CHAPTER

Prospects 8

In the previous chapters, we have systematically studied foundations of quantum programming along the line from superposition-of-data to superposition-of-programs. We saw from the previous chapters that various methodologies and techniques in classical programming can be extended or adapted to program quantum computers. On the other hand, the subject of quantum programming is not a simple and straightforward generalization of its classical counterpart, and we have to deal with many completely new phenomena that arise in the quantum realm but would not arise in classical programming. These problems come from the " weird" nature of quantum systems: for example, no-cloning of quantum data, non-commutativity of observables, coexistence of classical and quantum control flows. They make quantum programming a rich and exciting subject.

This final chapter presents an overview of further developments in and prospects for quantum programming. More explicitly, its aim is two-fold:

- We briefly discuss some important approaches to quantum programming or related issues that are not treated in the main body of this book.
- We point out some topics for future research that are, as I believe, important for further development of the subject, but are not mentioned in the previous chapters.

8.1 QUANTUM PROGRAMS AND QUANTUM MACHINES

Understanding the notions of algorithm, program and computational machine and their relationship was the starting point of computer science. All of these fundamental notions have been generalized into the framework of quantum computation. We already studied quantum circuits in Section 2.2. The studies of quantum programming presented in this book are mainly based on the circuit model of quantum computation. In this section, we consider the relationship between quantum programs and other quantum computational models.

Quantum Programs and Quantum Turing Machines:

Benioff [35] constructed a quantum mechanical model of a Turing machine. His construction is the first quantum mechanical description of a computer, but it

Foundations of Quantum Programming.http://dx.doi.org/10.1016/B978-0-12-802306-8.00008-2

is not a real quantum computer, because the machine may exist in an intrinsically quantum state between computation steps, but at the end of each computation step the tape of the machine always goes back to one of its classical states. The first truly quantum Turing machine was described by Deutsch [69] in 1985. In his machine, the tape is able to exist in quantum states too. A thorough exposition of the quantum Turing machine is given in [38]. Yao [218] showed that a quantum circuit model is equivalent to a quantum Turing machine in the sense that they can simulate each other in polynomial time.

The relationship between programs and Turing machines is well understood; see for example [41,127]. But up to now, not much research on the relationship between quantum programs and quantum Turing machines has been done, except some interesting discussions by Bernstein and Vazirani in [38]. For example, a fundamental issue that is still not properly understood is *programs as data* in quantum computation. It seems that this issue has very different implications in the paradigm of superposition-of-data studied in Part II of this book and in the paradigm of superposition-of-programs studied in Part III.

Quantum Programs and Nonstandard Models of Quantum Computation:

Quantum circuits and quantum Turing machines are quantum generalizations of their classical counterparts. However, several novel models of quantum computation that have no evident classical analogs have been proposed too:

(i) *Adiabatic Quantum Computation*: This model was proposed by Farhi et al. [80]. It is a continuous-time model of quantum computation in which the evolution of the quantum register is governed by a Hamiltonian that varies slowly. The state of the system is prepared at the beginning in the ground state of the initial Hamiltonian. The solution of a computational problem is then encoded in the ground state of the final Hamiltonian. The adiabatic theorem in quantum physics guarantees that the final state of the system will differ from the ground state of the final Hamiltonian by a negligible amount, provided the Hamiltonian of the system evolves slowly enough. Thus, the solution can be obtained with a high probability by measuring the final state. Adiabatic computing was shown by Aharonov et al. [10] to be polynomially equivalent to conventional quantum computing in the circuit model, and it can be seen as a special form of quantum annealing [136].

(ii) *Measurement-Based Quantum Computation*: In the quantum Turing machine and quantum circuits, measurements are mainly used at the end to extract computational outcomes from quantum states. However, Raussendorf and Briegel [183] proposed a one-way quantum computer and Nielsen [175] and Leung [151] introduced teleportation quantum computation, both of them suggesting that measurements can play a much more important role in quantum computation. In a one-way quantum computer, universal computation can be realized by one-qubit measurements together with a special entangled state, called a cluster state, of a large number of qubits. Teleportation quantum computation is based on Gottesman and Chuang's idea of teleporting quantum

gates [104] and allows us to realize universal quantum computation using only projective measurement, quantum memory, and preparation of the $|0\rangle$ state.

(iii) *Topological Quantum Computation*: A crucial challenge in constructing large quantum computers is quantum decoherence. Topological quantum computation was proposed by Kitaev [134] as a model of quantum computation in which a revolutionary strategy is adopted to build significantly more stable quantum computers. This model employs two-dimensional quasi-particles, called anyons, whose world lines form braids, which are used to construct logic gates of quantum computers. The key point is that small perturbations do not change the topological properties of these braids. This makes quantum decoherence simply irrelevant for topological quantum computers.

Only very few papers have been devoted to studies of programming in these nonstandard models of quantum computation. Rieffel et al. [187] developed some techniques for programming quantum annealers. Danos et al. [63] proposed a calculus for formally reasoning about (programs in) measurement-based quantum computation. Compilation for topological quantum computation was considered by Kliuchnikov et al. [138].

Research in this direction will be vital once the physical implementation of some of these models becomes possible. On the other hand, it will be challenging due to the fundamental differences between these nonstandard models and quantum circuits. For example, the mathematical description of topological quantum computation is given in terms of topological quantum field theory, knot theory and lower-dimensional topology. The studies of programming methodology for topological quantum computers (e.g., fixed point semantics of recursive programs) might even bring exciting open problems to these mainstream areas of mathematics.

8.2 IMPLEMENTATION OF QUANTUM PROGRAMMING LANGUAGES

This book is devoted to the exposition of high-level concepts and models of quantum programming. From a practical viewpoint, implementing quantum programming languages and designing quantum compilers are very important. Some research in this direction had already been reported in the early literature; for example, Svore et al. [207] proposed a layered quantum software architecture which is a four-phase design flow mapping a high-level language quantum program onto a quantum device through an intermediate quantum assembly language. Zuliani [242] designed a compiler for the language qGCL in which compilation is realized by algebraic transformation from a qGCL program into a normal form that can be directly executed by a target machine. Nagarajan et al. [176] defined a hybrid classical-quantum computer architecture – the Sequential Quantum Random Access Memory machine (SQRAM) – based on Knill's QRAM, presented a set of templates for quantum assembly code, and developed a compiler for a subset of Selinger's QPL.

A translation between the quantum extension of the **while**-language defined in Chapter 3 and a quantum extension of classical flowchart language was given in [228]. All of these studies are based on the popular circuit model of quantum computation. Nevertheless, as mentioned in the last section, Danos et al. [63] presented an elegant low-level language based on a novel and promising physical implementation model of quantum computation, namely the measurement-based one-way quantum computer.

Recently, quantum compilation including optimization of quantum circuits has been intensively researched. A series of compilation techniques has been developed through the recent projects of languages Quipper [106], LIQUi|> [215], Scaffold [3,126] and QuaFL [150]. In particular, significant progress in synthesis and optimization of quantum circuits has been made in the last few years; see for examples [20,44,45,99,137,188,237,239].

At this moment, the majority of research on quantum language implementation is devoted to quantum circuit optimization. No work except [242] has been reported in the literature on transformations and optimization of high-level language constructs like quantum loops, which is obviously an important issue (see, for example, [13], Chapter 9). In particular, we need to examine whether the techniques successfully applied in classical compiler optimization, e.g., loop fusion and loop interchange, can be used for quantum programs. On the other hand, the analysis techniques developed in Chapter 5 may help in, for example, data-flow analysis and redundancy elimination of quantum programs.

Another important topic for future research is the compilation of quantum programs with quantum control, defined in Chapters 6 and 7.

8.3 FUNCTIONAL QUANTUM PROGRAMMING

This book focuses on imperative quantum programming, but functional quantum programming has been an active research area in the last decade.

The lambda calculus is a formalism of high-order functions and it is a logical basis of some important classical functional programming languages, such as LISP, Scheme, ML and Haskell. The research on functional quantum programming started with an attempt to define a quantum extension of lambda calculus made by Maymin [165] and van Tonder [212]. In a series of papers [196,197,199], Selinger and Valiron systematically developed quantum lambda calculus with well-defined operational semantics, a strong type system and a practical type inference algorithm. As already mentioned in Subsection 1.1.1, a denotational semantics of quantum lambda calculus was recently properly defined by Hasuo and Hoshino [115] and Pagani et al. [178]. The no-cloning property of quantum data makes quantum lambda calculus closely related to linear lambda calculus developed by the linear logic community. Quantum lambda calculus was used by Selinger and Valiron [198] to provide a fully abstract model for the linear fragment of a quantum functional programming language, which is obtained by adding higher-order functions into Selinger's quantum flowchart language QFC [194].

One of the earliest proposals for functional quantum programming was made by Mu and Bird [173]. They introduced a monadic style of quantum programming and coded the Deutsch-Josza algorithm in Haskell. A line of systematic research on functional quantum programming had been pursued by Altenkirch and Grattage [14]. As mentioned in Subsection 1.1.1, they proposed a functional language QML for quantum computation. An implementation of QML in Haskell was presented by Grattage [105] as a compiler. An equational theory for QML was developed by Altenkirch et al. [15]. Notably, QML is the first quantum programming language with quantum control, but it was defined in a way very different from that presented in Chapters 6 and 7. The recent highlight of functional quantum programming is the implementation of languages Quipper [106,107] and LIQUi|> [215]: the former is an embedded language using Haskell as its host language, and the latter is embedded in F#.

The control flow of all quantum lambda calculus defined in the literature as well as functional quantum programming languages (except QML) is classical. So, an interesting topic for further research is to incorporate quantum control (case statement, choice and recursion) introduced in Chapters 6 and 7 into quantum lambda calculus and functional quantum programming.

8.4 CATEGORICAL SEMANTICS OF QUANTUM PROGRAMS

In this book, the semantics of quantum programming languages has been defined in the standard Hilbert space formalism of quantum mechanics. Abramsky and Coecke [5] proposed a category-theoretic axiomatization of quantum mechanics. This novel axiomatization has been successfully used to address a series of problems in quantum foundations and quantum information. In particular, it provides effective methods for high-level description and verification of quantum communication protocols, including teleportation, logic-gate teleportation, and entanglement swapping. Furthermore, a logic of strongly compact closed categories with biproducts in the form of proof-net calculus was developed by Abramsky and Duncan [6] as a categorical quantum logic. It is particularly suitable for high-level reasoning about quantum processes.

Heunen and Jacobs [117] investigated quantum logic from the perspective of categorical logic, and they showed that kernel subobjects in dagger kernel categories precisely capture orthomodular structure. Jacobs [123] introduced the block construct for quantum programming languages in a categorical manner. More recently, he [124] proposed a categorical axiomatization of quantitative logic for quantum systems where quantum measurements are defined in terms of instruments that may have a side-effect on the observed system. He further used it to define a dynamic logic with test operators that is very useful for reasoning about quantum programs and protocols.

Further applications of category-theoretic techniques to quantum programming will certainly be a fruitful direction of research. In particular, it is desirable to

have a categorical characterization of quantum case statement and quantum choice defined in Chapter 6 and quantum recursion based on second quantization defined in Chapter 7.

8.5 FROM CONCURRENT QUANTUM PROGRAMS TO QUANTUM CONCURRENCY

This book only considers sequential quantum programs, but concurrent and distributed quantum computing has already been extensively studied in the literature.

Quantum Process Algebras:

Process algebras are popular formal models of concurrent systems. They provide mathematical tools for the description of interactions, communications and synchronization between processes, and they also provide formal methods for reasoning about behavior equivalence between processes by proving various algebraic laws. Quantum generalization of process algebras has been proposed by several researchers. To provide formal techniques for modelling, analysis and verification of quantum communication protocols, Gay and Nagarajan [93,94] defined the CQP language by adding primitives for measurements and transformations of quantum states and allowing transmission of quantum data in the pi-calculus. To model concurrent quantum computation, Jorrand and Lalire [128,147] defined the QPAlg language by adding primitives expressing unitary transformations and quantum measurements, as well as communications of quantum states, to a classical process algebra, which is similar to CCS. Feng et al. [83,84,229] proposed a model qCCS for concurrent quantum computation, which is a natural quantum extension of classical value-passing CCS and can deal with input and output of quantum states, and unitary transformations and measurements on quantum systems. In particular, the notion of bisimulation between quantum processes was introduced, and their congruence properties were established. Furthermore, symbolic bisimulations and approximate bisimulations (bisimulation metrics) for quantum process algebras are proposed in [81,85,229], respectively. Approximate bisimulation can be used to describe implementation of a quantum process by some (usually finitely many) special quantum gates. One of the most spectacular results in fault-tolerant quantum computation is the threshold theorem, which means that it is possible to efficiently perform an arbitrarily large quantum computation provided the noise in individual quantum gates is below a certain constant. This theorem considers only the case of sequential quantum computation. Its generalization in concurrent quantum computation would be a challenge. The notion of approximate bisimulation provides us with a formal tool for observing robustness of concurrent quantum computation against inaccuracy in the implementation of its elementary gates, and I guess that it can be used to establish a concurrent generalization of the (fault-tolerance) threshold theorem.

Quantum process algebras have already been used in verification of correctness and security of quantum cryptographic protocols, quantum error-correction codes and linear optical quantum computing [24,25,67,68,89,90,98,141,143,219].

Quantum Concurrency:

Research on quantum process algebras was mainly motivated by applications in specification and verification of quantum communication protocols. Actually, concurrency in quantum programming has another special importance: Despite convincing demonstration of quantum computing devices, it is still beyond the ability of the current technology to scale them. So, it was conceived to use the physical resources of two or more small capacity quantum computers to form a large capacity quantum computing system, and various experiments in the physical implementation of distributed quantum computing have been reported in recent years. Concurrency is then an unavoidable issue in programming such distributed quantum computing systems.

Understanding the combined bizarre behavior of concurrent and quantum systems is extremely hard. Almost all existing work in this direction can be appropriately termed as *concurrent quantum programming* but not *quantum concurrent programming*; for example, a *concurrent quantum program* is defined by Yu et al. [238] as a collection of quantum processes together with a kind of *classical* fairness that is used to schedule the execution of the involved processes. However, the behavior of a *quantum concurrent program* is much more complicated. We need to very carefully define its execution model because a series of new problems arises in the quantum setting:

(i) *Interleaving* abstraction has been widely used in the analysis of classical concurrent programs. But entanglement between different quantum processes forces us to restrict its applications. Superposition-of-programs defined in Chapters 6 and 7 adds another dimension of difficulty to this problem; for example, how can the summation operator in quantum process algebras be replaced by a quantum choice?

(ii) Research in physics reveals that certain new *synchronization* mechanisms are possible in the quantum regime; for example, entanglement can be used to overcome certain classical limits in synchronisation [100]. An interesting question is: how to incorporate such new synchronisation mechanisms into quantum concurrent programming?

(iii) We still do not know how to define a notion of fairness that can better embody both the quantum feature of the participating processes and the entanglement between them. A possible way to do this is further generalizing the idea of "quantum coins" and employing some ideas from quantum games [77,168] to control the processes in a quantum concurrent program.

8.6 ENTANGLEMENT IN QUANTUM PROGRAMMING

It has been realized from the very beginning of quantum computing research that entanglement is one of the most important resources that enable a quantum computer to outperform its classical counterpart. However, entanglement in quantum programming has not been discussed at all in the previous chapters. The reason is

that research in this direction is almost nonexistent up to now. Here, we introduce the existing several pieces of work on entanglement that have direct or potential connections to quantum programming.

The role of entanglement in sequential quantum computation has been carefully analyzed by several researchers; see for example [130]. Abstract interpretation techniques were employed by Jorrand and Perdrix [129] and Honda [118] to analyze entanglement evolution in quantum programming written in a language similar to the quantum **while**-language defined in Chapter 3. The compiler [126] of quantum programming language Scaffold [3] facilitates a conservative analysis of entanglement. It was observed in [230] that information leakage can be caused by an entanglement, and thus the Trojan Horse may exploit an entanglement between itself and a user with sensitive information as a covert channel. This presents a challenge to programming language-based information-flow security in quantum computing.

It seems that entanglement is more essential in concurrent and distributed quantum computation than in sequential quantum computation [51,58]. An algebraic language for specifying distributed quantum computing circuits and entanglement resources was defined in [226]. It was also noticed in [83–85,229] that entanglement brings extra difficulties to defining bisimulations preserved by parallel composition in quantum process algebras. Conversely, quantum process algebras provide us with a formal framework for examining the role of entanglement in concurrent quantum computation. In the last section, we already mentioned the possible influence of entanglement on interleaving abstraction in the execution models of quantum concurrent programs.

I expect that research on entanglement in quantum programming, especially in the mode of concurrent and distributed computing, will be fruitful.

8.7 MODEL-CHECKING QUANTUM SYSTEMS

Analysis and verification techniques for quantum programs (with classical control) were studied in Chapters 4 and 5. A natural extension of this line of research is model-checking quantum programs and communication protocols. Actually, several model-checking techniques have been developed in the last decade, not only for quantum programs but also for general quantum systems.

The earlier work was mainly targeted at checking quantum communication protocols. Gay et al. [95] used the probabilistic model-checker PRISM [146] to verify the correctness of several quantum protocols including BB84 [36]. Furthermore, they [97] developed an automatic tool QMC (Quantum Model-Checker). QMC uses the stabilizer formalism [174] for the modelling of systems, and the properties to be checked by QMC are expressed in Baltazar et al. quantum computation tree logic [31].

However, to develop model-checking techniques for general quantum systems, including quantum programs, at least the following two problems must be carefully addressed:

- We need to clearly define a conceptual framework in which we can properly reason about quantum systems, including (1) *formal models of quantum systems*, and (2) *specification languages suited to formalise the properties of quantum systems to be checked*.
- The state spaces of the classical systems that model-checking techniques can be applied to are usually finite or countably infinite. But the state spaces of quantum systems are inherently continuous even when they are finite-dimensional, so we have to explore mathematical structures of the state spaces so that it suffices to examine only a finite number of (or at most, countably infinitely many) representative elements, e.g., those in an orthonormal basis.

The models of quantum systems considered in the current literature on quantum model-checking are either quantum automata or quantum Markov chains and Markov decision processes. The actions in a quantum automaton are described by unitary transformations. Quantum Markov models can be seen as a generalization of quantum automata where actions are depicted by general quantum operations (or super-operators).

Since some key issues in model-checking can be reduced to the reachability problem, reachability analysis of quantum Markov chains presented in Section 5.3 provides a basis of quantum model-checking. The issue of checking linear-time properties of quantum systems was considered in [231], where linear-time properties are defined to be an infinite sequence of sets of atomic propositions modelled by closed subspaces of the state Hilbert spaces. But model-checking more general temporal properties is still totally untouched. Indeed, we do not even know how to properly define a general temporal logic for quantum systems, although this problem has been studied by physicists for quite a long time (see [125] for example).

Another kind of quantum Markov chains were introduced by Gudder [111] and Feng et al. [88], which can be more appropriately termed as super-operator valued Markov chains because they are defined by replacing transition probabilities in a classical Markov chain with super-operators. Feng et al. [88] further noticed that super-operator valued Markov chains are especially convenient for a higher-level description of quantum programs and protocols and developed model-checking techniques for them, where a logic called QCTL (quantum computation tree logic, different from that in [31]) was defined by replacing probabilities in PCTL (probabilistic computation tree logic) with super-operators. A model-checker based on [88] was implemented by Feng, Hahn, Turrini and Zhang [86]. Furthermore, the reachability problem for recursive super-operator valued Markov chains was studied by Feng et al. [87].

8.8 QUANTUM PROGRAMMING APPLIED TO PHYSICS

Of course, the subject of quantum programming has been developed mainly with the purpose of programming future quantum computers. However, some ideas,

methodologies and techniques in quantum programming may also be applied to quantum physics and quantum engineering.

The hypothesis that *the universe is a quantum computer* has been proposed by several leading physicists [155,211]. If you agree with this view, I would like to argue that *God (or nature) is a quantum programmer*. Furthermore, I believe that translating various ideas from programming theory to quantum physics will produce some novel insights. For example, the notion of quantum weakest precondition defined in Subsection 4.1.1 provides with us a new way for backward analysis of physical systems. Recently, Floyd-Hoare logic has been extended to reason about dynamical systems with continuous evolution described by differential equations [46,180]. It is interesting to develop a logic based on this work and quantum Floyd-Hoare logic presented in Section 4.2 that can be used to reason about continuous-time quantum systems governed by the Schrödinger equation.

As pointed out by Dowling and Milburn [71], we are currently in the midst of a second quantum revolution: transition from quantum theory to quantum engineering. The aim of quantum theory is to find fundamental rules that govern the physical systems already existing in the nature. Instead, quantum engineering intends to design and implement new systems (machines, devices, etc.) that did not exist before to accomplish some desirable tasks, based on quantum theory.

Experiences in today's engineering indicate that it is not guaranteed that a human designer completely understands the behaviors of the systems she/he designed, and a bug in her/his design may cause some serious problems and even disasters. So, correctness, safety and reliability of complex engineering systems have been a key issue in various engineering fields. Certainly, it will be even more serious in quantum engineering than in today's engineering, because it is much harder for a system designer to understand the behaviours of quantum systems. The verification and analysis techniques for quantum programs may be adapted to design and implement automatic tools for correctness and safety verification of quantum engineering systems. Moreover, (a continuous-time extension of) model-checking techniques for quantum systems discussed in the last section are obviously useful in quantum engineering. It is sure that this line of research combined with quantum simulation [59,154] will be fruitful.