CHAPTER

1 Introduction

"The challenge [of quantum software engineering] is to rework and extend the whole of classical software engineering into the quantum domain so that programmers can manipulate quantum programs with the same ease and confidence that they manipulate today's classical programs."

excerpt from the 2004 report *Grand Challenges in Computing Research* [120].

Quantum programming is the study of how to program future quantum computers. This subject mainly addresses the following two problems:

- How can programming methodologies and technologies developed for current computers be extended for quantum computers?
- What kinds of new programming methodologies and technologies can effectively exploit the unique power of quantum computing?

Many technologies that have been very successful in traditional programming will be broken when used to program a quantum computer, due to the weird nature of quantum systems (e.g., no cloning of quantum data, entanglement between quantum processes, and non-commutativity of observables which are all assertions about program variables). Even more important and difficult is to discover programming paradigms, models and abstractions that can properly exploit the unique power of quantum computing – *quantum parallelism* – but cannot be sourced from knowledge of traditional programming.

1.1 BRIEF HISTORY OF QUANTUM PROGRAMMING RESEARCH

The earliest proposal for quantum programming was made by Knill in 1996 [139]. He introduced the Quantum Random Access Machine (QRAM) model and proposed a set of conventions for writing quantum pseudo-code. In the 20 years since then, research on quantum programming has been continuously conducted, mainly in the following directions.

Foundations of Quantum Programming. http://dx.doi.org/10.1016/B978-0-12-802306-8.00001-X

1.1.1 DESIGN OF QUANTUM PROGRAMMING LANGUAGES

Early research on quantum programming focused on the design of quantum programming languages. Several high-level quantum programming languages have been defined in the later 1990s and early 2000s; for example, the first quantum programming language, QCL, was designed by Ömer [177], who also implemented a simulator for this language. A quantum programming language in the style of Dijkstra's guarded-command language, qGCL, was proposed by Sanders and Zuliani [191,241]. A quantum extension of C++ was proposed by Bettelli et al. [39], and implemented in the form of a C++ library. The first quantum language of the functional programming paradigm, QPL, was defined by Selinger [194] based on the idea of classical control and quantum data. A quantum functional programming language QML with quantum control flows was introduced by Altenkirch and Grattage [14]. Tafliovich and Hehner [208,209] defined a quantum extension of a predicative programming language that supports the program development technique in which each programming step is proven correct when it is made.

Recently, two general-purpose, scalable quantum programming languages, Quipper and Scaffold, with compilers, were developed by Green et al. [106] and Abhari et al. [3], respectively. A domain-specific quantum programming language, QuaFL, was developed by Lapets et al. [150]. A quantum software architecture LIQUi|>, together with a quantum programming language embedded in F#, was designed and implemented by Wecker and Svore [215].

1.1.2 SEMANTICS OF QUANTUM PROGRAMMING LANGUAGES

Formal semantics of a programming language give a rigorous mathematical description of the meaning of this language, to enable a precise and deep understanding of the essence of the language beneath its syntax. The operational or denotational semantics of some quantum programming languages were already provided when they were defined; for example, qGCL, QPL and QML.

Two approaches to predicate transformer semantics of quantum programs have been proposed. The first was adopted by Sanders and Zuliani [191] in designing qGCL, where quantum computation is reduced to probabilistic computation by the observation (measurement) procedure, and thus predicate transformer semantics developed for probabilistic programs can be applied to quantum programs. The second was introduced by D'Hondt and Panangaden [70], where a quantum predicate is defined to be a physical observable represented by a Hermitian operator with eigenvalues within the unit interval. Quantum predicate transformer semantics was further developed in [225] with a special class of quantum predicates, namely projection operators. Focusing on projective predicates allows the use of rich mathematical methods developed in Birkhoff-von Neumann quantum logic [42] to establish various healthiness conditions of quantum programs.

Semantic techniques for quantum computation have also been investigated in some abstract, language-independent ways. Abramsky and Coeck [5] proposed a

category-theoretic formulation of the basic postulates of quantum mechanics, which can be used to give an elegant description of quantum programs and communication protocols such as teleportation.

Recent progress includes: Hasuo and Hoshino [115] found a semantic model of a functional quantum programming language with recursion via Girard's Geometry of Interaction [101], categorically formulated by Abramsky, Haghverdi and Scott [7]. Pagani, Selinger and Valiron [178] discovered a denotational semantics for a functional quantum programming language with recursion and an infinite data type using constructions from quantitative semantics of linear logic. Jacobs [123] proposed a categorical axiomatization of block constructs in quantum programming. Staton [206] presented an algebraic semantic framework for equational reasoning about quantum programs.

1.1.3 VERIFICATION AND ANALYSIS OF QUANTUM PROGRAMS

Human intuition is much better adapted to the classical world than the quantum world. This fact implies that programmers will commit many more faults in designing programs for quantum computers than in programming classical computers. Thus, it is crucial to develop verification techniques for quantum programs. Baltag and Smets [30] presented a dynamic logic formalism of information flows in quantum systems. Brunet and Jorrand [50] introduced a way of applying Birkhoff-von Neumann quantum logic in reasoning about quantum programs. Chadha, Mateus and Sernadas [52] proposed a proof system of the Floyd-Hoare style for reasoning about imperative quantum programs in which only bounded iterations are allowed. Some useful proof rules for reasoning about quantum programs were proposed by Feng et al. [82] for purely quantum programs. A Floyd-Hoare logic for both partial and total correctness of quantum programs with (relative) completeness was developed in [221].

Program analysis techniques are very useful in the implementation and optimization of programs. Termination analysis of quantum programs was initiated in [227], where a measurement-based quantum loop with a unitary transformation as the loop body was considered. Termination of a more general quantum loop with a quantum operation as the loop body was studied in [234] using the semantic model of quantum Markov chains. It was also shown in [234] that the Sharir-Pnueli-Hart method for proving properties of probabilistic programs [202] can be elegantly generalized to quantum programs by exploiting the Schrödinger-Heisenberg duality between quantum states and observables. This line of research has been continued in [152,153,235,236,238] where termination of nondeterministic and concurrent quantum programs was investigated based on reachability analysis of quantum Markov decision processes. Another line of research in quantum program analysis was initiated by Jorrand and Perdrix [129] who showed how abstract interpretation techniques can be used in quantum programs.

1.2 APPROACHES TO QUANTUM PROGRAMMING

Naturally, research on quantum programming started from extending traditional programming models, methodologies and technologies into the quantum realm. As stated in Section 1.1, both imperative and functional programming have been generalized for quantum computing, and various semantic models, verification and analysis techniques for classical programs have also been adapted to quantum programming.

The ultimate goal of quantum programming is to fully exploit the power of quantum computers. It has been well understood that the advantage of quantum computers over current computers comes from quantum parallelism – *superposition of quantum states* – and its derivatives such as entanglement. So, a key issue in quantum programming is how to incorporate quantum parallelism into traditional programming models. In my opinion, this issue can be properly addressed in the following two paradigms of superposition.

1.2.1 SUPERPOSITION-OF-DATA – QUANTUM PROGRAMS WITH CLASSICAL CONTROL

The main idea of the *superposition-of-data paradigm* is to introduce new program constructs needed to manipulate quantum data, e.g., unitary transformations, quantum measurements. However, the control flows of quantum programs in such a paradigm are similar to those of classical programs. For example, in classical programming, a basic program construct that can be used to define the control flow of a program is the conditional (**if** . . . **then** . . . **else** . . . **fi**) statement, or more generally the case statement:

$$\textbf{if } (\square i \cdot G_i \rightarrow P_i) \textbf{ fi} \tag{1.1}$$

where for each i, the subprogram P_i is guarded by the Boolean expression G_i, and P_i will be executed only when G_i is true. A natural quantum extension of statement (1.1) is the measurement-based case statement:

$$\textbf{if } (\square i \cdot M[q] = m_i \rightarrow P_i) \textbf{ fi} \tag{1.2}$$

where q is a quantum variable and M a measurement performed on q with possible outcomes $m_1, \ldots, m_n$, and for each i, P_i is a (quantum) subprogram. This statement selects a command according to the outcome of measurement M: if the outcome is m_i, then the corresponding command P_i will be executed. It can be appropriately called *classical case statement in quantum programming* because the selection of commands in it is based on classical information – the outcomes of a quantum measurement. Then other language mechanisms used to specify the control flow of quantum programs, e.g., loop and recursion, can be defined based on this case statement.

The programming paradigm defined here is called the superposition-of-data paradigm because the data input to and computed by these programs are quantum

data – superposition of data, but programs themselves are not allowed to be superposed. This paradigm can be even more clearly characterized by Selinger's slogan "quantum data, classical control" [194] because the data flows of the programs are quantum, but their control flows are still classical.

The majority of existing research on quantum programming has been carried out in the superposition-of-data paradigm, dealing with quantum programs with classical control.

1.2.2 SUPERPOSITION-OF-PROGRAMS – QUANTUM PROGRAMS WITH QUANTUM CONTROL

Inspired by the construction of quantum walks [9,19], it was observed in [232,233] that there is a fundamentally different way to define a case statement in quantum programming – *quantum case statement* governed by a quantum "coin":

$$\mathbf{qif}[c]\ (\square i \cdot |i\rangle \rightarrow P_i)\ \mathbf{fiq} \tag{1.3}$$

where $\{|i\rangle\}$ is an orthonormal basis of the state Hilbert space of an *external* "coin" system c, and the selection of subprograms P_i's is made according to the basis states $|i\rangle$ of the "coin" space that *can be superposed* and thus is quantum information rather than classical information. Furthermore, we can define a *quantum choice*:

$$[C]\left(\bigoplus_i |i\rangle \rightarrow P_i\right) \triangleq C[c]; \mathbf{qif}[c]\ (\square i \cdot |i\rangle \rightarrow P_i)\ \mathbf{fiq} \tag{1.4}$$

Intuitively, quantum choice (1.4) runs a "coin-tossing" program C to create a superposition of the execution paths of subprograms $P_1, \ldots, P_n$, followed by a quantum case statement. During the execution of the quantum case statement, each P_i is running along its own path within the whole superposition of execution paths of $P_1, \ldots, P_n$. Based on this kind of quantum case statement and quantum choice, some new quantum program constructs such as quantum recursion can be defined.

This approach to quantum programming can be termed the *superposition-of-programs paradigm*. It is clear from the definitions of quantum case statement and quantum choice that the control flow of a quantum program in the superposition-of-program paradigm is inherently quantum. So, this paradigm can also be characterized by the slogan "quantum data, quantum control"[1].

I have to admit that this paradigm is still in a very early stage of development, and a series of fundamental problems are not well understood. On the other hand, I believe that it introduces a new way of thinking about quantum programming that can help a programmer to further exploit the unique power of quantum computing.

[1]The slogan "quantum data, quantum control" was used in [14] and in a series of its continuations to describe a class of quantum programs for which the design idea is very different from that introduced here.

1.3 STRUCTURE OF THE BOOK

This book is a systematic exposition of the theoretical foundations of quantum programming, organized along the line *from superposition-of-data to superposition-of-programs*. The book focuses on imperative quantum programming, but most ideas and techniques introduced in this book can also be generalized to functional quantum programming.

The book is divided into four parts:

- Part I consists of this introductory chapter and Chapter 2, Preliminaries. The prerequisites for reading this book are knowledge of quantum mechanics and quantum computation and reasonable familiarity with the theory of programming languages. All prerequisites for quantum mechanics and quantum computation are provided in Chapter 2. For theory of programming languages, I suggest the reader consult the standard textbooks, e.g., [21,158,162,200].
- Part II studies quantum programs with classical control in the superposition-of-data paradigm. This part contains three chapters. Chapter 3 carefully introduces the syntax and the operational and denotational semantics of quantum programs with classical control (case statement, loop and recursion). Chapter 4 presents a logical foundation for reasoning about correctness of quantum programs with classical control. Chapter 5 develops a series of mathematical tools and algorithmic techniques for analysis of quantum programs with classical control.
- Part III studies quantum programs with quantum control in the superposition-of-programs paradigm. This part consists of two chapters. Chapter 6 defines quantum case statement and quantum choice and their semantics, and establishes a set of algebraic laws for reasoning about quantum programs with the constructs of quantum case statement and quantum choice. Chapter 7 illustrates how recursion with quantum control can be naturally defined using quantum case statement and quantum choice. It further defines the semantics of this kind of quantum recursion with second quantization – a mathematical framework for dealing with quantum systems where the number of particles may vary.
- Part IV consists of a single chapter designed to give a brief introduction to several important topics from quantum programming that have been omitted in the main body of the book and to point out several directions for future research.

The dependencies of chapters are shown in Figure 1.1.

- ***Reading the Book*:** From Figure 1.1, we can see that the book is designed to be read along the following three paths:
  - *Path 1*: Chapter 2 → Chapter 3 → Chapter 4. This path is for the reader who is mainly interested in logic for quantum programs.
  - *Path 2*: Chapter 2 → Chapter 3 → Chapter 5. This path is for the reader who is interested in analysis of quantum programs.
  - *Path 3*: Chapter 2 → Chapter 3 → Chapter 6 → Chapter 7. This path is for the reader who would like to learn the basic quantum program constructs in

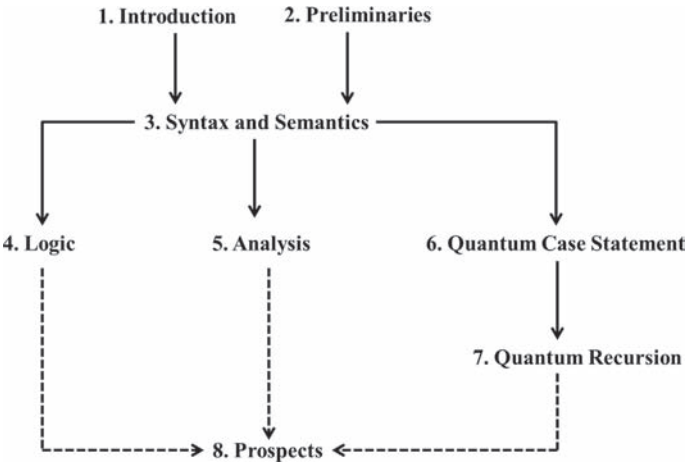

FIGURE 1.1

Dependencies of chapters.

not only the superposition-of-data but also the superposition-of-programs paradigms.

Of course, only a thorough reading from the beginning to the end of the book can give the reader a full picture of the subject of quantum programming.

- ***Teaching from the Book*:** A short course on the basics of quantum programming can be taught based on Chapters 2 and 3. Furthermore, Parts I and II of this book can be used for a one- or two-semester advanced undergraduate or graduate course. A one-semester course can cover one of the first two paths described previously. Since the theory of quantum programming with quantum control (in the superposition-of-programs paradigm) is still at an early stage of its development, it is better to use Chapters 6 and 7 as discussion materials for a series of seminars rather than for a course.
- ***Exercises*:** The proofs of some lemmas and propositions are left as exercises. They are usually not difficult. The reader is encouraged to try all of them in order to solidify understanding of the related materials.
- ***Research Problems*:** A couple of problems for future research are proposed at the end of each chapter in Parts II and III.
- ***Bibliographic Notes*:** The last sections of Chapters 2 through 7 are bibliographic notes, where citations and references are given, and recommendations for further reading are provided. The complete bibliography is provided in a separate section at the end of the book, containing the alphabetized list of both cited references and those recommended for further reading.
- ***Errors*:** I would appreciate receiving any comments and suggestions about this book. In particular, if you find any errors in the book, please email them to: Mingsheng.Ying@uts.edu.au or yingmsh@tsinghua.edu.cn.