“Qgol”
A system for simulating quantum computations: Theory, Implementation and Insights

Gregory David Baker

October 31st, 1996

Submitted as partial fulfilment of the requirements

of the award of the Honours degree in Science at Macquarie University.

Department of Computing,
School of Mathematics, Physics, Computing and Electronics,
Macquarie University.

Abstract

I developed a simulator in an attempt to widen research into quantum computing. The design experiences in doing so highlighted many links between the fields of quantum theory and functional programming, and showed an interesting problem in which object-oriented design is counter-productive. The non-imperative design for the simulator made it possible for me to write a formal abstract definition of quantum computation that corresponds more closely to the behaviour of existing quantum systems than previous proposals. Moreover, I was able to generalise this abstract definition to include all known computational universes, and also to provide a framework for creating new ones. (Some examples appear in the appendices.)

In writing the simulator, I have also have created a new experimental test to determine the possibility of using quantum physics for faster-than-light communication — a positive result in this experiment would show it to be impossible. (This is unique — all other tests can only "fail to prove existance" rather than provide a proof of non-existance).

Acknowledgements

I would like to thank the CSIRO Division of Radiophysics (now the Division of Telecommunications and Industrial Physics) for allowing me to spend so much work time on this,

Mike Johnson, for acting as supervisor even while he was away on holidays overseas,

God, for making such an interesting universe,

and my wife Karen, for her patience and understanding beyond what any newly-wed should suffer.

Contents

Chapter 1

Introduction to quantum computing

Quantum computers were first proposed in 1985, and one was built in 1995. They will likely be commonplace in 2030. They use the parallelism implicit in quantum theory to manipulate data more quickly than any classical computer can. I developed a a quantum computing simulator to make research in the area easier.

1.1 Historical background

We may be closer to the first quantum computer than we are to the first microprocessor.

It is projected, based on trends since 1950, that single-atom circuity components will be necessary sometime between 2010 and 2030. As microelectronic components become smaller, the quantum behaviour of individual electrons will start to become significant. As this occurs, it will become less and less practical to view computation as an exercise in manipulating classical states.

Computing is a physical operation — it is a process of taking various inputs, applying physically realisable operations, and producing outputs. Traditionally, computational states have been held as attributes of macroscopic Newtonian objects, and the operations have been energy-consuming Newtonian transformations, but neither of these are essential.

To reduce heat loss, for instance, reversible (adiabatic) circuits have been proposed [1]. The theory of reversible computation is nearly completely understood as it is a minor specialisation of irreversible computation — there are a few simple transformations which can turn an irreversible program into a reversible one without changing its time and space utilisation behaviour.

In quantum computing, operations are performed on *quantum* objects obeying the laws of *quantum* physics.

The study of quantum computation begin in 1985 when the Nobel prize winning physicist Richard Feynman [2] wrote in *Optics News*

> ...such a computer can be built using the laws of quantum mechanics. We are going to write a Hamiltonian, for a system of interacting parts, which will behave in the same way as a large system in serving as a universal computer.

Since then, the corpus of quantum computing papers has grown at an impressive rate. The *Journal of Modern Optics* devoted its 41st volume in 1994 exclusively to quantum computing research - *Physics Letters* did the same in 1993. Other notable discoveries have been Shor's fast factoring alorgorithm [3] which renders nearly all current cryptographic schemes obsolete, Jozsa's oracle [4] which solves some NP problems in polynomial time, and Grover's fast database search algorithm [5], which can find an element in an unsorted list of length N in $\sqrt{N}$ steps (see section 1.3).

On the experimental side, Chuang and Yamamoto [6] successfully built a 2-bit quantum computer in 1995, and performed some simple operations on it. There are problems with their current design preventing it from scaling to larger systems, but there is no shortage of alternate proposals [7] [8] [9].

1.2 Relevant quantum theory

Quantum objects behave very strangely. As Feynman [10] wrote,

> Things on a very small scale behave like nothing you have any direct experience about. They do not behave like waves, they do not behave like particles, they do not behave like clouds, or billiard balls, or weights on springs, or like anything that you have ever seen.

A quantum object can exist in several states at once. Where a classical object can be only “blue” or “red”, a quantum object may be in a *superposition* of multiple states.

We are familiar with statistical distributions (for example, a 50% chance of being “blue”, with a 50% chance of being “red”.) We could imagine that there is a spectrum of values, each of which has some magnitude.

In a quantum system, each of these magnitudes also has a phase angle. Thus, instead of the magnitudes being just real numbers between zero and one as they are with a statistical distribution, they are *complex* numbers with modulus between zero and one.

A particular distribution can be viewed as a vector in a very high-dimensional space. Each distinct value is considered a basis vector, each orthogonal to the other. These magnitudes with phases are called amplitudes (a legacy of the wave-like properties of many quantum systems).

A typical superposition is written like this:

$$0.6i \,|\, \text{Red} \,\rangle + 0.8 \,|\, \text{Blue} \,\rangle$$

(The amplitude associated with the value “Red” is the complex number $0.6i$, and the amplitude associate with the value “Blue” is 0.8.)

When a system is observed (i.e. when it interacts with its environment) the superposition can *collapse* — the universe randomly selects one of the states, with probability equal to the modulus square of the amplitude of that state.

If we were to observe a system in the superposition

$$0.6i \,|\, \text{Red} \,\rangle + 0.8 \,|\, \text{Blue} \,\rangle$$

we would have a probability of 0.36 of observing Red, and a 0.64 probability of observing Blue. If we had observed Red, it would then collapse to

$$1 \,|\, \text{Red} \,\rangle + 0 \,|\, \text{Blue} \,\rangle$$

If we observe it again, we have a 100% chance of observing Red. The system has collapsed from a superposition to a single state.

There are operations that can be done on quantum systems which do not "observe" anything[1] They all have in common some very interesting properties — for instance, they form a group (they have inverses, there is an identity operation, the set is closed, and composition is well-defined). All of them can be represented by unitary operations[2].

Quantum computation, at a theoretical level, is the study of what can be done using these unitary operations together with superposition-collapsing and observations. At a practical level it is the attempt to use the correspondence between this mathematics and the way the micro-world works to build devices with more computational power than would otherwise be possible.

1.3 Fast database searches

Grover [5] showed clearly how much extra computational power a quantum computer can bring — a quantum computer can find a particular element in an unsorted list of length N in $O(\sqrt{N})$ steps to within any given probability of success. Later versions of his

[1]In fact, the situation is far more complicated than I indicate here. Mabuchi and Zoller [11] demonstrated reversible measurements, and Nielsen and Caves [12] generalised unitary evolution and observation even when a system is coupled to its environment.

[2]A unitary operation is one whose complex conjugate transposed is its inverse. The complex conjugate is the operation of negating the imaginary part of a complex value — $3+2i$ has a complex conjugate of $3-2i$. The transpose of a matrix is "the mirror image across the ((top,left),(bottom,right)) diagonal".

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

becomes

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

algorithm [13] can find arbitrary numbers of identical elements.

It uses three $N \times N$ unitary matrix operations, each of which can be built in $O(\log N)$ steps (that is, is "composed" of $O(\log N)$ matrices which are either 4×4 or smaller. I provide an exact definitions of this "composition" – which I call a "connection product" – in section 2.2).

The first is a Fourier Transform, F

$$\frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{N-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{N-2} & \cdots & \omega \end{pmatrix}$$

where $\omega = e^{\frac{2\Pi}{N} i}$. This can be built using the FFT.

The second is a diffusion matrix, D,

$$\begin{pmatrix} -1 + \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & -1 + \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} & -1 + \frac{2}{N} & \cdots & \frac{2}{N} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{2}{N} & \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \end{pmatrix}$$

This is equal to FRF, where R is the very simple

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

The third, T is almost exactly an identity matrix — only one entry is changed. Suppose that s is the element, and that it can be found in the xth position in the unordered list. Then

$$T = \begin{pmatrix} 1 & \cdots & 0 & \\ \vdots & \vdots & \vdots & \\ \cdots & -1 & \cdots & x\text{th row} \\ \vdots & \vdots & \vdots & \\ 0 & \cdots & 1 & \end{pmatrix}$$

.

Note that this matrix can be built without first knowing x (finding x is the aim of the exercise). We only need to know how to recognize that the xth element in the list is s. This is because this matrix is actually an operation performed by some sequence of quantum instructions and the phase shift (the -1 element) corresponds to an `IF`-like function in the program code which will be passed some position y as an argument — it then performs a phase shift if the yth element in the unordered list is s.

We can now describe Grover's algorithm:

1. We create some quantum object, q which is capable of being in any state from 1 up to N, and set it to having a value of 1. In bra-ket notation we would write

$$q_0 = 1\left|1\right\rangle + 0\left|2\right\rangle + 0\left|3\right\rangle + \cdots + 0\left|N\right\rangle$$

.

In matrix form (which makes it clearer to see how the matrix operators are working) we would write

$$q_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

.

2. We apply F to q. q is now in an equal superposition of all numbers between 1 and N.

$$q_1 = \sum_{j=1}^{N} \frac{1}{\sqrt{N}} \left|j\right\rangle$$

(or in matrix form)

$$q_1 = \begin{pmatrix} \frac{1}{\sqrt{N}} \\ \frac{1}{\sqrt{N}} \\ \vdots \\ \frac{1}{\sqrt{N}} \end{pmatrix}$$

.

3. We now apply $(DT)^{\sqrt{N}}$ to the system. We now have[3]

[3]The details of the proof of this are omitted. It can be numerically verified that to have a 50% probability (i.e. an amplitude with magnitude more than $\frac{1}{\sqrt{2}}$) DT needs to be applied suprisingly few

$$q_2 = \left(\sum_{j=1}^{N} d \left| j \right\rangle \right) + (p - d) \left| x \right\rangle$$

where p is an amplitude larger than $\frac{1}{\sqrt{2}}$, and d is some very tiny amplitude. We could write this out in matrix form as :

$$q_2 = \begin{pmatrix} d \\ d \\ \vdots \\ p \\ \vdots \\ d \end{pmatrix}$$

.

Then we observe the system. The probability of observing q having a value x is (as mentioned in section 1.2) equal to the square of the modulus of its amplitude, i.e. something

times.

(Note that the probability of observing q in state x does not increase monotonically with the number of applications of DT. However, it does not suffer wild fluctuations, so in most circumstances it is a reasonable assumption. The discussion here and in [5] is unaffected by this detail.)

Size of list	**Number of applications**
10	1
100	3
1000	11
10000	38
100 000	123
1 000 000	393
10^7	1242
10^8	3962
10^9	12417

above $\frac{1}{2}$.

If we were unfortunate enough to miss observing x, we can try again — it is trivial to determine whether or not the qth element is s or not. The probability of getting a "hit" is very high, so we can rapidly increase our success probability by allowing time for just a few extra iterations through Grover's algorithm.

1.4 Simple quantum cryptography

Einstein, Podolsky and Rosen [14] discovered a suprising situation. Observing a quantum system can have effects that travel faster than light. Suprisingly, there is no information transfer in these effects, since there can be no cause-and-effect mechanism. (If one reference frame sees information travelling from A to B faster than light, then there are achievable reference frames which see information travelling the other way. Thus there is an equal amount of information being transferred in each direction — zero information either way.)

However strange this might be, it does provide a mechanism for what is called quantum cryptography[4].

Consider a system initially :

$$1\,|0,0\rangle + 0\,|1,0\rangle + 0\,|0,1\rangle + 0\,|1,1\rangle$$

where the first and second registers are just single bits. In traditional vector format this is

[4]This scheme was discovered independently by myself and a number of other researchers [15].

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

We now will apply a unitary operation

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

to the first register — this action is equivalent to applying

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ 0 & 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

to the whole system.

We now have a quantum superposition. Writing the new state out in a more traditional vector format, we now have

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \text{– amplitude of } |0,0\rangle \\ \frac{-1}{\sqrt{2}} & \text{– amplitude of } |1,0\rangle \\ 0 & \text{– amplitude of } |0,1\rangle \\ 0 & \text{– amplitude of } |1,1\rangle \end{pmatrix}$$

Writing it out in ket-notation,

$$(\frac{1}{\sqrt{2}})\,|0,0\rangle + (\frac{-1}{\sqrt{2}})\,|1,0\rangle + 0\,|0,1\rangle + 0\,|1,1\rangle$$

It is not often that one can use an `IF` statement in a quantum computation. (In general, `IF` statements are not reversible, and so cannot correspond to unitary operations.) But that is exactly what we are about to do, since in this case the variables mentioned in the decision expression are not the same as those mentioned in the remainder of the statement.

We will apply

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

which can be seen to be (`IF x THEN [TRUE,not y] ELSE [FALSE,y]`), which is not only unitary, but reversible.

Having done this, the system is now $\frac{1}{\sqrt{2}}\,|0,0\rangle + \frac{-1}{\sqrt{2}}\,|1,1\rangle$. The first and second registers must correspond to some physical object — so it is entirely sensible to suggest that Alice take register one, and Bob take register two. Bob catches a spaceship to somewhere a few light-years away, and takes a look at what register two contains. This collapses the superpostion. The probability of observing a state is equal to the square of the modulus of the amplitude of that state. So Bob has a 50% chance of observing a zero, and a 50% chance of observing a one.

Alice, still on earth, now holds a register which is no longer in a superposition. If Bob observed a zero, then Alice's register contains a zero — if Bob observed a one, then Alice's does also. Somehow, Bob has affected her register from many light-years away, seemingly violating Special Relativity which states that no information can be transferred faster than the speed of light. This is the EPR paradox.

Assuming there is no noise in the system, this provides a cryptographic protocol uncrackable even to a spy with unlimited computing power. Suppose Alice wishes to transmit an encrypted bit Z. If she transmits T=(`Z XOR Register1`) to Bob, then Bob can decrypt it with D=(`T XOR Register2`). Since there is exactly a 50% probability that T is one, regardless of Z, no eavesdropper can deduce Z from T. The only way of discovering the decryption pad is to steal one of the registers or else entangle another bit with one from either Alice or Bob. before the observation event.

There are various schemes which can even survive noisy channels [16] and detect eavesdroppers [17].

1.5 Qgol/Cgol

In a paper that will probably be described as the most significant contribution to quantum computing theory, Deutsch [18] wrote in *The Proceedings of the Royal Society of London*:

> Quantum computers raise interesting problems for the design of programming languages, which I shall not go into here. From what I have said, programs exist that would (in order of increasing difficulty) test the Bell inequality, test the linearity of quantum dynamics, and test the Everett interpretation. I leave it to the reader to write them.

So far, no other researcher has addressed these "interesting problems". It was with exactly this in mind that the idea of qgol/cgol came about. Principally it was to be a proof of concept, and to see what were the important issues in quantum computing language design. As a lesser goal, it was hoped that devising a way of expressing quantum computations simply could make a substantial contribution to the field[5], and help make it more comprehensible to non-physicists.

It was also hoped that qgol/cgol could be a simple-to-use language, give good performance on difficult problems, and be available to the largest possible target "audience". I was fortunate enough to have a brief email exchange with Adriano Barenco at Oxford, one of the principal researchers to use simulations. He was particularly interested in a high-performance system, and suggested a number of other additions to the wish-list.

The final version of qgol/cgol, as it became known, includes a combination text-and-graphics editor, with an intuitive menu-and-window based design. It was written entirely in the University of Nijmegen's cross-platform functional language Concurrent Clean, and as a result can be recompiled without changes to run (with native look-and-feel) under MacOS, Linux, OS/2, Sparc-Solaris, SunOS and 32-bit MS-Windows. It is only the present unavailability of compilers for other platforms which hinders any further portability.

The name — qgol/cgol — arose when it became obvious that interesting quantum computations would involve a classical "control system". It was discovered that a system that could only simulate "pure quantum" operations and "ordinary" observations is not a particularly useful system (see section 2.5). Since this control system would be intricately linked with the quantum system underlying it, it was given the working name "cgol", emphasising its classical, rather than quantum nature. Cgol is a simple free-form scripting

[5]Svozil provided Mathematica code in [28] for similar reasons. On big problems, however, it can ran so slowly as to be unusable.

language with a tiny grammar. It is perhaps most noteworthy for having tuples and lists as primitive abstract types.

The final version of qgol — the pure quantum state manipulation language — is a much more interesting language. Firstly, it is a true visual programming language — there is no underlying textual representation. The fundamental symbols of the system are gates (quantum operations) represented by raised blocks, and directed wires (quantum objects). The graphical editor in the qgol/cgol system lets the user select gates, place them on sheets, and wire them together. Some screenshots can be found in Appendix C.

Chapter 2

Universal quantum computers

There are some problems with the usual formalisms for quantum computers. I offer a better, but more conservative, formalism and conclude that there is no such thing as a universal quantum computer.

2.1 Problems with the Deutsch formalism

In an attempt to provide a logical foundation for what was then an ill-conceived notion of a "quantum computer", Deutsch [18] discussed a quantum Turing machine.

> Like a Turing machine, a model quantum computer Q consists of two components, a finite processor and an infinite memory, of which only a finite portion is ever used. The computation proceeds in steps of fixed duration T, and during each step only the processor and a finite part of the memory interact, the rest of the memory remaining static.

> The processor consists of M 2-state observables $(n_i), i \in Z_M$, where Z_M is the set of integers from 0 to M-1. The memory consists of an infinite sequence $(m_i), i \in Z$ of 2-state observables. Corresponding to Turing's "tape position" is another observable x, which has the whole of Z as its spectrum.

It is mathematically sound to construct such a model — it is quite possible to imagine that some universe exists in which it is possible for such an entity to be built. However, it is not entirely clear that it is possible in our universe. For example, how are we to get from one superposition of tape positions to another? Taken in its literal sense, one would need a motor which could move the tape head without interacting with the environment in any other way. From where would it get the energy to first accelerate, and then somehow decelerate it? Into what can it transfer its momentum? Or the worst case — how does one reversibly move an infinitely long tape?

These are not petty engineering issues, as would be the case in a classical system. And the problem doesn't go away by transforming the model — with a traditional classical computer architecture of "registers" and "memory", all we have done is hypothesise the same magical abilities into the bus — somehow it is able to transfer data over arbitrarily large distances in constant time without leaving a trace of doing so in the wider environment.

This is the problem of *information leakage*. Certain states will require the tape head to exert more energy — which ultimately comes from a power station somewhere, which does observe its load. The power station observes part of the program as it runs, and extracting information about its state — this changes the behaviour of the quantum computer. (This is what makes building quantum computers hard — we must devise a system in which we can switch on and off information transfer from the computer to the rest of the world, depending on whether we wish to observe the system or let it run.)

What is going wrong? It is hard to be sure. If it is possible to specify at each step in the computation a finite region D in which operations may be occuring, then it is conceivable to imagine a system for moving the action head around. (If nothing else, it could be decomposed into a finite number of 2-bit operations acting upon adjacent sections of tape. There are 4x4 (2-bit) non-classical unitary transformations which are known to be physically realisable [6] and which can create an interesting collection of operations [19] [20] [21] [22], and so a model based around their use stands a reasonable chance of being valid, and may constitute a rough approximation for how a typical quantum computer might be built.)

If we restrict out study to computations in which there are a bounded, finite number of computational steps, we can be guaranteed that everything occurs within a finite region of space using only a finite number of tape markings. This is a very significant restriction, but no-one has yet provided a workable formalism for any larger set of systems.

This makes possible a formalism for quantum computation in which all computations can be expressed in terms of local interactions. It was also first suggested by Deutsch [23] — the quantum computational network. Deutsch envisaged cyclic networks, which the formalism I provide does not allow. Very uncoincidentallly, this is exactly the paradigm which qgol (the purely quantum operations of qgol/cgol) affords the user.

2.2 Dataflow machines

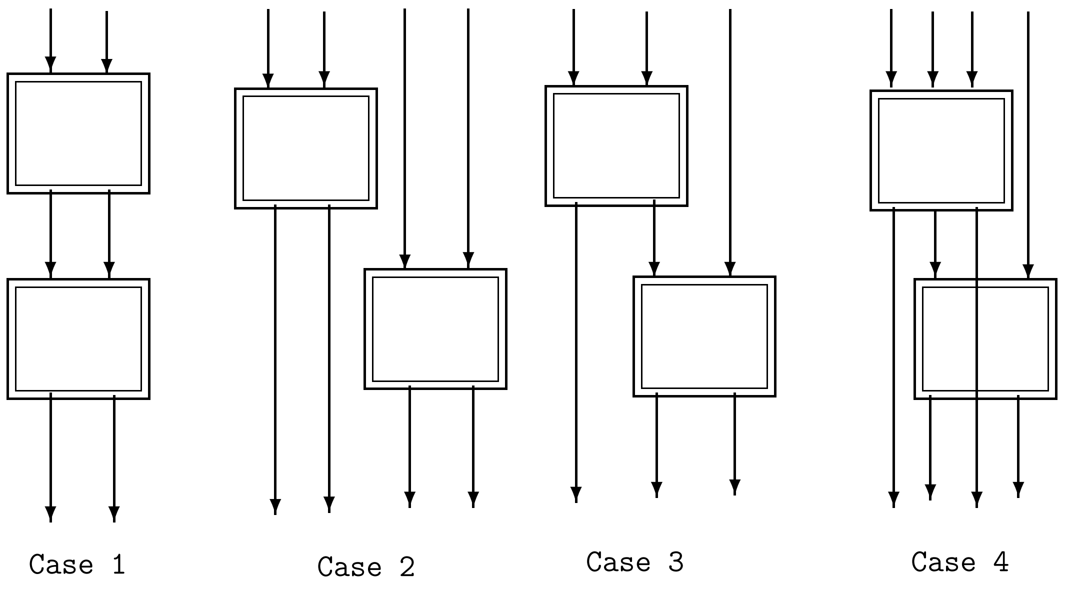
Dataflow machines were conceived at MIT in an attempt to develop a highly parallel data processing machine [24]. The paradigm is of data tokens flowing through a network of computational engines — each node performs an operation on data which is fed to it from input pipes and which it sent out through (possibly multiple) output pipes. Commercially and practically, they have not been a huge success, but they have spurred a number of advances in all kinds of non-imperative programming.

In a dataflow network it is possible to write each node as a matrix operation acting on a high-dimensional vector space. For instance, if each input can carry either a one or a zero, a node with m inputs and n outputs would be represented as a $2^m \times 2^n$ matrix. In a classical dataflow network these matrices have determinant one and have entries that are all zeros or ones. In a reversible system, the matrices are all square and have a single one per row and per column. In a stochastic reversible system, the matrices can have real entries. In a quantum network this is even further relaxed and the matrices can have complex entries, and the only constraint on the determinant is that it have a modulus of one.

It is a non-trivial exercise to combine these matrices into a larger matrix which describes the complete system. In fact, for the general case of a reversible network, it is impossible since there can be arbitrarily deep recursion, and the output can be potentially infinite. There may be no simply describable matrix form for such a calculation.

Fortunately, however, with the assumption of finiteness in the previous section, it is simple to show that a concise form exists.

There are four general cases to consider in combining operations, an example of each of which is given in figure 1.



The first case is the familiar internal product of two matrices:

$$(AB)_{ij} = \sum_k A_{ik}.Bkj$$

The second case is the external product. Given a matrix A of size $m \times n$, and a matrix B of size $s \times t$,

$$(A \otimes B)_{ij} = A_{(i \text{ div } s)(j \text{ div } t)}.B_{(j \text{ mod } s)(i \text{ mod } t)},$$
$$1 \leq i \leq s \times m, 1 \leq j \leq t \times n$$

The third case is a little trickier — two overlapping operations. We create A' from A by using external products with identity matrices on the wires which A misses. Similarly we create B' from B, and define

$$\left(A \overset{\text{overlap}}{\otimes} B \right)_{ij} = (A'B')_{ij}$$

Finally, there is the fully general case of two overlapping operations, possibly with some outputs from the first skipped by the second. We apply permutation matrices P and P' before and after B so that

$$\left(A \overset{\text{overlap,skip}}{\otimes} (PBP')\right)$$

is merely an overlapping operation.

Collectively, I call these matrix products *connection products.* Observe that connection products preserve unitarity.

2.3 An abstract formalism

We now have enough background to make a definition of computation which extends cleanly from classical, irreversible computation through to quantum computing.

Definition *A computation universe U is a 4-tuple (U_s, U_a, U_o, U_v), where U_s is a set of stable states, which is a subset of states U_a, U_o is a collection of operations on U_a which are closed under composition, and U_v is an observation function (a function of two variables, a stream of random numbers and an element of U_a, producing a stream of random numbers and an element of U_s as output).*

In most attempts to formalise the concept of computation, this last part (an observation function) is left out, because in a classical universe it is an identity function. As a result, nobody distinguishes between U_s and U_a. In a quantum universe, it is crucial.

Some examples:

T, the Turing universe $\Rightarrow$

$T_s = T_a =$ some subset of the integers

$T_o =$ computable functions on that subset

$T_v = (i \times i)$

R, reversible classical computers $\Rightarrow$

$R_s = R_a =$ some subset of the integers

$R_o =$ computable functions that have inverses

$R_v = (i \times i)$

S, stochastic reversible computers $\Rightarrow$

$S_s =$ some subset of the integers

$S_a = \{q_s = \{(x, c_s)|x \leftarrow \Re, c_s \leftarrow S_a\}| \sum_{(\_,x)\leftarrow c_s}(x) = 1\}$

$S_o = \{f|f$ is a finite connection product of small probabilistic operations$\}$

$S_v =$ typical observation function

Q, the quantum universe $\Rightarrow$

$Q_s =$ some subset of the integers

$Q_a = \{q_s = \{(x, c_s)|x \leftarrow \Im, c_s \leftarrow Q_s\}| \sum_{(\_,x)\leftarrow c_s} x\overline{x} = 1\}$

$Q_o = \{f|f$ is a finite connection product of 2-bit unitary operations$\}$

$Q_v =$ typical observation function

2.4 Is it worth being interrupted?

It was not until Griffiths' and Nui's paper [25] that anyone found a use for multiple observations occurring at different stages of a computation. Even then, it was able to be done equally simply with a single observation and a slightly transformed composite gate. There is no known algorithm which depends on being observed mid-process for its time-complexity behaviour . In fact, as the next three theorems show, no such algorithm can exist within the formalism of section 2.3.

Definition *An eigenstate is said to be recycled if it suffers an observation, is then (classically) set to a new value, and then suffers quantum operations.*

Theorem *Any system with no recycled elements which is observed at its completion and during its progress is equivalent to a system which is observed once upon completion.*

Proof The gate-based design of qgol strongly suggests this non-existance. All observations are gates without outputs. This offers no lack of generality - if the result of an observation is needed in some part of the rest of the computation, it could be copied instead before-hand. (The hypothetical output-observation of figure 2a can be replaced by the circuit of figure 2b).

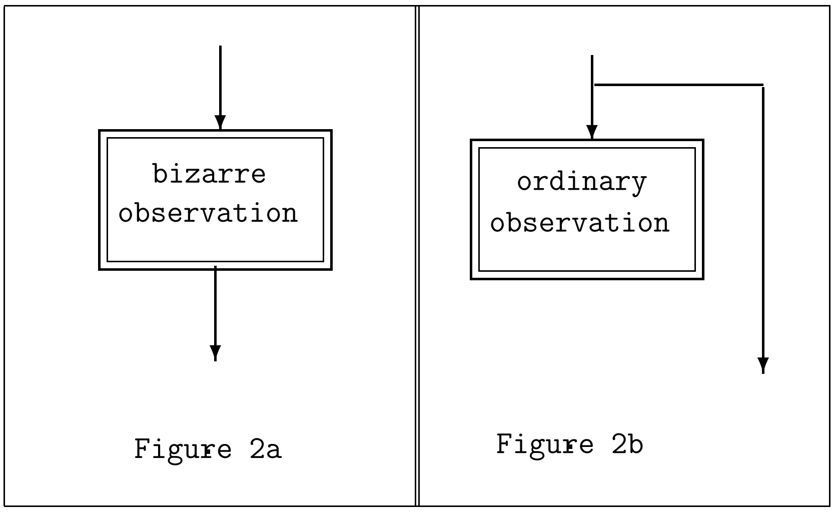


Figure 2a Figure 2b

Now, any operator in a quantum system can have its action delayed until one of its outputs is needed for some other input. (Or it can act early, and leave the results lying around).

Since observations are output-less, and the system does no recycling, their action can be delayed arbitrarily. Thus, if there is an observation occuring upon system completion, any occuring during progress can be postponed to occur at the same time (hence as part of the same observation) as the completion observation. □

Theorem *Removing recycling does not change a polynomial time problem into an exponential problem.*

Proof If the problem can be solved in polynomial time, then only polynomially many recyclings occur. With only a polynomial increase in the number of eigenstates (wires) of the system, observed wires do not need to be reused. This cannot change the problem into an exponential one. □

Similarly, exponential problems cannot be turned into double-exponential problems. Note that the preceeding proofs do not show that recycling cannot improve space behaviour. Nor does it stop a polynomial-time problem becoming constant-time as a result of recycling.

Theorem *Every algorithm which requires observation mid-process can be turned into an algorithm with the same time-complexity behaviour which does not.*

Proof This follows directly from the previous two theorems. If there are recycled elements, then we can use the result of the second theorem to remove them. Since we now have no recycled elements, we can use the first theorem to remove any mid-process observations. □

2.5 There is no universal pure quantum computer

A universal Turing machine is a Turing machine that can take as input an encoding of any program and any input, and evaluate it with only polynomial slowdown. This last constraint is usually ignored since without being counter-productive, there is no known way for a set of operations to be exponentially more difficult when emulated.

By analogy, we can imagine a universal quantum computer. Now the non-exponentiality constraints become more significant.

Definition *A pure quantum computer consists of a non-cyclic network of gates performing unitary operations which gets observed upon completion.*

Theorem *No pure quantum network is universal.*

Proof A pure network will have some finite number of gates G, and some finite number of inputs N. The computation will complete in no more than $T = O(G)$ seconds. It is possible for any N to find a problem which takes more than T seconds to complete. This can be done by taking a function $f : 2^G \rightarrow 2^G$ and a sufficiently large value of n, and attempting to determine whether f^{nT} has some property which is not preserved by f.

Thus, for any network W, problems exist which W cannot solve. $\square$

Chapter 3

The implementation

Quantum systems, suprisingly, cannot easily be described as collections of interacting objects — object-oriented design fails quite spectacularly. On the other hand, there are many constructs in functional programming which have exact analogues in quantum theory. This makes the design of a quantum evaluator very simple. Two examples are discussed: the correspondence between quantum theory's no-cloning theorem and uniqueness types; and between quantum measurement theory and a common constrained parameterisation.

3.1 Quantum simulation is not object oriented

I rewrote the qgol/cgol simulation system several times. In my early versions there were two longstanding problems:

- How is it possible to ensure that operations are unitary?
- How can one compose primitive operations?

The penultimate re-implementation was an attempt to address these issues. I carefully followed an obscure object specification and design model through every stage. Nevertheless, the project failed to meet its goals, and still remains unfinished — the technical hurdles which developed prompted a complete shift in program design and focus.

It is quite unusual to discover a problem domain which is unamenable to object oriented programming. I hope that the following discussion may point towards some of the properties of domains which exhibit this behaviour.

The object-oriented model I envisaged was two-fold: on the one hand, there was the object space of operations. Operations formed a tall heirarchy of inheritance. Addition of two numbers to produce a third was on operation subclassed from the class of Binary operations, in turn subclassed from all Classical operations, which was subclassed from the Operation class. Shor's Fourier transform operation was there as well, derived from SingleVariable, derived from Quantum, derived from Operation.

On the other hand there was a Universe which was a list of unentangled HilbertSpaces. Each HilbertSpace was a set of pairs (amplitude, Eigen). An Eigen was a set of pairs (variable-name, value). A quantum algorithm was a sequence of operation messages which were sent to a universe.

The implementation language did not provide automatic garbage collection. This highlighted an interesting problem — when can one be assured that it is safe to garbage-collect a given non-trivial object? Even though the amplitude associated with some Eigen might currently be zero, and even though there may currently be no variable associated with that value, it might be possible that some fourier-type operation will resurrect it in a new Universe.

This is not an implementation problem — there is physical significance to this question. In a classical system, if I observe a system and discover that it is not in state x, do an operation and discover that it is in state y, then I can be assured that any actions associated with state x never happened. In implementing a simulator for a classical system, I can garbage collect anything to do with state x.

But in simulating a quantum system, this assurance does not exist. A simulator cannot be assured of state x's irrelevance until after the observation of state y, which can't be calculated until all the states of the system in between have been calculated, and that may include state x.

Meanwhile, I was still unable to address any of the three questions raised by the prototype simulator. There was still no way of determining whether or not a given operation was unitary and composition was still unmanageable. I implemented a "probe" command which performed some tests for unitarity, but there was no guarantee that it would discover all non-unitary functions, nor correctly identify those that were.

I realised that I could always defeat the probe when I devised a "hostile function" — with some effort, an imperative language operation could work out whether it was being probed for its unitariness or whether it was being applied to some eigenstate. In C^{++}, as a potentially object-oriented language where each operation was an object with internal and mutable state, this was not only entirely possible, but reasonably simple.

A C^{++} object can access global information. A C^{++} object can even sometimes find out the location of an object which called it by looking at the stack. The design of C^{++} makes possible for an object to act differently when called by the probe instead of a Universe. The probe could not be trusted to give correct answers.

There is no easy way around this — mutable internal state is fundamental. But there is worse still. Consider the alternative face of object-orientation: message-passing. Message-passing is action at a distance, which is a very reasonable concept in the macroscopic world — it happens all the time. But if action at a distance can happen at the quantum level, then there are radical changes neeeding to be made to modern physics. Message passing between Universes is an even greater outrage as it would vindicate all hopeful science-fiction writers around the world. And what would a message passed between a Universe and an Operation mean?

It could be argued that bad design is at fault, but it takes a lot of inspiration to come up with any other way of partitioning the problem. Certainly it is by far the most natural way of describing the problem domain — every design text would encourage the system architect to model the problem this way.

3.2 The Concurrent Clean language

Concurrent Clean [26] is claimed (quite reasonably) to be the fastest-running implementation of any lazily-evaluated language. Parallelism and threading is available for certain versions. It is very portable and provides a high-level mechanism for developing user-interfaces.

It is a true functional language — there are no statements, only referentially transparent expressions. A Concurrent Clean program is a collection of Modula-3 style modules, each of which contains a collection of function definitions. The evaluation of the function "Start" (applied to appropriate arguments) in the main module defines the behaviour of the program. Referential transparency immediately elminated the problem of "hostile functions" mentioned in section 3.1. With no internal state and no chance of causing remote "side-effects" on other parts of the system, there was no danger of action-at-a-distance, nor of the discrepancies between testing and operational behaviour also mentioned in section 3.1. Even the garbage collection problem was solved simply by the nature of the langauge.

It is lazily evaluated: no value is determined (no function is called) until it is strictly necessary. Many modern compilers for imperative languages do "dead code analysis" in which they observe sections of code which will not be executed under any circumstances, or which will have no effect when they do. Lazily evaluated langauges take this one step further — it is in the nature of the evaluation scheme that dead code never gets used. This can lead to impressive speed improvements in certain algorithms, and also allows greater flexibility. For instance, there is no danger in creating an infinite list. If only a finite number of list elements will be used in the course of a program, then only those will be evaluated, and no others. Even a computation that has a kind of "infinite loop" in it can still terminate.

One more small example — this lazy evaluation makes it entirely sensible to read a huge file into memory in some format at the beginning of a program, and then work through whatever data structure has been built. Lazy evaluation (and the garbage collection with which it is intricately tied) ensures that at any one moment only a small portion is actually in memory. Most of the huge file will be sitting on disk, since the need for that portion of the data structure has not yet arisen.

3.3 Uniqueness Types

The above is true of most lazy functional languages, such as Haskell [27]. To all this, Concurrent Clean adds type attributes. Each object has a type, and this type can be annotated to say that it is strict, and/or that it is unique. Strictness opposes the lazy evaluation, and guarantees that an object is evaluated whenever possible, rather than whenever it is necessary. Uniqueness is a much more intriguing concept, with links to diverse fields such as linear logic [33][1]

Garbage collection is easier if the (compile-time) type system can assure the run-time system that certain areas of memory would be free for reuse at appropriate times. This is exactly what uniqueness does — a unique type can appear at most once in the right-hand side of an expression. This means that if a function has a unique type as one of its parameters, then it can know that that memory can be corrupted safely — no other operation will care.

Having done that, a design pattern emerges. It is very simply possible to describe sequential computations.

```
sequential_function ::  *State → *State
sequential_function s1 = final_thing
    where
         s2 = first_operation s1
         s3 = second_operation s2
         final_thing = last_operation s3
```

The type system guarantees that the programmer makes no accidental errors of the form `s3 = second_operation s1` since `s1` is a unique state.

Using this trick, the designers of Clean were able to build a highly sophisticated system for interacting with a windowing system and to do I/O without losing referential transparency.

[1]Linear logic arose from logic programming, which Zapatrin [34] has shown can map neatly onto quantum processors.

3.4 A Functional Specification

Using the full power of the Concurrent Clean language, I was able to provide definitions for all essential functions in four lines — suggesting that Clean provides good abstractions for solving quantum problems.

```
:: Quantum c
```

defines an abstract data type called "Quantum", parameterised by a single type (`c`). This type `c` could be anything — a Char, a list of Integers, an n-tuple of Reals, or even another abstracted type.

It may seem odd that Quantum is an abstract data type — why is it in the same class as Set, List or Dictionary? The idea is that a `Quantum c` is a system which can exist in a superposition of `c`s. For example, a coin can either be `Heads` or `Tails` — a quantum coin can be in any *superposition* of `Heads` or `Tails`.

```
prepare :: c → *(Quantum c)
```

`prepare` is a function which takes an object of any type, and returns a Quantum abstracted from that type. The `*` before the definitions signifies that this is a unique type, as discussed in section 3.3. Furthermore, notice that all references to Quantum types are unique — see section 3.5.

```
observe :: *(Quantum c) Real → c | Eq c
```

Performing an `observe` on a system takes some quantum superposition, and returns a single distinct state. It does so randomly — unfortunately, randomness is inexpressible in a referentially transparent system and it is necessary to explicitly include a random Real for the function's use.

```
entangle ::  (c d → e) *(Quantum c) *(Quantum d) → *(Quantum e)
```

`entangle` is a higher-order function. It takes as its first argument a function. This function needs to be invertible (a fact which cannot be expressed in this type system). Its second and third arguments are the separate quantum systems which are to be entangled.

3.5 The No-Cloning Theorem

The *no-cloning theory* [30] [31] is probably the most import discovery of quantum computation. It states that given a quantum system, one cannot make a copy of it without either destroying the first (which is merely quantum teleportation), or becoming entangled with it.

That is, given a state

$$\sum_i p_i \left| a_i \right\rangle$$

where all a_is are independant, it is possible to create

$$\sum_i p_i \left| a_i, a_i \right\rangle$$

but not to create

$$\left(\sum_i p_i \left| a_i \right\rangle \right) \otimes \left(\sum_i p_i \left| a_i \right\rangle \right)$$

This is equivalent to Heisenburg's Uncertainty Principle. The Uncertainty Principle states that it is impossible to know simultaneously the values of two complementary properties

— for example, the momentum and position of a particle. If it were possible to "clone" a system, then it would be possible to make a measurement of position in system and of momentum in the other, defeating the Uncertainty Principle.

Why this brief digression from implementation issues? Because this shows that the Heisenburg Uncertainty Principle is exactly equivalent to asserting that a state is unique. We may do operations on the non-unique components of a state (i.e. the bits in the a_is), which may include copying, but we cannot copy the complete system. The computer scientist would say that it would break the garbage collection scheme, the linear logician would call it an inference error, and the physicist would cite the no-cloning theorem.

Defining the abstract type `Quantum` to be unique regardless of its abstraction meant remarkably better sanity checking. There is no chance of accidentally forgetting the no-cloning theorem in designing an algorithm — it will cause a compile-time error. It is incredible that in two totally separate fields (quantum theory and functional programming) that the same idea should arise independently, and so neatly. More suprisingly, this weird convergence happens again, several times — see section 3.6 and chapter 4.

3.6 Constrained parameterisation

Constrained parameterisation is an attempt to introduce the object-oriented concept of inheritance into a pure functional framework.

Consider the usual partitioning of an OO problem — we decide, for example, that there is a commonality between Circles, Squares and Triangles — they all may have an attribute of Colour and of Area. By abstracting away this commonality into a base class of Shapes, we attempt to create robust, maintainable software.

When the dreaded moment comes (some years later) to add functionality to support Polygons, we decide that a Polygon IsA Shape, because it is meaningful to talk of the Colour and Area of a Polygon. It isn't, on the other hand, meaningful to talk of the Colour or Area of a BankWithdrawl, and so we decide BankWithdrawl IsA Shape to be false.

Clean and Haskell (and a number of other functional languages) view inheritance merely in terms of this functional specification. Using Concurrent Clean notation :

```
class Shape a
  where
     ColourOf ::  a -> Colour | Shape a
     AreaOf ::  a -> Real | Shape a
```

(The layout rule, i.e. indentation, determines when a class declaration is finished).

The specification states that provided there is a function (called `ColourOf`) which can take an object of type `a` to an object of type `Colour`, and a function called `AreaOf` which can take an `a` to a `Real` number, then `a` is a Shape. Interestingly, this very loose approach makes contravariance much easier — there could easily have been added a "reverse" method

```
CreateFrom ::  [Vertex] -> a | Shape a
```

so that to be a Shape it is necessary to have a method of creation from a list of Vertices.

There was an example of a very common class mentioned in section 3.4, that of `Eq`, the type that demands that there be an equality `==` operation available on a type, and hence also an inequality operation.

In developing the initial functions in section 3.4, there were several puzzling compiler errors with respect to the `observe` function. Curiously, it was complaining that there were not enough constraints on the type of `c`, and that it was far too general.

It was not until I pondered the physical meaning behind "observation" that I realised that it is meaningless to talk of observing states unless they are distinguishable in some way. So I concluded that somehow, it would be necessary to include the constraint that there must be an equality/inequality operation defined on any "observable" type. (This had not been forced at that stage.)

With nothing better on hand to bring this miscreant function into line, this constraint was added mindlessly (hence the | `Eq c` constraint above). The function then immediately compiled without fault. This was one of the defining moments showing the appropriateness of functional programming to this problem domain — it is not often that the programming paradigm actually can correct misspecifications.

Chapter 4

Quantum telegraph

It is conceivable that quantum communication schemes may provide a method for signalling faster than the speed of light. I provide a technologically feasible experiment which could determine whether this is possible, and report on my efforts to perform it.

4.1 Description

Quantum physics may offer a way of sending information faster than the speed of light. This is called "quantum telegraph". In the EPR paradox, for example (see section 1.4), there is certainly a strange faster-than-light effect occuring — the question is whether there are other schemes which might also be able to transfer information.

Kadomtsev [32] in a recent paper in *Physical Review Letters* describes a scheme for communicating over short distances using an observation which is also an entagling of the observed and the observer. So far, no-one has offered any rebuttal. Experimental evidence either way may be some time in development.

The idea behind quantum telegraph is this[1]. Suppose we have a system of the form:

$$p_0 \left| a = a_0, b = b_0 \right\rangle + p_1 \left| a = a_1, b = b_1 \right\rangle + \dots$$

What we would like to do is be able to perform one of (say) two different operations on a and then observe b, and attempt to determine with better than 50% which operation was performed. If we can do this, then faster-than-light communication is possible; a Martian holding b can instantly derive information about actions upon a.

Is this possible? The answer to this question is the opposite as to whether or not quantum telegraph is possible.

Theorem *Quantum telegraph is possible iff there is a data structure partitioning which has the property that the data structures used in modelling two quantum objects can be stored on different processing elements with communication occuring between the processors only when the quantum algorithm performs operations on both at the same time.*

Proof The proof is trivial. If it is possible to parallelise calculations without the need for synchronisation or message-passing, then no information is being transferred between threads. Since each thread is simulating a discrete object, it follows that no information is transferred between discrete objects in the real world using this "quantum magic". Conversely, if quantum telegraph is possible, then it should be impossible to simulate completely a quantum system without some recourse to message passing. □

This is quite an advance as far as the study of faster-than-light communication goes, be-

[1]This describes quantum telegraph in general. Kadomtsev's scheme is far more complicated than this. This complete thesis would double in size if his complete treatment were included here.

cause it turns the assertion “quantum telegraph does not exist” into a positively provable statement — if one can exhibit a suitably parallelised simulator, then quantum telegraph is impossible. Moreover, it is not even a mathematical problem, but merely an engineering one — an experiment to try.

4.2 Experimental attempts

Kodomtsev’s paper asserted that for discrete systems, telegraph is impossible. Qgol, which only simulates discrete systems, then should have been possible to parallelise — doing so would have proven the assertion. However, I failed completely to do so. Admittedly, I do not have much experience as a parallel/distributed programmer, but I was unable to find a suitable way to turn the internal data-structures into cartesian products of separable structures as would be necessary.

It always seemed necessary to be able to distinguish between global states. It was not sufficient to know that with respect to this eigenvector the object had had a value of (say) 3 (ignoring that the other object had a value of 0). It was necessary to be able to distinguish this eigenvector from the one in which the local object had value 3 and the remote one had a value 1.

There may be a rather subtle reason for this inability. Consider the following type definitions, used in the quantum evaluator of the Qgol simulator.

```
::ProcessorState :== LUtable WireID Type

::Processor :== [(ProcessorState,Complex)]

::GateOperation :== ![Type] → [([Type],Complex)]
```

It depends on the pre-existence of the types `LUtable`, `Complex` and `Type`. `LUtable` is a bi-parametric type which is a database of key to value pairs. All functions defined on `LUtable`s happen to be constrained only to work on key types which are orderable (the internal implementation is a binary tree). `Type` is an agglomerate type, principally used to subvert the type system in Clean to make programming easier. In this implementation it can be an integer, a string, a real, a complex, a list of `Type`s, or a tuple of `Type`s. It is effectively opaque to the evaluator.

The definitions above then, create type aliases — a `ProcessorState` is a look-up table, going from register numbers (`WireID`s) to the values contained in those registers. (`ProcessorState` would be defined exactly the same way in a classical simulation.) Each `ProcessorState` can be viewed as a basis vector of the Hilbert space in which the computation takes places. Hence, we define a `Processor` to be a set [list] of `ProcessorState` and amplitude pairs. This could equally easily have been a boxed array of amplitude pairs, since in theory all `ProcessorStates` should be denumerable, but in practice such arrays would be very sparse.

Finally, `GateOperations` are defined to be functions going from some types to some set [list] of `Type` and amplitude pairs.

The actual functions which use these definitions have the following signature:

```
evaluate ::  Processor GateOperation [WireID] [WireID] → Processor
```
// The Ints are input and output registers respectively

```
observe ::  Processor WireID Real → (Type,Processor)
```
// The Int is the register to observe, the Real is random

(Hopefully these definitions should be self-explanatory). Their implementations are easily as dull and predictable as would be expected.

Why this long digression into the implementation? Note that nowhere has it been mentioned that the sets [lists] were finite. In a lazily evaluated language such as Concurrent Clean, this isn't even a necessary precondition for a terminating program.

Suppose the set of `WireID`s had cardinality equal to $\aleph_1$. There would be no type errors and the simulator would re-compile almost unchanged. The sets [lists] do not even need to be countable. If Qgol had been parallelisable, the proof of its parallelisability would have carried through even to uncountably large systems. Kadomtsev's scheme was for systems in which one of the observables had a continuous (i.e. uncountably wide) spectrum, so my inability to parallelise qgol is a statement equivalent to being unable to find a flaw in his design.

Chapter 5

Conclusions

The study of quantum computation is a rapidly growing field, bringing together physics, mathematics and computing. Like all cross-disciplinary fields, there are many suprising synergies.

It is perhaps only slightly suprising that functional programming should provide a useful paradigm for writing programs that model quantum systems. Quantum theory is highly unintuitive and is best described by abstract mathematics — functional programming is at its best in problems well described by mathematics.

Using the same line of argument, it is not suprising to find object-oriented techniques failing to describe quantum systems adequately. Object oriented design demands that systems have a description as a collection of separable subsystems, each of which communicates in fixed ways — this is not the way the quantum universe is (section 3.1). The concept of garbage collection is not necessarily a meaningful concept in a quantum universe, either — it demands that we are able to say “this now exists” or “this no longer exists” at fixed points in time, which makes little sense in a universe of objects which may be in superpositions of existance and non-existance.

What is suprising is that the work in the peripherals of functional programming — constrained parameterisation, uniqueness types and parallel programming — should appear in quantum computing theory as distinguishability (section 3.6), the no-cloning theorem (section 3.5) and quantum telegraph (chapter 4) respectively. Even dataflow architectures provide a model for quantum computation (section 2.2).

While we do know that certain algorithms have a reasonable chance of practical implementation and acceleration on real quantum systems (such as Glover's fast database search (section 1.3), or Shor's factoring algorithm), we do not fully know what form quantum computers will take. I have taken the most conservative description of a quantum computer — one for which almost all the details have been resolved — and turned it into a mathematically well-defined notion (section 2.3). This description makes sense from the smallest known computational universes (Appendix A) to the largest (Appendix B) and encompasses Turing machines and quantum computing (section 2.3).

Accepting a very conservative definition of a quantum computer destroys the possibility of universality (section 2.5) and simplifies the nature of a quantum algorithm (section 2.4) — an algorithm can only be a sequence of unitary operations followed by a complete observation.

But this is quite enough — enough to bring in the EPR paradox, non-locality, and quantum cryptography (section 1.4). There are many more reliable schemes than the one I devised, but few are as easy to implement using practical quantum devices.

Perhaps the most important result of this thesis is the theorem of section 4.1. Faster-than-light communication using quantum effects has mind-boggling implications — it is essential that we know whether or not it is possible. This thesis describes a unique experimental test that can be performed (with no equipment other than an ordinary compiler) which could resolve the issue permanently. My tentative results of section 4.2 suggests that we can not yet rule out the possibility of quantum telegraph.

Bibliography

[1] Bennet, C. H. (1982)
Thermodynamics of Computations - a Review
International Journal of Theoretical Physics, **21**, pp. 219-253

[2] Feynman, R. P. (1985)
Quantum Mechanical Computers
Optics News, February 1985, pp. 11-20

[3] Shor, P. (1994)
Algorithms for Quantum Computation: Discrete Logarithms and Factoring
Proc. 35th Annual Symposium on Foundations of Computer Science.

[4] Jozsa, J. and Deutsch, D. (1992)
Rapid solution of problems by quantum computation
Proceedings of the Royal Society of London **A439**, pp 553-558.

[5] Grover, L. K. (1996)
A fast quantum mechanical algorithm for database search
Proceedings, STOC 1996, pp. 212-219.

[6] Chuang, I.L., Yamamoto, Y. (1995)
A Simple Quantum Computer
Phys. Rev. A **52** p.3489, 1995.

[7] Lloyd, S. (1993)
A Potentially Realizable Quantum Computer
Science, **261**, pp 1569-1571.

[8] Černý, V. (1993)
Quantum Computers and intractable (NP-complete) computing problems
Physical Review A, Volume **48** Number 1, July 1993. pp. 116-119

[9] Barenco, A., Deutsch, D. and Ekert, A. (1995)
Conditional Quantum Dynamics and Logic Gates
Phys. Rev. Lett **74** (1995) 4083-4086

[10] Feynman, R. P. (1965)
The Feynman Lectures on Physics, Vol III

[11] Mabuchi, H., and Zoller, P. (1996)
Phys. Rev. Lett. *76*, 3108.

[12] Nielsen, M. A., and Caves, M. C. (1996)
Reversible quantum operations and their application to teleportation
LANL E-print archive **QUANT-PH-9608001**

[13] Boyer, M., Brassard, G., Hoyer, P., Tapp, A. (1996)
Tight bounds on quantum searching
PhysComp '96.

[14] Einstein, A., Podolsky, P. and Rosen, N. (1933)
Can quantum-mechanical description of physical reality be considered complete?
Phys. Rev., **47**, pp 777-780.

[15] Ardehali, M. (1995)
Quantum bit commitment based on EPR
LANL E-print archive **QUANT-PH-9512026**

[16] Bennett, C. H., Brassard, G., Schumacher, B., Smolin, J., Wootters, W. K. (1996)
Purification of Noisy Entanglement, and Faithful Teleportation via Noisy Channels
Phys. Rev. Lett.**76** 722, 1996.

[17] Chuang, I. L. and Laflamme, R. (1995)
Quantum Error Correction by Coding
LANL e-print archive **QUANT-PH-9511003**

[18] Deutsch, D. (1985)
Quantum Theory, the Church-Turing principle and the universal quantum computer
Proc. Royal Society of London **A400**, pp. 97-117

[19] DiVincenzo, D. P. (1994)
Two-bit gates are universal for quantum computation
Physical Review A (accepted).

[20] Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H. (1994)
Elementary gates for quantum computation
Physical Review A (accepted)

[21] Barenco, A., Deutsch, D., Ekert, A. (1995)
Universality in Quantum Computation
Proc. R. Soc. London **A449**, pp. 669-677.

[22] Smolin, J., DiVincenzo, D.P., (1994)
Results on two-bit gates design for quantum computers
Proceedings of the Workshop on Physics and Computation, PhysComp '94, p. 14.

[23] Deutsch D. (1989)
Quantum Computational Networks
Proc. Royal Society of London **A425**, pp. 73-90

[24] Veen, A. H. (1986)
A Review of Dataflow Architectures
ACM Computing Surveys **18** Number 4, Dec. 1986, pp. 365-396.

[25] Griffiths, R. B. and Nui, C-S. (1995)
Semi-classical Fourier Transform for Quantum Computation
LANL E-print archive **QUANT-PH-9511007**

[26] Concurrent Clean
University of Nijmegen
`http://www.cs.kun.nl/~clean`

[27] *Haskell Report (version 1.2)*
SIGPLAN Notices (Vol. 27, No. 5, May 1992)

[28] Svozil, K. (1994)
Quantum computation and complexity theory
(Course given at the Institut für Informationssysteme, Abteilung für Datenbanken and Expertsysteme, University of Technology, Vienna.)
LANL E-print archive **HEP-TH 9412047**

[29] Deutsch, D., and Josza, R. (1992)
Rapid solution of problems by quantum computation.
Proc R. Soc. London, A.

[30] Wootters, W. K. and Zurek, W. H. (1982)
Nature **299**, 802.

[31] Bu´zek, V. and Hillery, M. (1996)
Quantum copying: Beyond the No-Cloning theorem
LANL E-print archive **QUANT-PH 9607018**

[32] Kadomtsev, B.B. (1995)
Quantum telegraph : is it possible?
(Phys. Rev., **A210**, pp. 371-376).

[33] Girard, J-Y. (1987)
Linear Logic
Theoretical Computer Science **50** (1) (1987) pp. 1-102.

[34] Zapatrin, R. R. (1995)
Logic programming as quantum measurement
International J. Theoretical Phys. **34** (1995) pp. 1813-1821.

[35] Thorne et al. (1990)
Cauchy Problems in Spacetimes with Closed Timelike Curves
Physical Review D **42** p. 1915.

Appendix A

Sub-Turing universes

One of the nice things about the formalism I gave in section 2.3 is that it ties together all known systems for computation. Moreover, it lets us digress and devise others, and suggests a number of questions — one of the most interesting is whether there are more extreme universes than we know of at the moment.

Are there any interesting universes which are smaller than T? At the most trivial level, yes. A monadic universe consisting of one state with the identity operation as its complete set of state modification functions, as well as being its observation function is one definite example. Unfortunately, not much computation can be done in such a universe.

Finding a less tiny example is a little harder, particularly as the concept of "smaller" is not particularly well defined.

Consider a universe in which addition and subtraction are exponentially difficult problems. Such a universe can be devised by allowing a successor/predecessor function, but not providing any means to test for x being greater than y. In this way, it becomes exponentially difficult to write out a state in any particular base.

L $\Rightarrow$

$L_s = L_a =$ some subset of the integers

$L_v = (i \times i)$

$L_o = \{$ `succ, pred , eq` $\}$

This curious little example arose from a couple of days musing — but it is not at all clear whether it is unique in its behaviour, what it may be equivalent to, or even whether there is any other computational universe between L and T.

Appendix B

Universes beyond Q

In most computer architectures there is a measurable positive time delay between input ariving at a gate and output being emitted, but it is quite possible to imagine systems in which this is not the case. We call this computing universe N.

It is equivalent to a conventional van Neumann computer in which the memory bus is given both a time and a location in which to store data. If a piece of data is placed in a memory location in the past, then the computation reverts to that time and continues forward. Thus, to factor 391, one would write a program such as this :

```
t=0:  A(t=0)=2

t=1:  pause

t=2:  IF ((391 mod A) ≠ 0) THEN A(t=1)=A(t=2)+1

t=3:  PRINT ‘‘Factor is’’ A
```

The evaluation of this program would proceed thus:

```
t=0:  A=2

t=1:  pause

t=2:  391 mod 2≠0, so A(t=1)=3

t=1:  pause

t=2:  391 mod 3≠0, so A(t=1)=4

...

t=2:  391 mod 17=0

t=3:  PRINT ‘‘Factor is 17’’
```

In fact, it becomes possible to show that any problem with a solution coming from a finite set can be solved in constant time — constant with regard to the size of the problem, and constant with regard to the size of the set.

This would seem to pose some problems with entropy, since it is possible to produce arbitrary amounts of information (i.e. ordered states) while expending only a finite amount of effort. In actual fact, the energy comes from the operation of sending data backwards in time, a result which has been borne out by Thorne [35] and others when looking at the bizarre behaviour of space time in the region of long rotating gravitiational singularities.

The halting problem also is solvable, unsuprisingly. Since it is possible to compress an infinite number of computations into a constant period of time, it is easy to see whether or not a computation finishes or not. But what about Turing's proof? Can't we create a "nasty" function which examines its own halting behaviour and then does the opposite? Sure — and here is an example of another obscure connection between computing and highly theoretical physics — such a "nasty" function is nothing more than a grandfather paradox.

The grandfather paradox is the most common objection to the whole idea of time travel in general, and it is rather amusing to see how it resolves itself in negative time-delay computing. In abstract terms, one attempts to store in memory in the past the value opposite to what is currently there.

Suppose first that it is unreasonable to observe a system between receiving and transmitting a data packet backwards in time, in the same way that we have considered it "unreasonable" to observe a running quantum system. This is not to say that it is impossible, merely that doing so would destroy the very thing you are wanting to observe. This gives us complete freedom in specifying N_a. N_s should be some subset of the integers, as is traditional for "serious" computing systems. It will be sufficient just to consider $N_s = \{0, 1\}$. N_o is simple — it will be all possible functions over N_a — since any given calculalation can be performed in constant time.

Now for the magic. Let N_a be defined in the same way as S_a:

$$N_a = \left\{ q_s = \{(x, c_s) | x \leftarrow \Re, c_s \leftarrow N_s\} \middle| \sum_{(\_,x)\leftarrow c_s} x = 1 \right\}$$

and let N_v select one state randomly in the same way as S_v or Q_v does.

The operation of grandfather-paradoxing is merely finding a solution for $X = \texttt{NOT}\ X$, which is impossible in an ordinary system. But in this universe it is simple — $X = \left\{(\frac{1}{2}, 0), (\frac{1}{2}, 1)\right\}$. When we observe the system, we discover either a 1 or a 0 — with an equal probability of each. It should be impossible to observe the mechanism for producing this result independant of the result itself.

We cannot say "because we observed a 1, therefore the program should have placed a 0 in memory a few seconds into the past". That would be confusing the roles of G_a and G_s . We cannot complain of being cheated out of seeing the paradox any more than we complain of not being able to observe which slit an electron went through to form a diffraction pattern in a double slit experiment.

Note that the definition above bears a striking resemblance to that of Q, the computational universe in which we live. In fact Q arises from N by a strict extension of the role of x above. Is it possible to add a phase in some natural way to x?

If this is the case, it would mean that quantum theory could arise spontaneously from restricted time-travel, with no *a priori* assumptions about the universe. It would be an amusing footnote in history if a grandly unifying physical theory arose from Computer Science (quantum computing) rather than from Physics!

Appendix C

Screen Shots