed by M to

the system. Finally, the state is measured. The probability of error is determined by the number of iterations performed by the algorithm.

The algorithm can be understood in a beautiful way with the help of a geometric analysis of the operators. Let α be the sum over all x , which are not solutions, and let β be the solution:

$$\alpha = \frac{1}{\sqrt{N-1}} \times \sum_{x \neq x_1} |x\rangle$$

$$\beta = |x_1\rangle$$

Then the oracle U_f performs a *reflection* about the vector α in the plane defined by α and β (see figure 6.1). In other words, $U_f(a \times \alpha + b \times \beta) = a \times \alpha - b \times \beta$. Similarly, the inversion about mean operator is a reflection about the vector ψ in the plane defined by α and β . Therefore, the result of U_f followed by M is a *rotation* in this plane. We define θ to be the rotation angle:

$$\theta = 2 \times \arcsin \sqrt{1/N}$$

Since each rotation leaves us in the plane defined by α and β , then the state of the system after i rotations by θ radians is:

$$\psi_i = \cos((2 \times i + 1) \times \theta/2) \times \alpha + \sin((2 \times i + 1) \times \theta/2) \times \beta$$

Suppose we charge one unit of time for each call to the oracle and all other operations are free. Consider the specification:

$$S = \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \times (r' = x_1) +$$

$$\left(1 - \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \right) \times (r' \neq x_1) / (N - 1)$$

where r is the result variable from the domain $0, \dots, N$. The specification says that we want the solution $\left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2$ of the time, where $t' - t$ is the number of times we use the oracle. We show that Grover's algorithm implements this specification.

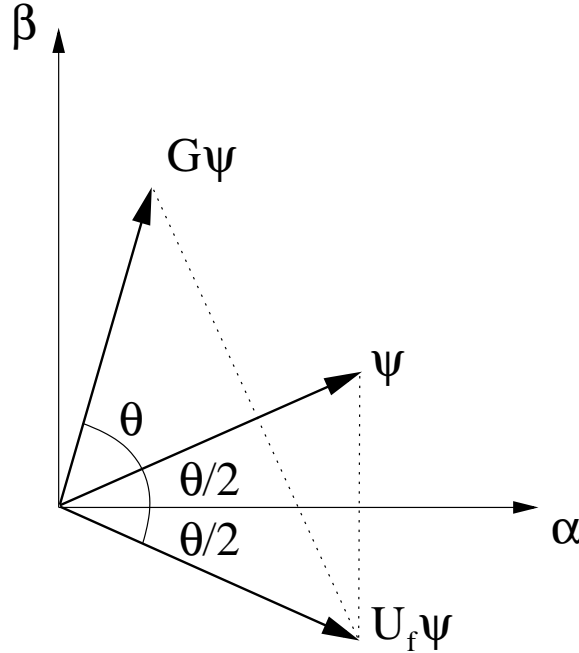


Figure 6.1: Grover's search algorithm

As usual, we want to specify the quantum state that, when measured, gives the desired distribution. With a quantum state variable $\psi : 0, \dots, N \rightarrow \mathbb{C}$:

$$\begin{aligned}
 S &= \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \times (r' = x_1) + \\
 &\quad \left(1 - \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \right) / \\
 &\quad (N - 1) \times (r' \neq x_1) \qquad \text{trigonometry} \\
 &= \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \times (r' = x_1) + \\
 &\quad \left(\cos \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 / (N - 1) \times (r' \neq x_1) \qquad \text{arithmetic} \\
 &= \left| \left(\sin((2 \times (t' - t) + 1) \times \theta/2) \times |\mathbf{x}_1\rangle + \right. \right. \\
 &\quad \left. \left. \sum_{x \neq x_1} x \cdot \cos((2 \times (t' - t) + 1) \times \theta/2) \times |\mathbf{x}\rangle / \sqrt{N-1} \right) r' \right|^2 \qquad \text{measure} \\
 &= \psi' = \sin((2 \times (t' - t) + 1) \times \theta/2) \times |\mathbf{x}_1\rangle + \\
 &\quad \cos((2 \times (t' - t) + 1) \times \theta/2) \times \sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle / \sqrt{N-1}; \\
 &\quad \text{measure } \psi r \qquad \qquad \qquad \text{def. of } \alpha \&\beta
 \end{aligned}$$

$$\begin{aligned} &= \psi' = \sin((2 \times (t' - t) + 1) \times \theta/2) \times \beta + \\ &\quad \cos((2 \times (t' - t) + 1) \times \theta/2) \times \alpha; \end{aligned}$$

measure ψ r

Let P be the description of the quantum state immediately before the measurement.

$$\begin{aligned} P &= \psi' = \sin((2 \times (t' - t) + 1) \times \theta/2) \times \beta + \cos((2 \times (t' - t) + 1) \times \theta/2) \times \alpha \\ &= \psi' = \psi_{t'-t} \end{aligned}$$

Since $\psi_{t'-t}$ is the state obtained by $t' - t$ rotations by θ radians, we define the specification R to describe the rotation. With a natural k that represents the number of iterations performed:

$$R = \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i$$

Adding initialisation, we prove:

$$\begin{aligned} P &\Leftarrow \psi' = \psi_k \wedge t' = t + k && \text{substitutions} \\ &= i := 0; \psi := \psi_0; \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i && \text{definition of } R \\ &= i := 0; \psi := \psi_0; R \end{aligned}$$

Our task has been simplified. We now need to implement $\psi := \psi_0$ and R and we are done. To implement the assignment, we note that ψ_0 is an equally weighted superposition of all basis states $|\mathbf{x}\rangle$ for $x : 0, ..N$, and we already know that this superposition can be produced by the application of n Hadamards to the zero state of an n -qubit system. In other words,

$$\begin{aligned} \psi_0 &= \cos((2 \times 0 + 1) \times \theta/2) \times \alpha + \sin((2 \times 0 + 1) \times \theta/2) \times \beta && \text{def. of } \theta \\ &= \cos(\arcsin \sqrt{1/N}) \times \alpha + \sin(\arcsin \sqrt{1/N}) \times \beta && \text{trigonometry} \\ &= \alpha \times \sqrt{N-1}/\sqrt{N} + \beta/\sqrt{N} && \text{def. of } \alpha \text{ and } \beta \\ &= \sum x \neq x_1 \cdot |\mathbf{x}\rangle/\sqrt{N} + |\mathbf{x}_1\rangle/\sqrt{N} && \text{arithmetic} \end{aligned}$$

$$\begin{aligned}
&= \sum x \cdot |\mathbf{x}\rangle / \sqrt{N} && \text{Hadamard} \\
&= H^{\otimes n} |0\rangle^{\otimes n}
\end{aligned}$$

Then, using the Substitution Law:

$$P \Leftarrow i := 0; \psi := |0\rangle^{\otimes n}; \psi := H^{\otimes n}\psi; R$$

Having understood the geometry of Grover's algorithm, implementing R is easy. Adding the time increment before the call to the oracle, we have:

$$R \Leftarrow \mathbf{if } i = k \mathbf{ then } ok \mathbf{ else } (i := i + 1; t := t + 1; \psi := U_f\psi; \psi := M\psi; R)$$

Proving the refinement is a bit of work. Firstly, let us analyse the action of the oracle followed by the inversion about mean on the state in the α, β plane. On the general state ψ , the action of the operators is:

$$\begin{aligned}
M(U_f\psi) &= \lambda x \cdot 2 \times \left(\sum i \cdot (U_f\psi)_i / N \right) - (U_f\psi)_x && \text{oracle} \\
&= \lambda x \cdot \frac{2}{N} \times \left(\sum i \cdot (-1)^{f_i} \times \psi_i \right) - (-1)^{f_x} \times \psi_x && \text{action of } f \\
&= \lambda x \cdot \frac{2}{N} \times \left(\sum i \cdot (-1)^{i=x_1} \times \psi_i \right) - (-1)^{x=x_1} \times \psi_x && \text{arithmetic} \\
&= \lambda x \cdot \frac{2}{N} \times \left(\sum i \neq x_1 \cdot \psi_i \right) - \frac{2}{N} \times \psi_{x_1} - (-1)^{x=x_1} \times \psi_x \\
&= \lambda x \cdot \mathbf{if } x = x_1 \mathbf{ then } \frac{2}{N} \times \left(\sum i \neq x_1 \cdot \psi_i \right) + \frac{N-2}{N} \times \psi_{x_1} \\
&\quad \mathbf{else } \frac{2}{N} \times \left(\sum i \neq x_1 \cdot \psi_i \right) - \frac{2}{N} \times \psi_{x_1} - \psi_x
\end{aligned}$$

Therefore, the action of the operators on the solution state $|\mathbf{x}_1\rangle$ is:

$$\begin{aligned}
M(U_f|\mathbf{x}_1\rangle) &= \lambda x \cdot \mathbf{if } x = x_1 \mathbf{ then } \frac{N-2}{N} \mathbf{ else } -\frac{2}{N} \\
&= \frac{N-2}{N} \times |\mathbf{x}_1\rangle - \frac{2}{N} \times \sum x \neq x_1 \cdot |\mathbf{x}\rangle
\end{aligned}$$

and the action on the non-solutions state is:

$$M\left(U_f\left(\sum x \neq x_1 \cdot |\mathbf{x}\rangle\right)\right) = \lambda x \cdot \mathbf{if } x = x_1 \mathbf{ then } \frac{2 \times (N-1)}{N} \mathbf{ else } \frac{N-2}{N}$$

$$= \frac{2 \times (N-1)}{N} \times |\mathbf{x}_1\rangle + \frac{N-2}{N} \times \sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle$$

Then the action of the two operators on the state ψ_i in the α, β plane is:

$$\begin{aligned}
& M(U_f \psi_i) \\
&= M \left(U_f \left(\cos((2 \times i + 1) \times \theta/2) / \sqrt{N-1} \times \left(\sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle \right) + \right. \right. \\
&\quad \left. \left. \sin((2 \times i + 1) \times \theta/2) \times |\mathbf{x}_1\rangle \right) \right) \quad \text{linearity} \\
&= \cos((2 \times i + 1) \times \theta/2) / \sqrt{N-1} \times M \left(U_f \left(\sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle \right) \right) + \\
&\quad \sin((2 \times i + 1) \times \theta/2) \times M(U_f(|\mathbf{x}_1\rangle)) \quad \text{action of } M(U_f) \\
&= \cos((2 \times i + 1) \times \theta/2) / \sqrt{N-1} \times \\
&\quad \left(\frac{2 \times (N-1)}{N} \times |\mathbf{x}_1\rangle + \frac{N-2}{N} \times \sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle \right) + \\
&\quad \sin((2 \times i + 1) \times \theta/2) \times \\
&\quad \left(\frac{N-2}{N} \times |\mathbf{x}_1\rangle - \frac{2}{N} \times \sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle \right) \quad \text{arithmetic} \\
&= \left(\cos((2 \times i + 1) \times \theta/2) \times \frac{2 \times \sqrt{N-1}}{N} + \right. \\
&\quad \left. \sin((2 \times i + 1) \times \theta/2) \times \frac{N-2}{N} \right) \times |\mathbf{x}_1\rangle + \\
&\quad \left(\cos((2 \times i + 1) \times \theta/2) \times \frac{N-2}{N} - \right. \\
&\quad \left. \sin((2 \times i + 1) \times \theta/2) \times \frac{2 \times \sqrt{N-1}}{N} \right) / \\
&\quad \sqrt{N-1} \times \sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle
\end{aligned}$$

We now recall some trigonometry to prove that this state is identical to the state of the system after $i + 1$ rotations by θ radians.

$$\begin{aligned}
& \cos((2 \times i + 1) \times \theta/2) \times \frac{N-2}{N} - \\
& \sin((2 \times i + 1) \times \theta/2) \times \frac{2 \times \sqrt{N-1}}{N} \quad \text{definition of } \theta \\
&= \cos((2 \times i + 1) \times \theta/2) \times (2 \times \cos^2(\theta/2) - 1) - \\
& \sin((2 \times i + 1) \times \theta/2) \times (2 \times \sin(\theta/2) \times \cos(\theta/2)) \quad \text{trigonometry}
\end{aligned}$$

$$\begin{aligned}
&= \cos((2 \times i + 1) \times \theta/2) \times \cos \theta - \\
&\quad \sin((2 \times i + 1) \times \theta/2) \times \sin \theta && \text{trigonometry} \\
&= \cos((2 \times i + 1) \times \theta/2 + \theta) \\
&= \cos((2 \times (i + 1) + 1) \times \theta/2)
\end{aligned}$$

and similarly

$$\begin{aligned}
&\cos((2 \times i + 1) \times \theta/2) \times \frac{2 \times \sqrt{N-1}}{N} + \\
&\sin((2 \times i + 1) \times \theta/2) \times \frac{N-2}{N} && \text{definition of } \theta \\
&= \sin((2 \times i + 1) \times \theta/2) \times (2 \times \cos^2(\theta/2) - 1) + \\
&\quad \cos((2 \times i + 1) \times \theta/2) \times (2 \times \sin(\theta/2) \times \cos(\theta/2)) && \text{trigonometry} \\
&= \sin((2 \times i + 1) \times \theta/2) \times \cos \theta + \\
&\quad \cos((2 \times i + 1) \times \theta/2) \times \sin \theta && \text{trigonometry} \\
&= \sin((2 \times i + 1) \times \theta/2 + \theta) \\
&= \sin((2 \times (i + 1) + 1) \times \theta/2)
\end{aligned}$$

so that

$$\begin{aligned}
M(U_f \psi_i) &= \sin((2 \times (i + 1) + 1) \times \theta/2) \times |\mathbf{x}_1\rangle + \\
&\quad \cos((2 \times (i + 1) + 1) \times \theta/2) / \sqrt{N-1} \times \sum_{x \neq x_1} x \cdot |\mathbf{x}\rangle && (6.3) \\
&= \psi_{i+1}
\end{aligned}$$

Finally, we are ready to prove the refinement. Starting with the right side:

$$\begin{aligned}
&\mathbf{if } i = k \mathbf{ then } ok \\
&\mathbf{else } (i := i + 1; t := t + 1; \psi := U_f \psi; \psi := M\psi; R) && \text{expand } ok, R \\
&= \mathbf{if } i = k \mathbf{ then } \psi' = \psi \wedge t' = t \wedge i' = i \wedge k' = k \\
&\mathbf{else } (i := i + 1; t := t + 1; \psi := U_f \psi; \psi := M\psi; && \text{substitute, weaken} \\
&\quad \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i)
\end{aligned}$$

$$\begin{aligned}
&\implies \mathbf{if } i = k \mathbf{ then } \psi' = \psi \wedge t' = t \\
&\quad \mathbf{else } M(U_f \psi) = \psi_{i+1} \Rightarrow \psi' = \psi_k \wedge t' = (t + 1) + k - (i + 1) \quad \text{math, weaken} \\
&\implies \mathbf{if } i = k \mathbf{ then } \psi = \psi_i \Rightarrow \psi' = \psi \wedge t' = t + k - i \\
&\quad \mathbf{else } \psi = \psi_i \wedge M(U_f \psi) = \psi_{i+1} \Rightarrow \psi' = \psi_k \wedge t' = t + k - i \quad \text{context} \\
&\implies \mathbf{if } i = k \mathbf{ then } \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i \\
&\quad \mathbf{else } \psi = \psi_i \wedge M(U_f \psi_i) = \psi_{i+1} \Rightarrow \psi' = \psi_k \wedge t' = t + k - i \quad \text{from (6.3)} \\
&\implies \mathbf{if } i = k \mathbf{ then } \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i \\
&\quad \mathbf{else } \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i \quad \text{case idempotence} \\
&\implies \psi = \psi_i \Rightarrow \psi' = \psi_k \wedge t' = t + k - i
\end{aligned}$$

In summary,

$$\begin{aligned}
S &= \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \times (r' = x_1) + \\
&\quad \left(1 - \left(\sin \left((2 \times (t' - t) + 1) \times \arcsin \sqrt{1/N} \right) \right)^2 \right) \times (r' \neq x_1) / (N - 1) \\
&= P; \mathbf{measure } \psi \ r \\
P &\Leftarrow i := 0; \psi := |0\rangle^{\otimes n}; \psi := H^{\otimes n} \psi; R \\
R &\Leftarrow \mathbf{if } i = k \mathbf{ then } ok \mathbf{ else } (i := i + 1; t := t + 1; \psi := U_f \psi; \psi := M \psi; R)
\end{aligned}$$

Specification S carries a lot of useful information. For example, it tells us that the probability of finding a solution after k iterations is $\left(\sin((2 \times k + 1) \times \arcsin \sqrt{1/N}) \right)^2$. Or we might ask how many iterations should be performed to minimise the probability of an error. Examining first and second derivatives, we find that the above probability is minimised when $t' - t = (\pi \times i) / (4 \times \arcsin \sqrt{1/N}) - 1/2$ for integer i . Of course, the number of iterations performed must be a natural number. It is interesting to note that probability of error is periodic in the number of iterations, but since we don't gain anything by performing extra iterations, we pick $i = 1$. Finally, assuming $1 \ll N = 2^n$, we obtain an elegant approximation to the optimal number of iterations: $\lceil \pi \times \sqrt{2^n} / 4 \rceil$,

with the probability of error approximately $1/2^n$.

In contrast, the best a classical algorithm can do to solve the problem is to randomly sample elements from $0, \dots, 2^n$ and check the value of the oracle f on each element. To match Grover's algorithm, such a classical algorithm requires $\Theta(2^n)$ calls to the oracle.

6.6 Computing with Mixed States

As we have discussed in section 3.2, the state of a quantum system after a measurement is traditionally described as a *mixed state*. An equation $\psi = \{|0\rangle\}/2 + \{|1\rangle\}/2$ should be understood as follows: the state ψ is $|0\rangle$ with probability $1/2$ and it is $|1\rangle$ with probability $1/2$. In contrast to a pure state, a mixed state does not describe a physical state of the system. Rather, it describes our knowledge of in what state the system is. Writing $\psi = \{|0\rangle\}/2 + \{|1\rangle\}/2$ is inconsistent, since, in addition to the ambiguity in the notation $\{\}$ also used for sets, $\{|0\rangle\}/2 + \{|1\rangle\}/2$ is not in the domain of ψ . Moreover, unitary operations are not applicable to mixed states. Traditionally, the application of operators is extended linearly: $U(\{|0\rangle\}/2 + \{|1\rangle\}/2) = \{U|0\rangle\}/2 + \{U|1\rangle\}/2$, which is also inconsistent.

In our framework, there is no need for an additional mechanism to compute with mixed states. Indeed, a mixed state is not a system state, but a distribution over system states, and all our programming notions apply to distributions. The above mixed state is the following distribution over a quantum state ψ : $(\psi = |0\rangle)/2 + (\psi = |1\rangle)/2$. This expression tells us, for each possible value in the domain of ψ , the probability of ψ having that value. For example, ψ is the state $|0\rangle$ with probability $(|0\rangle = |0\rangle)/2 + (|0\rangle = |1\rangle)/2$, which is $1/2$, it is $|1\rangle$ with probability $(|1\rangle = |0\rangle)/2 + (|1\rangle = |1\rangle)/2$, which is also $1/2$, and for any non-zero scalars α and β , ψ is $\alpha \times |0\rangle + \beta \times |1\rangle$ with probability $(\alpha \times |0\rangle + \beta \times |1\rangle = |0\rangle)/2 + (\alpha \times |0\rangle + \beta \times |1\rangle = |1\rangle)/2$, which is 0. One way to obtain

this distribution is to measure an equally weighted superposition of $|0\rangle$ and $|1\rangle$:

$$\begin{aligned}
& \psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}; \text{ **measure** } \psi \ r && \text{measure} \\
& \equiv \psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}; |\psi r'\|^2 \times (\psi' = |\mathbf{r}'\rangle) && \text{sequential composition} \\
& \equiv \sum r'', \psi'' \cdot (\psi'' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \times |\psi'' r'\|^2 \times (\psi' = |\mathbf{r}'\rangle) && \text{one point law} \\
& \equiv (|0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) r'\|^2 \times (\psi' = |\mathbf{r}'\rangle) \\
& \equiv (\psi' = |\mathbf{r}'\rangle)/2
\end{aligned}$$

Distribution of the quantum state is then:

$$\sum r' \cdot (\psi' = |\mathbf{r}'\rangle)/2 = (\psi' = |0\rangle)/2 + (\psi' = |1\rangle)/2$$

as desired.

Similarly, there is no need to extend the application of unitary operators. Consider the following toy program:

$$\psi := |0\rangle; \psi := H\psi; \text{ **measure** } \psi \ r; \text{ **if** } r = 0 \text{ **then** } \psi := H\psi \text{ **else** } ok$$

In the second application of Hadamard the quantum state is mixed, but this is not evident from the syntax of the program. It is only in the analysis of the final quantum state that the notion of a mixed state is meaningful. The operator is applied to a (pure) system state, though we are unsure what that state is.

$$\begin{aligned}
& \psi := |0\rangle; \psi := H\psi; \text{ **measure** } \psi \ r; \\
& \text{ **if** } r = 0 \text{ **then** } \psi := H\psi \text{ **else** } ok && \text{as before} \\
& \equiv (\psi' = |\mathbf{r}'\rangle)/2; \\
& \text{ **if** } r = 0 \text{ **then** } \psi := H\psi \text{ **else** } ok && \text{sequential composition} \\
& \equiv \sum r'', \psi'' \cdot (\psi'' = |\mathbf{r}''\rangle)/2 \times \\
& \quad ((r'' = 0) \times (\psi' = H\psi'') \times (r' = r'')) + \\
& \quad (r'' = 1) \times (\psi' = \psi'') \times (r' = r'') && \text{one point law}
\end{aligned}$$

$$\begin{aligned}
&= ((\psi' = H|0\rangle) \times (r' = 0) + (\psi' = |1\rangle) \times (r' = 1)) / 2 \\
&= (\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \times (r' = 0)/2 + \\
&\quad (\psi' = |1\rangle) \times (r' = 1)/2
\end{aligned}$$

The distribution of the quantum state after the computation is:

$$\begin{aligned}
&\sum r' \cdot (\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2}) \times (r' = 0)/2 + (\psi' = |1\rangle) \times (r' = 1)/2 \\
&= (\psi' = |0\rangle/\sqrt{2} + |1\rangle/\sqrt{2})/2 + (\psi' = |1\rangle)/2
\end{aligned}$$

A lot of properties of measurements and mixed states can be proven from the definitions of measurement and sequential composition. For example, the fact that a measurement in the computational basis, performed immediately following a measurement in the same basis, does not change the state of the system and yields the same result as the first measurement with probability 1, is proven as follows:

$$\begin{aligned}
&\mathbf{measure} \ \psi \ r; \ \mathbf{measure} \ \psi \ r && \mathbf{measure} \\
&= |\psi \ r'|^2 \times (\psi' = |\mathbf{r}'\rangle); |\psi \ r'|^2 \times (\psi' = |\mathbf{r}'\rangle) && \mathbf{sequential \ composition} \\
&= \sum \psi'', r'' \cdot |\psi \ r''|^2 \times (\psi'' = |\mathbf{r}''\rangle) \times |\psi'' \ r'|^2 \times (\psi' = |\mathbf{r}'\rangle) && \mathbf{one \ point \ law} \\
&= |\psi \ r'|^2 \times (\psi' = |\mathbf{r}'\rangle) && \mathbf{measure} \\
&= \mathbf{measure} \ \psi \ r
\end{aligned}$$

In case of a general quantum measurement, the proof is similar, but a little more computationally involved.

Recall the example from section 4.2.2:

$$\begin{aligned}
P &\Leftarrow \mathbf{if} \ x = 0 \ \mathbf{then} \ ok \ \mathbf{else} \ (x := x - rand \ 2; \ t := t + 1; \ P) \\
P &= (x \geq 0 \Rightarrow x' = 0) \wedge (t' \geq t + x)
\end{aligned}$$

Let us replace the magical *rand* with a measurement that results in 0 and 1 with prob-

ability $1/2$ each. Introducing a quantum state ψ and $r : 0, 1$:

$$P \Leftarrow \mathbf{if} \ x = 0 \ \mathbf{then} \ ok$$

$$\quad \mathbf{else} \ (\psi := |0\rangle; \ \psi := H\psi; \ \mathbf{measure} \ \psi \ r; \ x := x - r; \ t := t + 1; \ P)$$

We need to prove that the new program is correct. In other words, we need to show that the quantum program also satisfies the specification $x \geq 0 \Rightarrow x' = 0$. Starting with the right side and ignoring time for now:

$$\mathbf{if} \ x = 0 \ \mathbf{then} \ ok$$

$$\mathbf{else} \ (\psi := |0\rangle; \ \psi := H\psi; \ \mathbf{measure} \ \psi \ r;$$

$$\quad x := x - r; \ x \geq 0 \Rightarrow x' = 0) \quad \text{measure}$$

$$\equiv \mathbf{if} \ x = 0 \ \mathbf{then} \ ok$$

$$\mathbf{else} \ (\psi := |0\rangle; \ \psi := H\psi; \ |\psi r'\|^2 \times (\psi' = |\mathbf{r}'\rangle) \times (x' = x); \quad \text{substitute,}$$

$$\quad x := x - r; \ x \geq 0 \Rightarrow x' = 0) \quad \text{apply } H|0\rangle$$

$$\equiv \mathbf{if} \ x = 0 \ \mathbf{then} \ ok$$

$$\mathbf{else} \ ((\psi' = |\mathbf{r}'\rangle) \times (x' = x)/2; \ x \geq r \Rightarrow x' = 0) \quad \text{seq. composition}$$

$$\equiv \mathbf{if} \ x = 0 \ \mathbf{then} \ ok$$

$$\mathbf{else} \ \sum \psi'', r'', x'' \cdot (\psi'' = |\mathbf{r}''\rangle) \times (x'' = x)/2 \times \quad \text{sum}$$

$$\quad (x'' \geq r'' \Rightarrow x' = 0)$$

$$\equiv \mathbf{if} \ x = 0 \ \mathbf{then} \ ok$$

$$\mathbf{else} \ (x \geq 0 \Rightarrow x' = 0)/2 + (x \geq 1 \Rightarrow x' = 0)/2 \quad \text{if then else , ok}$$

$$\equiv (x = 0) \times (x' = x) +$$

$$(x \neq 0) \times (x < 0)/2 + (x \neq 0) \times (x' = 0)/2 +$$

$$(x \neq 0) \times (x < 1)/2 + (x \neq 0) \times (x' = 0)/2 \quad \text{arithmetic}$$

$$\equiv (x \geq 0 \Rightarrow x' = 0)$$

We expect the distribution of x' and t' to be the same as in section 4.2.2, namely:

$$(0 = x' = x = t' - t) + (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t' - t}}$$

Let us prove it. Adding time and starting with the right side:

$$\begin{aligned} & \sum \psi', r' \cdot \mathbf{if} \ x = 0 \ \mathbf{then} \ ok \\ & \quad \mathbf{else} \ \left(\begin{array}{l} \psi := |0\rangle; \ \psi := H\psi; \ \mathbf{measure} \ \psi \ r; \\ x := x - r; \ t := t + 1; \end{array} \right. \quad \text{measure,} \\ & \quad (0 = x' = x = t' - t) + \quad \text{substitute,} \\ & \quad (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t' - t}} \quad \text{apply } H|0\rangle \\ = & \sum \psi', r' \cdot \mathbf{if} \ x = 0 \ \mathbf{then} \ ok \\ & \quad \mathbf{else} \ \left(\begin{array}{l} (\psi' = |\mathbf{r}'\rangle) \times (t' = t) \times (x' = x)/2; \\ (0 = x' = x - r = t' - t - 1) + \\ (0 = x' < x - r \leq t' - t - 1) \times \\ \binom{t' - t - 2}{x - r - 1} \times \frac{1}{2^{t' - t - 1}} \end{array} \right. \quad \text{sequential} \\ & \quad \text{composition} \\ = & \sum \psi', r' \cdot \mathbf{if} \ x = 0 \ \mathbf{then} \ ok \\ & \quad \mathbf{else} \ \sum \psi'', r'', t'', x'' \cdot (\psi'' = |\mathbf{r}''\rangle) \times (t'' = t) \times (x'' = x)/2 \times \\ & \quad (0 = x' = x'' - r'' = t' - t'' - 1) + \\ & \quad (0 = x' < x'' - r'' \leq t' - t'' - 1) \times \\ & \quad \binom{t' - t'' - 2}{x'' - r'' - 1} \times \frac{1}{2^{t' - t'' - 1}} \quad \text{sum} \end{aligned}$$

$$\begin{aligned}
&= \sum \psi', r' \cdot \mathbf{if } x = 0 \mathbf{ then } ok \\
&\quad \mathbf{else} \left((0 = x' = x = t' - t - 1)/2 + \right. \\
&\quad \quad (0 = x' = x - 1 = t' - t - 1)/2 + \\
&\quad \quad (0 = x' < x \leq t' - t - 1) \times \\
&\quad \quad \left. \binom{t' - t - 2}{x - 1} \times \frac{1}{2^{t'-t}} + \right. \\
&\quad \quad (0 = x' < x - 1 \leq t' - t - 1) \times \\
&\quad \quad \left. \binom{t' - t - 2}{x - 2} \times \frac{1}{2^{t'-t}} \right) \quad \text{math} \\
&= \sum \psi', r' \cdot \mathbf{if } x = 0 \mathbf{ then } ok \\
&\quad \mathbf{else} \left(\frac{1}{2^{t'-t}} \times (0 = x' = x = t' - t - 1) + \right. \quad \text{expand } ok, \\
&\quad \quad \left. \frac{1}{2^{t'-t}} \times (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \right) \quad \text{if then else} \\
&= \sum \psi', r' \cdot \left((x = 0) \times (x' = x) \times (t' = t) \times (\psi' = \psi) \times (r' = r) + \right. \\
&\quad \quad (x \neq 0) \times \frac{1}{2^{t'-t}} \times (0 = x' = x = t' - t - 1) + \quad \text{sum,} \\
&\quad \quad \left. (x \neq 0) \times \frac{1}{2^{t'-t}} \times (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \right) \quad \text{simplify} \\
&= (0 = x' = x = t' - t) + \\
&\quad (0 = x' < x \leq t' - t) \times \binom{t' - t - 1}{x - 1} \times \frac{1}{2^{t'-t}}
\end{aligned}$$

This concludes the proof. As before, we should expect to wait $2 \times x$ time units for the computation to complete.

Chapter 7

Conclusion and Future Work

We have presented a new approach to developing, analysing, and proving correctness of quantum programs. Since we adopt Hehner's theory as the basis for our work, we inherit its advantageous features, such as simplicity, generality, and elegance. Our work extends probabilistic predicative programming in the same fashion that quantum computation extends probabilistic computation. We have provided tools to write quantum as well as classical specifications, develop quantum and classical solutions for them, and analyse various properties of quantum specifications and quantum programs, such as implementability, time and space complexity, and probabilistic error analysis uniformly, all in the same framework. Today quantum computation is a field of interest of scientists of very heterogeneous backgrounds: physicists, computer scientists, information theorists, and cryptographers, among others. Our hope is that the simplicity of our work, as well as our use of a familiar Dirac-like notation, will contribute to bringing together researchers from these different areas.

Traditionally, quantum computation is presented in terms of quantum circuits. We attempt to depart from this convention for the same reason that classical computation is generally not presented in terms of classical circuits. As we develop more complex quantum algorithms, we will need ways to express higher-level concepts with control

structures in a readable fashion. As was mentioned in Chapter 2, a few attempts were already made in this direction. Since the works of Ömer [13] and later Betelli *et al.* [2] do not involve mathematical proof techniques, there is little use in comparing our work to theirs. The quantum programming language, proposed by Selinger [15], is also very different from our approach. It uses density matrices to represent quantum states, which has both advantages and disadvantages. One obvious advantage of his approach is uniform treatment of pure and mixed states. Disadvantages of using density matrices include the inherent difficulty of applying operations to them and problems with visual recognition of quantum states and their properties. A claimed advantage is a lack of growth in distributions as the computation progresses. However, the expressions describing the density matrices still grow in the same fashion. In addition, their use of flow charts to represent quantum programs also decreases readability and ease of writing somewhat, for more complex algorithms. The work of Sanders and Zuliani [14] is more similar to our work. Our claim is that, while being no less expressive than **qGCL**, our approach is simpler and more intuitive. Facility of developing recursive programs, familiarity of notation, and uniformity in time complexity analysis are the evidence.

Research in the immediate future will involve reasoning about non-locality. We will attempt to express quantum teleportation, dense coding, and various games involving entanglement, in a way that makes complexity analysis of these quantum algorithms simple and natural. We can easily express teleportation as refinement of a specification $\phi' = \psi$, for distinct qubits ϕ and ψ , in a well-known fashion. However, we are more interested in the possibilities of simple proofs and analysis of programs involving communication, both via quantum channels and exhibiting the LOCC (local operations, classical communication) paradigm. We also plan to formalise quantum cryptographic protocols, such as BB84 [1], in our framework and provide formal analysis of these protocols. This will naturally lead to formal analysis of distributed quantum algorithms

(e.g. distributed Shor's algorithm for factoring [16]).

Bibliography

- [1] C. H. Bennet and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *IEEE Int. Conf. Computers, Systems and Signal Processing*, pages 175–179, Bangalore, India, 1984.
- [2] S. Bettelli, T. Calarco, and L. Serafini. Toward an architecture for quantum programming. In *The European Physical Journal*, pages 181–200, 2003.
- [3] M. Boyer, G. Brassard, P Høyer, and A. Tapp. Tight bounds on quantum searching. In *Fortschritte der Physik*, pages 493–506, 1998.
- [4] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London*, pages 97–117, 1985.
- [5] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London*, 439:553–558, 1992.
- [6] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219, Philadelphia, PA, May 1996.
- [7] E.C.R. Hehner. *a Practical Theory of Programming*. Springer, New York, second edition, 2004. Available free at www.cs.utoronto.ca/~hehner/aPToP.

- [8] E.C.R. Hehner. Probabilistic predicative programming. In *Mathematics of Program Construction*, Stirling, Scotland, July 2004. www.cs.utoronto.ca/~hehner/PPP.pdf.
- [9] R. Jozsa. Quantum algorithms and the fourier transform. *Proceedings of the Royal Society of London*, pages 323–337, 1998.
- [10] E. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.
- [11] W. K. Nicholson. *Linear Algebra with Applications*. PWS Publishing Company, third edition, 1993.
- [12] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- [13] B. Ömer. Quantum programming in qcl. Master’s thesis, TU Vienna, 2000.
- [14] J. W. Sanders and P. Zuliani. Quantum programming. In *Mathematics of Program Construction*, pages 80–99, 2000.
- [15] P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 2004.
- [16] A. Yimsiriwattana and S. J. Lomonaco Jr. Distributed quantum computing: A distributed shor algorithm. *quant-ph/0403146*, March 2004.

Appendix A

Notations

$ v\rangle$	ket in Dirac notation (6)
z^*	complex conjugate (6)
$\langle v w\rangle$	inner product (6)
\mathcal{H}	Hilbert space (6)
$\langle v $	dual vector (7)
$ w\rangle\langle v $	outer product (7)
$ z $	norm (8)
A^\dagger	adjoint (8)
\otimes	tensor product (9)
H	Hadamard transform (16)
x'	final value of variable x (19)
σ	pre-state (19)
σ'	post-state (19)
ok	(19)

$:=$	assignment (19)
$\times / + - \oplus$	math symbols (20)
$= \neq < > \leq \geq$	(20)
$\wedge \vee \Leftarrow \Rightarrow$ if then else $= \Leftarrow \Rightarrow$	boolean symbols (20)
$P; R$	sequential composition (20)
$x, ..y$	from (including) x to (excluding) y (21)
$A : B$	bunch inclusion (21)
$\lambda x : D \cdot b$	function (22)
$\forall \exists \sum$	quantifiers (22)
t	time variable (23)
s	space variable (23)
∞	infinity (24)
$P; R$	sequential composition (generalised) (27)
if then else	generalised (27)
<i>rand</i>	pseudo-random number generator (27)
$\langle \psi \phi \rangle$	inner product of states (32)
$ \mathbf{x} \rangle$	Dirac-like notation (33)
\otimes^n	tensored n times (34)
I	identity (34)
U	unitary transformation (34)
H	Hadamard transform on states (35)
measure	quantum measurement (36)

U_f	quantum oracle (46)
M	inversion about mean (46)

Appendix B

Refinement Laws

Here we list the rules that are used in proofs in this work.

For state variables x, y, \dots , probability p , booleans a, b , and c , boolean specification P , and probabilistic specifications r and S :

$$ok \equiv x' = x \wedge y' = y \wedge \dots \quad \text{expand } ok$$

$$\text{if } p \text{ then } R \text{ else } R \equiv p \times R + (1 - p) \times S \quad \text{if then else}$$

$$R; S \equiv \sum x'', y'', \dots \cdot R'' \times S'' \quad \text{sequential composition}$$

$$x := e; S \equiv (\text{for } x \text{ substitute } e \text{ in } S) \quad \text{substitution}$$

$$\text{if } b \text{ then } P \text{ else } P \equiv P \quad \text{case idempotence}$$

$$\sum x : D \cdot (x = e) \times S \equiv (\text{for } x \text{ substitute } e \text{ in } S) \quad \text{one point law}$$

$$a \Rightarrow (b \Rightarrow c) \equiv a \wedge b \Rightarrow c \quad \text{portation}$$

$$a \wedge (a \Rightarrow b) \Rightarrow b \quad \text{Modus Ponens}$$

For a state ψ of an n -qubit system:

$$\sum x : 0, \dots, 2^n \cdot |\psi_x|^2 = 1 \quad \text{unitarity of quantum state}$$

$$U_f = \lambda \psi : 0, \dots, 2^n \rightarrow \mathbb{C} \cdot x : 0, \dots, 2^n \cdot (-1)^{f_x} \times \psi x \quad \text{quantum oracle}$$

$$H = \lambda \psi : 0, 1 \rightarrow \mathbb{C} \cdot i : 0, 1 \cdot (\psi 0 + (-1)^i \times \psi 1) / \sqrt{2} \quad \text{Hadamard transform}$$

$$H^{\otimes n} |0\rangle^{\otimes n} = \sum x : 0, \dots, 2^n \cdot |\mathbf{x}\rangle / \sqrt{2^n} \quad \text{Hadamard on } |0\rangle^{\otimes n}$$

$$H^{\otimes n} |\mathbf{x}\rangle = \sum y : 0, \dots, 2^n \cdot (-1)^{\mathbf{x} \cdot \mathbf{y}} \times |\mathbf{y}\rangle / \sqrt{2^n} \quad \text{Hadamard on } |\mathbf{x}\rangle$$

For a collection $M = M_{0, \dots, 2^n}$ of measurement operators, which satisfy the completeness equation $\sum m : 0, \dots, 2^n \cdot M_m^\dagger M_m = I$, an observable $O = \sum m \cdot \lambda_m \times P_m$, where P_m is the projector on the eigenspace of O with eigenvalue λ_m , and an orthonormal basis $B = b_{0, \dots, 2^n}$:

$$\begin{aligned} & \text{measure}_M \psi r && \text{general} \\ & = \langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle \times \left(\psi' = \frac{M_{r'} \psi}{\sqrt{\langle \psi | M_{r'}^\dagger M_{r'} \psi \rangle}} \right) \times (x' = x) \times (y' = y) \dots && \text{measurement} \end{aligned}$$

$$\begin{aligned} & \text{measure}_O \psi r && \text{projective} \\ & = \langle \psi | P_{r'} \psi \rangle \times \left(\psi' = \frac{P_{r'} \psi}{\sqrt{\langle \psi | P_{r'} \psi \rangle}} \right) \times (x' = x) \times (y' = y) \dots && \text{measurement} \end{aligned}$$

$$\begin{aligned} & \text{measure}_B \psi r && \text{measurement} \\ & = |\langle b_{r'} | \psi \rangle|^2 \times (\psi' = b_{r'}) \times (x' = x) \times (y' = y) \dots && \text{in basis } B \end{aligned}$$

$$\begin{aligned} & \text{measure } \psi r && \text{measurement} \\ & = |\psi r'|^2 \times (\psi' = |\mathbf{r}'\rangle) \times (x' = x) \times (y' = y) \dots && \text{in comp. basis} \end{aligned}$$

Appendix C

Proofs omitted from Chapter 6

C.1 A classical solution for Deutsch-Jozsa problem

Our classical solution is:

$$R \equiv b' = \forall i : 0, \dots, 2^{n-1} + 1 \cdot f_i = f_0$$

$$R \Leftarrow y := f_0; x := 1; P$$

$$P \equiv \forall i : 0, \dots, x \cdot f_i = y \implies R$$

$$\equiv \forall i : 0, \dots, x \cdot f_i = y \implies b' = \forall i : 0, \dots, 2^{n-1} + 1 \cdot f_i = f_0$$

$$P \Leftarrow \text{if } y \neq f_x \text{ then } b := \perp$$

$$\text{else if } x = 2^{n-1} \text{ then } b := \top \text{ else } (x := x + 1; P)$$

The proof of the first refinement is as follows:

$$y := f_0; x := 1; P$$

definition of P

$$\equiv y := f_0; x := 1;$$

$$(\forall i : 0, \dots, x \cdot f_i = y \implies b' = \forall i : 0, \dots, 2^{n-1} + 1 \cdot f_i = f_0)$$

substitute

$$\equiv \forall i : 0, \dots, 1 \cdot f_i = f_0 \implies b' = \forall i : 0, \dots, 2^{n-1} + 1 \cdot f_i = f_0$$

simplify

$$\equiv b' = \forall i : 0, \dots, 2^{n-1} + 1 \cdot f_i = f_0$$

definition of R

$\equiv R$

The proof of the second refinement is as follows:

$$\begin{aligned}
& \mathbf{if } y \neq fx \mathbf{ then } b := \perp \\
& \quad \mathbf{else if } x = 2^{n-1} \mathbf{ then } b := \top \mathbf{ else } (x := x + 1; P) \quad \text{definition of } P \\
\equiv & \mathbf{if } y \neq fx \mathbf{ then } b := \perp \\
& \quad \mathbf{else if } x = 2^{n-1} \mathbf{ then } b := \top \\
& \quad \mathbf{else } (x := x + 1; (\forall i : 0, ..x \cdot fi = y \implies \\
& \quad \quad b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0)) \quad \text{substitute} \\
\equiv & \mathbf{if } y \neq fx \mathbf{ then } b := \perp \\
& \quad \mathbf{else if } x = 2^{n-1} \mathbf{ then } b := \top \\
& \quad \mathbf{else } (\forall i : 0, ..x + 1 \cdot fi = y \implies b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \quad \text{assignment} \\
\equiv & \mathbf{if } y \neq fx \mathbf{ then } b' = \perp \wedge x' = x \wedge y' = y \\
& \quad \mathbf{else if } x = 2^{n-1} \mathbf{ then } b' = \top \wedge x' = x \wedge y' = y \\
& \quad \mathbf{else } (\forall i : 0, ..x + 1 \cdot fi = y \implies b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0)
\end{aligned}$$

We now need to prove three refinements:

$$y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \implies P$$

$$y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \implies P$$

$$y = fx \wedge x \neq 2^{n-1} \wedge$$

$$(\forall i : 0, ..x + 1 \cdot fi = y \implies b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \implies P$$

The first proof:

$$y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \implies$$

$$(\forall i : 0, ..x \cdot fi = y \implies b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \quad \text{portation}$$

$$\begin{aligned}
& \implies y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge \forall i : 0, \dots x \cdot fi = y \implies \\
& \quad b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0 \\
& \implies \top
\end{aligned}$$

The second proof:

$$\begin{aligned}
& y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \implies \\
& \quad (\forall i : 0, \dots x \cdot fi = y \implies b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0) \quad \text{portation} \\
& \implies y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge \forall i : 0, \dots x \cdot fi = y \implies \\
& \quad b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0 \\
& \implies \top
\end{aligned}$$

The third proof:

$$\begin{aligned}
& y = fx \wedge x \neq 2^{n-1} \wedge \\
& \quad (\forall i : 0, \dots x + 1 \cdot fi = y \implies b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0) \implies \\
& \quad (\forall i : 0, \dots x \cdot fi = y \implies b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0) \quad \text{portation} \\
& \implies y = fx \wedge x \neq 2^{n-1} \wedge \forall i : 0, \dots x \cdot fi = y \wedge \\
& \quad (\forall i : 0, \dots x + 1 \cdot fi = y \implies b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0) \implies \\
& \quad b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0 \\
& \implies x \neq 2^{n-1} \wedge \forall i : 0, \dots x + 1 \cdot fi = y \wedge \\
& \quad (\forall i : 0, \dots x + 1 \cdot fi = y \implies b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0) \implies \\
& \quad b' = \forall i : 0, \dots 2^{n-1} + 1 \cdot fi = f0 \quad \text{Modus Ponens} \\
& \implies \top
\end{aligned}$$

C.2 Deutsch-Jozsa problem with time restriction

We add a time variable t :

$$R \equiv (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} + 1$$

$$R \Leftarrow t := t + 1; y := f0; x := 1; P$$

$$P \equiv \forall i : 0, ..x \cdot fi = y \Longrightarrow$$

$$(b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1$$

$$P \Leftarrow t := t + 1; \text{if } y \neq fx \text{ then } b := \perp$$

$$\text{else if } x = 2^{n-1} \text{ then } b := \top \text{ else } (x := x + 1; P)$$

The proof of the first refinement is as follows:

$$t := t + 1; y := f0; x := 1;$$

$$(\forall i : 0, ..x \cdot fi = y \Longrightarrow$$

$$(b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{substitute}$$

$$\equiv \forall i : 0, ..1 \cdot (fi = f0) \Longrightarrow$$

$$(b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 1 + 2^{n-1} - 1 + 1 \quad \text{simplify}$$

$$\equiv b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} + 1 \quad \text{definition of } R$$

$$\equiv R$$

To prove the second refinement, we divide the proof into three simpler proofs, as in the previous section:

$$t := t + 1; y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t \Longrightarrow P$$

$$t := t + 1; y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t \Longrightarrow P$$

$$t := t + 1; y = fx \wedge x \neq 2^{n-1} \wedge (x := x + 1; P) \Longrightarrow P$$

The first proof:

$$\begin{aligned}
& t := t + 1; y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& \quad (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{substitute} \\
= & y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t + 1 \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& \quad (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{portation} \\
= & y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t + 1 \wedge \\
& \forall i : 0, ..x \cdot fi = y \implies \\
& \quad (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1 \\
= & \top
\end{aligned}$$

The second proof:

$$\begin{aligned}
& t := t + 1; y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& \quad (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{substitute} \\
= & y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t + 1 \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& \quad (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{portation} \\
= & y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t + 1 \wedge \\
& \forall i : 0, ..x \cdot fi = y \implies \\
& \quad (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1 \\
= & \top
\end{aligned}$$

The third proof:

$$\begin{aligned}
& t := t + 1; y = fx \wedge x \neq 2^{n-1} \wedge \\
& (x := x + 1; (\forall i : 0, ..x \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1)) \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{substitute} \\
= & t := t + 1; y = fx \wedge x \neq 2^{n-1} \wedge \\
& (\forall i : 0, ..x + 1 \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x - 1 + 1) \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{substitute} \\
= & y = fx \wedge x \neq 2^{n-1} \wedge \\
& (\forall i : 0, ..x + 1 \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 1 + 2^{n-1} - x - 1 + 1) \implies \\
& (\forall i : 0, ..x \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1) \quad \text{portation} \\
= & y = fx \wedge x \neq 2^{n-1} \wedge \forall i : 0, ..x \cdot fi = y \wedge \\
& (\forall i : 0, ..x + 1 \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 1 + 2^{n-1} - x) \implies \\
& b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1 \quad \text{absorption} \\
= & x \neq 2^{n-1} \wedge \forall i : 0, ..x + 1 \cdot fi = y \wedge \\
& (\forall i : 0, ..x + 1 \cdot fi = y \implies \\
& (b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 1 + 2^{n-1} - x) \implies \\
& b' = \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0) \wedge t' \leq t + 2^{n-1} - x + 1 \quad \text{Modus Ponens}
\end{aligned}$$

$\equiv \top$

Thus, we have proved an upper bound on t , $t' \leq t + 2^{n-1} + 1$. We now show that the bound is tight, i.e. we obtain $t' = t + 2^{n-1} + 1$ for some input. We prove:

$$R \equiv (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} + 1)$$

$$R \Leftarrow t := t + 1; y := f0; x := 1; P$$

$$P \equiv (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1)$$

$$P \Leftarrow t := t + 1; \text{if } y \neq fx \text{ then } b := \perp$$

$$\text{else if } x = 2^{n-1} \text{ then } b := \top \text{ else } (x := x + 1; P)$$

The proof of the first refinement is as follows:

$$t := t + 1; y := f0; x := 1;$$

$$(\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1) \quad \text{substitute}$$

$$\equiv (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 1 + 2^{n-1} - 1 + 1) \quad \text{math}$$

$$\equiv R$$

For the second refinement we prove:

$$t := t + 1;$$

$$\text{if } y \neq fx \text{ then } b := \perp$$

$$\text{else if } x = 2^{n-1} \text{ then } b := \top$$

$$\text{else } (x := x + 1; \quad \text{substitute}$$

$$(\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1))$$

$$\equiv \text{if } y \neq fx \text{ then } b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t + 1$$

$$\text{else if } x = 2^{n-1} \text{ then } b' = \top \wedge x' = x \wedge y' = y \wedge t' = t + 1$$

$$\text{else } (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 1 + 2^{n-1} - x - 1 + 1)$$

As before, we have three refinements to prove:

$$y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t + 1 \implies P$$

$$y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t + 1 \implies P$$

$$y = fx \wedge x \neq 2^{n-1} \wedge (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 1 + 2^{n-1} - x) \implies P$$

The first proof:

$$\begin{aligned} & y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t + 1 \implies \\ & (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1) \quad \text{portation} \\ \implies & y \neq fx \wedge b' = \perp \wedge x' = x \wedge y' = y \wedge t' = t + 1 \wedge \\ & \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1 \quad \text{simplify} \\ \implies & \perp \implies t' = t + 2^{n-1} - x + 1 \quad \text{simplify} \\ \implies & \top \end{aligned}$$

The second proof:

$$\begin{aligned} & y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t + 1 \implies \\ & (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1) \quad \text{portation} \\ \implies & y = fx \wedge x = 2^{n-1} \wedge b' = \top \wedge x' = x \wedge y' = y \wedge t' = t + 1 \wedge \\ & \forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1 \\ \implies & \top \end{aligned}$$

The third proof is trivial:

$$\begin{aligned} & y = fx \wedge x \neq 2^{n-1} \wedge \\ & (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 1 + 2^{n-1} - x) \implies \\ & (\forall i : 0, ..2^{n-1} + 1 \cdot fi = f0 \implies t' = t + 2^{n-1} - x + 1) \quad \text{weakening} \\ \implies & \top \end{aligned}$$