Review

Chinese Science Bulletin

SPECIAL TOPIC
Quantum Information

June 2012 Vol. 57 No. 16: 1903–1909
doi: 10.1007/s11434-012-5147-6

Quantum programming: From theories to implementations

YING MingSheng[1,2*], FENG Yuan[1,2], DUAN RunYao[1,2], LI YangJia[1,2] & YU NengKun[1,2]

[1] *Center for Quantum Computation and Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology, Sydney, NSW 2007, Australia;*
[2] *State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*

Received November 14, 2011; accepted February 9, 2012

This paper surveys the new field of programming methodology and techniques for future quantum computers, including design of sequential and concurrent quantum programming languages, their semantics and implementations. Several verification methods for quantum programs and communication protocols are also reviewed. The potential applications of programming techniques and related formal methods in quantum engineering are pointed out.

quantum computation, programming languages, semantics, verification, engineered quantum systems

Citation: Ying M S, Feng Y, Duan R Y, et al. Quantum programming: From theories to implementations. Chin Sci Bull, 2012, 57: 1903–1909, doi: 10.1007/s11434-012-5147-6

Even though quantum hardware is still in its infancy, people widely believe that building a large-scale and functional quantum computer is merely a matter of time and concentrated effort. As the techniques of quantum devices progress, architectural studies will become critical for designing and implementing bigger quantum computing systems. Indeed, a research group from top US universities, including MIT and UC Berkeley, has conducted their research on quantum computing architecture, with support from the DARPA Quantum Information Program [1]. On the other hand, once quantum computers come into being, quantum software will play the key role in exploiting the power of quantum computers. However, today's software development methodologies and techniques are purely classical, and they are not suited to quantum computers. In the last 15 years, researchers began to realize the importance of quantum software. In fact, quantum software engineering was listed as a grand challenge in UK Grand Challenges in Computing Research, and the goal of research on quantum software is explained clearly by the following excerpt from the Grand Challenges Report [2]: The challenge 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.

In US, EU and Canada there are already several research teams devoting to theoretical studies of quantum software, in particular quantum programming. More recently in 2010, IARPA in US initiated a Quantum Computer Science Program, and one of its major research issues is high-level quantum programming languages and quantum programming environments, including quantum compilers. Two excellent survey papers of quantum programming are [3,4], two newer surveys are [5] by one of the authors and [6], and [7,8] also contain a brief survey of quantum programming. This paper will further review the field of quantum programming, with emphasis on the authors' recent work conducted at Tsinghua (China) and UTS (Australia). Another quantum programming research group in China is led by Profs. Jiafu Xu and Fangmin Song at Nanjing University [9].

*Corresponding author (email: Mingsheng.Ying@uts.edu.au; yingmsh@tsinghua.edu.cn)

1 Quantum programming languages

Currently, quantum algorithms are expressed mainly in the very low level of quantum circuits. In the history of classical computation, however, it was realized long time ago that programming languages provide a technique that allows us to think about a problem that we intend to solve in a high-level, conceptual way, rather than the details of implementation. In order to offer a similar technique in quantum computation, people began to study the principles, design and semantics of quantum programming languages and to understand quantum computation at a higher-level [3, 4, 10]. The earliest proposal for quantum programming language was made by Knill [11], who introduced the Quantum Random Access Machine (QRAM) model for quantum computing and proposed a set of conventions for writing quantum pseudocode. Since then, several high-level quantum programming languages have been defined. The first quantum programming language, QCL, was designed by Ömer [12]; he 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 [13]. A quantum extension of C++ was proposed by Bettelli et al. [14], and implemented in the form of a C++ library. The first and now influential quantum language of the functional programming paradigm, QPL, was defined by Selinger [15] based on the idea of classical control and quantum data. A quantum functional programming language QML with quantum control was introduced by Altenkirch and Grattage [16]. Tafliovich and Hehner [17, 18] defined a quantum extension of predicative programming language that supports the program development technique in which each programming step is proven correct when it is made.

2 Understanding quantum loop and recursive programs

It is crucial for the design and implementation of quantum programming languages to thoroughly understand computational mechanism of various complex quantum program constructs. Loop and recursion are powerful program constructs in classical computation, and understanding their behaviors and exploiting their powers are one of the major challenges in classical programming methodology. In quantum computation, looping technique has also attracted a few researchers' attention. For example, Bernstein and Vazirani [19] introduced some programming primitives including looping in the context of quantum Turing machines; some high-level control features such as loop and recursion are provided in Selinger's QPL [15]. However, the full power of quantum loop programs is still to be exploited. Recently the authors initiated a systematic study of quantum loop programs [20]. They introduced a general scheme of quantum loop programs. The computational process of a quantum loop was carefully described, and then the essential difference between quantum loops and classical loops was analyzed, which mainly comes from quantum measurements in the loop guards. Furthermore, they introduced the notions of termination and almost termination (according to termination probability) of a quantum loop. In particular, in a fixed finite-dimensional state space, they found a necessary and sufficient condition under which a quantum loop program terminates on a given (mixed) input state by employing Jordan normal form of complex matrices. A similar condition is also given for almost termination. Moreover, they proved that a small disturbance either on the unitary transformation in the loop body or on the measurement in the loop guard can make any quantum loop to (almost) terminate, provided that some obvious dimension restriction is satisfied.

Termination analysis of quantum loops was continued in [21], where nondeterministic quantum programs are considered. A nondeterministic quantum program is modelled by a quantum Markov decision process, which is essentially a finite set of quantum loops with the same loop guard. A characterization of reachable space and diverging states of a nondeterministic quantum program is presented. A zero-one law for termination probability of the states in the reachable space is proved, and an algorithm is found for checking termination of nondeterministic quantum programs within a fixed finite-dimensional state space. Furthermore, a class of concurrent quantum programs were introduced in our unpublished paper[1)], which are also a set of quantum loops with the same loop guard, but in which these loops are executed according to schedules satisfying certain fairness conditions. This notion of concurrent quantum program is a natural quantum generalisation of probabilistic concurrent program defined by Hart et al. [22].

For further studies in this area, we still have not a clear understanding of quantum loops in (countably) infinite-dimensional state spaces. The most important problems are those distinguishing quantum loops and recursions from the classical ones. A popular view of quantum programming, proposed by Selinger [15], is often summarized by the slogan "quantum data and classical control". This view means that data manipulated by programs are quantum and thus they can be in quantum superpositions. However, the control states of programs are still classical in the sense that it is not allowed to execute a quantum superposition of two statements. The investigation of quantum loop and recursive programs mentioned above is carried out in the paradigm of quantum data and classical control. Recently, some researchers introduced quantum control structures into quantum programs and initiated the studies of quantum programming in the much more complicated paradigm of quantum data and quantum control [16]. Thereofore, an interesting issue is to extend the

1) Yu N K, Ying M S. Termination of concurrent quantum programs. 2012

investigation of quantum loops and recursions into the paradigm of quantum data and quantum control which adds a new dimension of complexity in understanding behaviors of quantum loop and recursive programs.

3 Semantics of quantum programming languages

Formal semantics of a programming language gives 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 fact that human intuition is much better adapted to the classical world than the quantum world is one of the major reasons it is difficult to find efficient quantum algorithms. It should also imply that programmers will commit many more faults in designing programs for quantum computers than programming classical computers. Thus, it seems that giving clear and formal semantics to quantum programming languages is even more critical than in classical computing.

Various semantics of quantum programming languages have been introduced. The operational or denotational semantics of some quantum programming languages were already provided when they were defined; for example qGCL, QPL and QML. Semantic techniques for quantum computation have also been investigated in some abstract, language-independent ways. For example, Abramsky and Coeck [23] proposed a category-theoretic formulation of the basic postulates of quantum mechanics, which can be used to give an elegant description of quantum programs including quantum protocols such as teleportation, logic-gate teleportation, and entanglement swapping. An edited volume collecting the main semantic techniques in quantum computation was recently published [24].

Since it provides a goal-directed program development strategy, predicate transformer semantics has a very wide influence in classical programming methodology. Two approaches to predicate transformer semantics of quantum programs have been proposed in the literature. The first was proposed by Sanders and Zuliani [13] in designing qGCL. In their approach, quantum computation is reduced to probabilistic computation by the observation (measurement) procedure. Thus, predicate transformer semantics developed for probabilistic programs can be conveniently applied to quantum programs. The second was proposed by D'Hondt and Panagaden [25], where the notion of predicate is directly taken from quantum mechanics, that is, a quantum predicate is defined to be an observable (a Hermitian operator) with eigenvalues within the unit interval. In this approach, forward operational semantics of quantum programs are described by super-operators (completely positive operators), and a beautiful duality between state-transformer (forwards) and predicate-transformer (backwards) semantics is observed by employing the Kraus representation theorem for super-operators. One of the advantages of the second approach is that it provides a very natural framework to model and reason about quantum programs. However, in order to further develop this approach, a major obstacle has to be overcome, namely non-commutativity of quantum weakest preconditions represented by Hermitian operators. The significance of this problem comes from the following two observations: (1) Physical simultaneous verifiability of quantum weakest preconditions depends on commutativity between them according to the Heisenberg uncertainty principle; (2) Various logical operations of quantum weakest preconditions will be needed in reasoning about complicated quantum programs, but defining these operations requires commutativity between the involved quantum predicates [26]. The authors [27] found several conditions under which quantum weakest preconditions commute. They [28] further systematically developed quantum predicate transformer semantics with a special class of quantum predicates, namely projection operators. Focusing on projection predicates allowed them to use rich mathematical methods developed in Birkhoff-von Neumann quantum logic [29]. In particular, the notion of commutator introduced by Takeuti [30] in quantum set theory (and widely used in automata theory based on quantum logic [31, 32]) helped them to establish various healthiness conditions of quantum programs, e.g. termination law, conjunctivity, disjunctivity and continuity. An interesting topic for further studies in this direction is to establish link between the existing two approaches to quantum predicate transformer semantics, viz. probabilistic and Hermitian operator approaches, employing Gleason Theorem [33].

4 Verification of quantum programs

4.1 Proof systems

To help reason about the correctness of (sequential) quantum programs, several proof systems for verification of quantum programs have been developed. Baltag and Smets [34] presented a dynamic logic formalism of information flows in quantum systems. Brunet and Jorrand [35] introduced a way of applying Birkhoff and von Neumann's quantum logic in reasoning about quantum programs. Chadha et al. [36] proposed a Hoare-style proof system for reasoning about imperative quantum programs in which only bounded iterations are allowed. Some useful proof rules were proposed in [37] for purely quantum programs. Finally, a full-fledged Hoare logic for both partial and total correctness of quantum programs was developed in [38].

4.2 Quantum Sharir-Pnueli-Hart method

Sharir, Pnueli and Hart [39] presented a general method for proving properties of probabilistic programs, in which a probabilistic program is modelled by a Markov chain and an assertion on the output distribution is extended into an invariant assertion on all intermediate distributions. Their method is essentially a probabilistic generalisation of the classical

Floyd inductive assertion method. In [40], the authors considered quantum programs modelled by quantum Markov chains which are defined by super-operators. It is shown that the Sharir-Pnueli-Hart method can be elegantly generalised to quantum programs by exploiting the Schrödinger-Heisenberg duality between quantum states and observables. In particular, a completeness theorem for the Sharir-Pnueli-Hart verification method of quantum programs is established. On the other hand, the Sharir-Pnueli-Hart method is in principle effective for verifying all properties of quantum programs that can be expressed in terms of Hermitian operators (observables), but it is not feasible for many practical applications because of the complicated calculation involved in the verification. For the case of finite-dimensional state spaces, the authors found a method for verification of quantum programs much simpler than the Sharir-Pnueli-Hart method by employing the matrix representation of super-operators and Jordan decomposition of matrices. In particular, this method enables us to compute easily the average running time and even to analyze some interesting long-run behaviors of quantum programs in a finite-dimensional state space.

5 Implementations of quantum programming languages: Quantum compilers

The majority of previous research on quantum programming has been devoted to designing high-level languages. From a practical viewpoint of implementing high-level quantum programming languages and designing quantum compilers, we should also start with the lower levels in the hierarchy of programming languages and work upward. Some research in this direction has already been reported in the literature; for example, Svore et al. [41] proposed a layered quantum software architecture which is indeed a four-phase design flow mapping a high-level language quantum program onto a quantum device through an intermediate quantum assembly language. Zuliani [42] designed a compiler for 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. [43] 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 complier for a subset of Selinger's QPL. All of these designs are based on the popular circuit model of quantum computation. Nevertheless, Danos et al. [44] presented an elegant low-level language based on a novel and promising physical implementation model of quantum computation, namely measurement-based one-way quantum computer. One advantage of their language is that a standardization theorem was established, which enables us to transform all programs into a standard form executable in a novel physical architecture allowing performing all the entanglement in the beginning.

The authors [45] defined a quantum extension of a classical flowchart language. At the current stage, quantum computer hardware is still very limited and its architecture is uncertain, so it seems appropriate to choose a flowchart language as a low-level language. The language defined in [45] possesses two classes of program variables: classical variables and quantum variables. These classes of variables are separated, and a program state consists of a state of classical variables and a state of quantum variables. The language is obtained from the classical flowchart language by adding two kinds of commands for quantum operations, namely unitary transformations and measurements. The outcome of a measurement is assigned to a classical variable. Therefore, measurement command provides a way of connecting classical and quantum variables. Only classical variables are allowed to occur in the test condition of a conditional jump. This embodies the idea of classical control and quantum data, and thus it is consistent with the QRAM model in [11]. To compare the expressibility of the quantum flowchart language with that of higher-level quantum programming languages, we introduced a quantum extension of while-language. In particular, we presented a structured quantum programming theorem which gives a translation from quantum flowchart programs to quantum while-programs. This can be seen as a theoretical support to the structural programming paradigm for quantum computers pursued by Ömer [12].

6 Quantum process algebras

Process algebras (or process calculi) are popular formal models of classical 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 recently proposed by some researchers. To provide formal techniques for modelling, analysis and verification of quantum communication protocols, Gay and Nagarajan [46] 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 [47] 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. In a series of papers [48–51], the authors 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 probabilistic bisimulation between quantum processes was introduced, and their congruence properties were established. Also, a

theory of approximate strong bisimulations (strong bisimulation metrics) for quantum process algebras is proposed. As is well-known, a set of classical gates is universal if it can be used to compute exactly an arbitrary Boolean function. However, exact universality does not make sense in quantum computation because all quantum gates form a continuum which cannot be generated by a finite set of quantum gates. Instead, a set of quantum gates is said to be universal provided any quantum gate can be approximated to arbitrary accuracy by a circuit constructed from the gates in this set. Approximate bisimulation can be used to describe approximation between quantum processes and, in particular, implementation of a quantum process by some (usually finitely many) special quantum gates. The most spectacular result in fault-tolerant quantum computation is the threshold theorem which means 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 quantum sequential computation. Its generalization in quantum concurrent computation would be a great challenge. The notions of approximate bisimulation and noisy channel [52] will provide us with a formal tool for observing robustness of quantum concurrent computation against inaccuracy in the implementation of its elementary gates, and we guess that it can be used to establish a concurrent generalization of the (fault-tolerance) threshold theorem.

The role of entanglement in quantum computation and information is a very interesting and important problem. It has been carefully analyzed by several researchers in the framework of sequential computation (see for example [53]). It seems that entanglement is more essential in quantum concurrent computation than in sequential quantum computation. The algebra of quantum processes developed in [48, 49, 51] should provide us with a natural and cleaned up formal framework for analyzing the role of entanglement in concurrent quantum computation.

Quantum information science is usually divided into two subareas: quantum computation and quantum communication. Quantum computation offers the possibility of considerable speed over classical computation by exploring the power of superposition of quantum states, and communication protocols are proposed by employing quantum mechanical principles (in particular, no-cloning property and entanglement), which are provable secure. Studies of quantum process algebras and distributed quantum computation [50] allow us to glue the two subareas of quantum information science.

7 Applications of programming methodology to quantum engineering

As pointed out by Dowling and Milburn [54], 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 do 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 behaviours of the systems she/he designed, and a bug in her/his design may cause some serious problems and even disasters. Therefore, correctness, safety and reliability of complex engineering systems have attracted wide attention and have been systematically studied in various engineering fields. As pointed out in Section 1, human intuition is much better adapted to the classical world than the quantum world. This also implies that human engineers will commit many more faults in designing and implementing complex quantum engineering systems. Thus, correctness, safety and reliability problem will be even more serious in quantum engineering than in today's engineering.

In the last four decades, computer scientists have systematically developed theories of correctness and safety as well as methodologies, techniques and even automatic tools for correctness and safety verification of computer systems. In particular, model-checking [55] is an effective automated technique that checks whether a formal (temporal logic) property is satisfied in a formal model of a system. It has become one of the dominant techniques for verification of computer systems nearly 30 years after its inception. Many industrial-strength systems have been verified by employing model-checking techniques.

A question then naturally arises: is it possible and how to use model-checking techniques to verify correctness and safety of quantum engineering systems? It seems that the current model-checking techniques cannot be directly applied to quantum systems because of some essential differences between the classical world and the quantum world. To develop model-checking techniques for quantum systems, at least the following two problems must be addressed:

(1) The classical system modelling method cannot be used to describe the behaviors of quantum systems, and the classical specification language is not suited to formalize the properties of quantum systems to be checked. Therefore, we need to carefully and clearly define a conceptual framework in which we can properly reason about quantum systems, including formal models of quantum systems and formal description of temporal properties of quantum systems.

(2) The state spaces of the classical systems that model-checking techniques can be applied to are usually finite or countably infinite. However, the state spaces of quantum systems are inherently continuous even when they are finite-dimensional. In order to check quantum systems, we have to exploit some deep mathematical properties 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, of their state spaces.

There have been quite a few papers devoted to model-

checking quantum systems, with the target of checking quantum communication protocols. For example, Gay et al. [56] used the probabilistic model-checker PRISM [57] to verify the correctness of several quantum protocols including BB84 [58]. Furthermore, they [59, 60] developed an automatic tool QMC (Quantum Model-Checker). QMC uses the stabilizer formalism [61] for the modelling of systems, and the properties to be checked by QMC are expressed in Baltazar, Chadha, Mateus and Sernadas quantum computation tree logic [62, 63]. The purpose of [64] is to develop model-checking techniques that can be used not only for quantum communication protocols but also for other quantum engineering systems. The authors defined a mathematical framework in which we can examine various linear-time properties of quantum systems, such as safety and liveness properties. Also, they discovered an algorithm for checking invariants of quantum systems.

8 Conclusions

Programming research for classical computers is a huge area, including numerous subareas such as programming languages and semantics, programming systems and tools, specification, testing and verification, programming paradigms, to name just a few. A full-fledged discipline of quantum programming will be at least as large as classical programming. It should be emphasised that the subject of quantum programming methodology is not a simple and straightforward generalisation of its classical counterpart. Some completely new phenomena arise in the quantum case. So we have to address some major problems that would not arise in the realm of classical and probabilistic programming. These problems come from the w eird nature of quantum systems. For example, no-cloning of quantum data requires us to distinguish much more carefully call-by-value and call-by-name semantics of programming languages. It also means that the typing systems of quantum programming languages are essentially different from those of classical computing [46]. Non-commutativity of observables (not simultaneous verifiability according to the Heisenberg uncertainty principle) is another typical feature of quantum systems. It was observed by the authors [27] that non-commutativity is a major obstacle in developing predicate transformer semantics of quantum programming languages.

Quantum programming methodology is still at the very early stage, and its knowledge base is highly fragmentary and disconnected. Certainly, much of the major problems is left unsolved. In particular, the studies of quantum programming reviewed in this paper is solely based on the circuit model of quantum computation. It is interesting to investigate principles and semantics of programming languages for other models of quantum computation, such as topological, adiabatic, and measurement-based quantum computation. For example, the mathematical description of topological quantum computation [65] 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) will even bring novel and exciting open problems to these mainstream areas of mathematics.

This work was partly supported by the Australian Research Council (DP110103473) and the National Natural Science Foundation of China (60736011).

1 Copsey D, Oskin M, Impens F, et al. Toward a scalable, silicon-based quantum computing architecture. IEEE J Select Topics Quant Electron, 2003, 9: 1552–1569
2 Hoare T, Milner R. Grand Challenges in Computing Research. Swindon, UK: The British Computer Society, 2004
3 Gay S J. Quantum programming languages: Survey and bibliography. Math Struct Comput Sci, 2006, 16: 581–600
4 Selinger P. A brief survey of quantum programming languages. In: Proceedings of the 7th International Symposium on Functional and Logic Programming, LNCS 2998, 2004. 1–6
5 Ying M S. Foundations of quantum programming. In: Proceedings of the 8th Asian Symposium on Programming Languages and Systems (APLAS 2010), LNCS 6461, 2010. 16–20
6 Rüdiger R. Quantum programming languages: An introdutionary overview. Comput J, 2007, 50: 134–150
7 Miszczak J A. Models of quantum computation and quantum programming languages. Bull Polish Acad Sci: Tech Sci, 2011, 59: 305–324
8 Ying M S. Quantum computation, quantum theory and AI. Artif Intell, 2010, 174: 162–176
9 Xu J F, Song F M. Quantum programming languages: A tentative study. Sci China Ser F-Inf Sci, 2005, 51: 623–637
10 Abramsky S. High-level methods for quantum computation and information. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004), 2004. 410–414
11 Knill E H. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996
12 Ömer B. Structural quantum programming. Ph.D. Thesis. Vienna: Vienna University of Technology, 2003
13 Sanders J W, Zuliani P. Quantum programming. In: Proceedings of Mathematics of Program Construction, LNCS 1837, 2000. 88–99
14 Bettelli S, Calarco T, Serafini L. Toward an architecture for quantum programming. Eur Phys J D, 2003, 25: 181–200
15 Selinger P. Towards a quantum programming language. Math Struct Comput Sci, 2004, 14: 527–586
16 Altenkirch T, Grattage J. A functional quantum programming language. In: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS), 2005. 249–258
17 Tafliovich A, Hehner E C R. Quantum predicative programming. In: Proceedings of the 8th International Conference on Mathematics of Program Construction (MPC), LNCS 4014, 2008. 433–454
18 Tafliovich A, Hehner E C R. Programming with quantum communication. Electron Notes Theoret Comput Sci, 2009, 253: 99–118
19 Bernstein E, Vazirani U. Quantum complexity theory. SIAM J Comput, 1997, 26: 1411–1473
20 Ying M S, Feng Y. Quantum loop programs. Acta Inform, 2010, 47: 221–250
21 Li Y J, Yu N K, Ying M S. Termination of nondeterministic quantum programs. arXiv: 1201.0891v1
22 Hart S, Sharir M, Pnueli A. Termination of probabilistic concurrent programs. ACM Trans Progr Lang Syst, 1983, 5: 356–380

23 Abramsky S, Coecke B. A categorical semantics of quantum protocols. In: Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS), 2004. 415–425

24 Mackie I, Gay S. Semantic Techniques in Quantum Computation. New York: Cambridge University Press, 2010

25 D'Hondt E, Panangaden P. Quantum weakest preconditions. Math Struct Comput Sci, 2006, 16: 429–451

26 Varadarajan V S. Geometry of Quantum Theory. New York: Springer-Verlag, 1985

27 Ying M S, Chen J X, Feng Y, et al. Commutativity of quantum weakest preconditions. Inform Proc Lett, 2007, 104: 152–158

28 Ying M S, Duan R Y, Feng Y, et al. Predicate Transformer Semantics of Quantum Programs. Semantic Techniques in Quantum Computation. New York: Cambridge University Press, 2010. 311–360

29 Birkhoff G, von Neumann J. The logic of quantum mechanics. Ann Math, 1936, 37: 823–843

30 Takeuti G. Quantum Set Theory. Current Issues in Quantum Logics. New York: Plenum Press, 1981. 303–322

31 Ying M S. A theory of computation based on quantum logic (I). Theor Comput Sci, 2005, 344: 134–207

32 Ying M S. Quantum Logic and Automata Theory. Handbook of Quantum Logic and Quantum Structures. London: Elsevier, 2007. 619–754

33 Gleason A G. Measures on the closed subspaces of a Hilbert space. J Math Mech, 1957, 6: 885–893

34 Baltag A, Smets S. LQP: The dynamic logic of quantum information. Math Struct Comput Sci, 2006, 16: 491–525

35 Brunet O, Jorrand P. Dynamic quantum logic for quantum programs. Int J Quant Inform, 2004, 2: 45–54

36 Chadha R, Mateus P, Sernadas A. Reasoning about imperative quantum programs. Elect Notes Theor Comput Sci, 2006, 158: 19–39

37 Feng Y, Duan R Y, Ji Z F, et al. Proof rules for the correctness of quantum programs. Theor Comput Sci, 2007, 386: 151–166

38 Ying M S. Floyd-Hoare logic for quantum programs. ACM Trans Progr Lang Syst, 2011, 33: 19

39 Sharir M, Pnueli A, Hart S. Verification of probabilistic programs. SIAM J Comput, 1984, 13: 292–314

40 Ying M S, Yu N K, Feng Y, et al. Verification of quantum programs. arXiv: 1106.4063, 2011

41 Svore K M, Aho A V, Cross A W, et al. A layered software architecture for quantum computing design tools. IEEE Comput, 2006, 39: 74–83

42 Zuliani P. Compiling quantum programs. Acta Inform, 2005, 41: 435–473

43 Nagarajan R, Papanikolaou N, Williams D. Simulating and compiling code for the sequential quantum random access machine. Elect Notes Theor Comput Sci, 2007, 107: 101124

44 Danos V, Kashefi E, Panangaden P. The measurement calculus. J ACM, 2007, 54: 8

45 Ying M S, Feng Y. A flowchart language for quantum programming. IEEE Trans Software Eng, 2011, 37: 466–485

46 Gay S J, Nagarajan R. Communicating quantum processes. In: Proceedings of the 32nd ACM Symposium on Principles of Programming Languages (POPL05), 2005

47 Jorrand P, Lalire M. Toward a quantum process algebra. In: Proceedings of the 1st ACM Conference on Computing Frontiers, 2004

48 Feng Y, Duan R Y, Ji Z F, et al. Probabilistic bisimulations for quantum processes. Inform Comput, 2007, 205: 1608–1639

49 Ying M S, Feng Y, Duan R Y, et al. An algebra of quantum processes. ACM Trans Comput Logic, 2009, 10: 19

50 Ying M S, Feng Y. An algebraic language for distributed quantum computing. IEEE Trans Comput, 2009, 58: 728–743

51 Feng F, Duan R Y, Ying M S. Bisimulations for quantum processes. In: Proceedings of the 38th ACM Symposium on Principles of Programming Languages (POPL11), 2011. 523–534

52 Ying M S. Pi-calculus with noisy channels. Acta Inform, 2005, 41: 525–593

53 Jozsa R, Linden N. On the role of entanglement in quantum-computational speed-up. Proc Royal Soc London Ser A-Math Phys Eng Sci, 2003, 459: 2011–2032

54 Dowling J P, Milburn G J. Quantum technology: The second quantum revolution. Phil Trans Royal Soc London A, 2003, 361: 1655–1674

55 Baier C, Katoen J P. Principles of Model Checking. Cambridge, Massachusetts: MIT Press, 2008

56 Gay S J, Nagarajan R, Papanikolaou N. Probabilistic model-checking of quantum protocols. In: Proceedings of the 2nd International Workshop on Developments in Computational Models (DCM06), 2006

57 Kwiatkowska M, Norman G, Parker P. Probabilistic symbolic model-checking with PRISM: A hybrid approach. Int J Software Tools Tech Transfer, 2004, 6: 128–142

58 Bennett C H, Brassard G. Quantum cryptography: Public key distribution and coin tossing. In: Proceedings of International Conference on Computers, Systems and Signal Processing, 1984

59 Gay S J, Nagarajan R, Panaikolaou N. QMC: A model checker for quantum systems. In: Proceedings of the 20th International Conference on Automated Verification (CAV08), LNCS 5123, 2008. 543–547

60 Papanikolaou N K. Model checking quantum protocols. Ph.D. Thesis. Coventry, UK: University of Warwick, 2008

61 Nielsen M A, Chuang I L. Quantum Computation and Quantum Information. Cambridge: Cambridge University Press, 2000

62 Baltazar P, Chadha R, Mateus P. Quantum computation tree logic—model checking and complete calculus. Int J Quant Inform, 2008, 6: 219–236

63 Baltazar P, Chadha R, Mateus P, et al. Towards model-checking quantum security protocols. In: Proceedings of the 1st Workshop on Quantum Security (QSec07), 2007

64 Ying M S, Li Y J, Yu N K, et al. Model-checking linear-time properties of quantum systems. arXiv: 1101.0303, 2011

65 Preskill J. Topological quantum computation. In: Lecture Notes for Physics 219: Quantum Computation. Pasadena: California Institute of Technology, 2004