Logic and Quantum Computation

Peter Selinger, University of Ottawa

IPM Workshop on Quantum Computing
August 1–5, 2004, Tehran

Introduction

The goal of these lectures is to describe a "structural" theory of quantum computation. Quantum programs can be reasoned about at many different levels. They can be reasoned about *numerically* and *algebraically*, which involves specific calculations with complex numbers and vector space notation. Programs can also be reasoned about *symbolically*, by expressing them in a formal language and formulating axioms and reasoning principles which apply to all programs. We term this latter approach the "structural" approach, and it is the main topic of these lectures.

To allow symbolic reasoning, programs must be expressed in a *formal language*. This language should be structured: large programs are built up from smaller ones by means of syntactic operations, such as sequential or parallel composition or the introduction of loops. A basic principle which should be enjoyed by a structured programming language is the *principle of compositionality*, according to which the behavior of a composite program should be uniquely determined by the behaviors of its parts. The set of programs, under these structural operations, is then subject to some axioms and equations. In many cases, these axioms and equations will be sufficient to reason about a given quantum algorithm in a purely symbolic way.

There are several potential benefits to studying quantum computation in the context of formal languages. First, the use of formal languages allows the creation of a meta-theory: it allows one to quantify over *all possible programs*, rather than reasoning about one program at a time. Also, the uniform construction of large programs from smaller parts gives rise to a principle of reasoning about programs by induction.

Another potential benefit of this approach is the extension of known links between logic and complexity theory. In the classical (i.e., non-quantum) world, there are many such connections. For example, for many complexity classes (such as $\mathbf{P}$, $\mathbf{NP}$, etc.) there exist logics which characterize these complexity classes precisely, in the sense that a function is provably total in the logic if and only if it belongs to the given complexity class. By studying quantum computing in terms of formal languages, one opens the door to extending such results to quantum complexity classes.

The structural approach to quantum computing also offers the possibility to investigate and employ new high-level language features and abstractions. Currently, it seems that each newly discovered quantum algorithm relies on some particular and unique trick. Quantum computations are described and analyzed at the gate-level, which is akin to reasoning in assembly language in the classical world. On the other hand, high-level abstractions are useful in classical programming languages, because they allow algorithms to be expressed in a "natural" style. Features such as data types, recursion, higher-order functions, block structure, object orientation, and so forth, allow the human programmer to achieve complex tasks by organizing them into understandable units. Thus, high-level languages are easy to reason about, while compilers take care of the low-level implementation. One might speculate that a similar benefit can be achieved by introducing abstract features into quantum programming languages. However, little is currently known about the kinds of abstractions that would be useful in this context.

Outline of the lectures

We briefly summarize the content of each of the eight lectures. Suggestions for further reading are given at the end of lectures 4–8.

1. The QRAM model. After a brief review of the basic axioms of quantum computation, we describe the QRAM model of quantum computation due to Knill [4]. In the QRAM model, the operation of a quantum device is controlled by a general-purpose classical computer. Unlike in some other models of quantum computation, unitary operations and measurements can be explicitly interleaved, and the associated classical computations are made explicit.

2. Mixed states and density matrices. We define the notion of a mixed state as a probability distribution on quantum states. It is important to note that a mixed state is not a description of a physical state of a system, but rather of our *knowledge* about a physical state. As such, the concept of a mixed state is a subjective concept. Mixed states also arise in the description of non-closed quantum systems. The density matrix formalism is a convenient notation for mixed states, and we show how the basic operations of quantum computation can be expressed in terms of their actions on density matrices. It follows that two mixed states are indistinguishable from each other (by any series of local operations) if they induce the same density matrix. For (partial) density matrices A and B, we write $A \sqsubseteq B$ if $B - A$ is hermitian positive, and we note that this defines a partial order with least upper bounds of increasing chains. This partial order is important in the description of loops and recursion in quantum programming languages.

3. The quantum flow chart language. We describe a formal language for writing quantum algorithms, the quantum flow chart language of [7]. In this language, every program is seen as a function from inputs to outputs, mapping (in general) mixed states to mixed states. The meaning of a program can be understood in more than one way: (1) by running the program on an abstract machine such as the QRAM model, (2) by annotating the inputs of the program with indeterminate density matrices, then propagating the density matrices downward following a simple set of rules, or (3) inductively, by associating to each program fragment a denotation which is a linear function of mixed states, and computing the denotation of composite programs from the denotations of their parts. Methods (1) and (3) work for all programs, whereas method (2) only works for loop-free code. A semantics in the style of (1) is known as an *operational semantics*, and style (3) is known as *denotational semantics*. All three approaches coincide, in the sense that they calculate the same answers.

4. Completely positive maps and superoperators. We characterize those functions from density matrices to density matrices which can arise as the denotation of a quantum program. These functions are precisely the (partial) *superoperators*. A linear function from matrices to matrices is called *positive* if it maps hermitian positive to hermitian positive matrices. A function F is called *completely positive* if $F \otimes \mathrm{id}_n$ is positive for all n; this means that F is positive as a blockwise operation. Finally, a completely positive map is called a *superoperator* if it is trace-non-increasing. We give a useful characterization of completely positive maps and superoperators in terms of their so-called characteristic matrices, as well as another characterization known as the *Kraus Representation Theorem*. For a more complete exposition of the topics of lectures 1–4, see [7].

5. Basic category theory. Many potential denotational models of quantum computation (for instance, the category $\mathbf{Hilb}$ of finite-dimensional Hilbert spaces and linear maps, the category $\mathbf{CPM}$ of signatures and completely positive maps, and the category $\mathbf{Super}$ of superoperators) possess a symmetric monoidal structure with additional properties. To study the axiomatics satisfied by these models, we introduce some basic category theory. We introduce symmetric monoidal categories and their graphical calculus. We then consider various extensions of symmetric monoidal categories: traced monoidal categories, compact closed categories, strongly compact closed categories, and categories with biproducts. For each of these classes of categories (except the last one), we describe a graphical language. For a general introduction to category theory, see [5].

6. Categorical semantics of quantum protocols. As we have argued in the discussion of the quantum flow chart language and its semantics, reasoning about mixed-state quantum processes takes place in the category $\mathbf{CPM}$ of completely positive maps. Abramsky and Coecke [1] have shown that a large part of this reasoning can be generalized to any strongly compact closed category with biproducts. The category $\mathbf{CPM}$ is one example of such a category, but there are others (for instance, the category $\mathbf{Rel}$ of sets and relations). We describe Abramsky and Coecke's work, and we discuss how the quantum teleportation protocol can be formalized in this framework. The point of working axiomatically is that it allows one to say, for any particular program or protocol, precisely which additional axioms are needed for the protocol to work. Further, the presence of the graphical languages for these categories yields a *graphical formalism* for reasoning about protocols, as opposed to the usual algebraic formalisms such as the bra-and-ket notation. For more details, see [1].

7. The lambda calculus and the proof-as-programs paradigm. As noted in the introduction, in the classical world there are important connections between logic and complexity theory. One source of such connections is the so-called *Curry-Howard isomorphism*, also known as the *proof-as-programs paradigm*. It asserts that a logic can be viewed as a programming language, and a programming language can be viewed as a logic, by identifying formulas with types, and proofs with programs. We present the simply-typed lambda calculus, which was introduced by Church and Curry in the 1930's and 40's. The lambda calculus can be seen as a basic higher-order programming language, but which can also be seen as a system for writing proofs in propositional intuitionistic logic. Thus, the lambda calculus illustrates the Curry-Howard isomorphism in its most basic form. For a much more detailed introduction, see [3] or [6].

8. Linear logic and higher-order quantum computation. In this last lecture, we describe an extension of the Curry-Howard isomorphism to quantum computation. We present a lambda calculus for higher-order quantum computation due to Valiron [8]. An important issue in quantum programming languages is *linearity*, also known as the *no-cloning property*: unlike classical information, quantum information cannot be duplicated. Thus, a quantum programming language must ensure that a qubit is uniquely referenced. As shown by Valiron, the appropriate logic for a higher-order quantum programming language is *linear logic*. Linear logic was introduced in the late 1980's by Girard as a resource-sensitive logic [2], and its proof theory has been extensively studied. Linear logic is characterized by the fact that each assumption made in a proof must be used exactly once (and thus, it cannot be "duplicated"). This is very different from standard classical logic, where an assumption, once made, can be used as many times as needed. In linear logic, duplicable assumptions are marked with the special syntax $!A$. Under the Curry-Howard isomorphism, this gives rise to a type system for a quantum lambda calculus where the type of every value determines whether that value can be duplicated or not. Details about the higher-order quantum lambda calculus can be found in Valiron's forthcoming M.Sc. thesis. A preliminary version of his results has also appeared in [8].

References

[1] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 415–425, 2004. Extended version at arXiv:quant-ph/0402130.

[2] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[3] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, New York, 1989.

[4] E. H. Knill. Conventions for quantum pseudocode. LANL report LAUR-96-2724, 1996.

[5] S. Mac Lane. *Categories for the Working Mathematician*. Springer Graduate Texts in Mathematics 5. 1971.

[6] P. Selinger. Lecture notes on the lambda calculus. Course notes, available from http://quasar.mathstat.uottawa.ca/∼selinger/papers/, 2001.

[7] P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

[8] B. Valiron. Quantum typing. In P. Selinger, editor, *Proceedings of the 2nd International Workshop on Quantum Programming Languages, Turku, Finland*, TUCS General Publication No. 33, pages 163–178, 2004.