Nikolaos K. Papanikolaou

MODEL CHECKING QUANTUM PROTOCOLS

A DISSERTATION SUBMITTED TOWARDS FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisor: *Professor Rajagopal Nagarajan*

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WARWICK
COVENTRY
2008/9

Ἀφιερώνεται στὴ γιαγιὰ τὴ Ντέπη μὲ πολλὴ ἀγάπη

(This work is dedicated to my beloved grandmother Depi)

ABSTRACT

This thesis describes model checking techniques for protocols arising in quantum information theory and quantum cryptography. We discuss the theory and implementation of a practical model checker, QMC, for quantum protocols. In our framework, we assume that the quantum operations performed in a protocol are restricted to those within the stabilizer formalism; while this particular set of operations is not universal for quantum computation, it allows us to develop models of several useful protocols as well as of systems involving both classical and quantum information processing. We detail the syntax, semantics and type system of QMC's modelling language, the logic QCTL which is used for verification, and the verification algorithms that have been implemented in the tool. We demonstrate our techniques with applications to a number of case studies.

DECLARATION

The work described in this thesis is original work and is based on a collaboration with Rajagopal Nagarajan and Simon Gay. The material presented is based on the following publications:

- **S. J. Gay**, **R. Nagarajan** and **N. Papanikolaou**. "Specification and Verification of Quantum Protocols". In: *Semantics of Quantum Computation*. Ed. by **S. J. Gay** and **I. Mackie**. Cambridge University Press. To appear 2009/2010.
- **S. J. Gay**, **R. Nagarajan**, and **N. Papanikolaou**. "Probabilistic Model–Checking of Quantum Protocols". In: *Proceedings of DCM 2006: Proceedings of the 2nd International Workshop on Developments in Computational Models*. arXiv:quant-ph/0504007. 2006.
- **S. J. Gay**, **R. Nagarajan**, and **N. Papanikolaou**. "QMC: A Model Checker for Quantum Systems". In: *Proceedings of 20th International Conference on Automated Verification (CAV 2008) (Princeton, NJ, USA, July 7—14, 2008)*. Vol. 5123. Lecture Notes in Computer Science. 2008, pp. 543–547.
- **S. J. Gay**, **R. Nagarajan**, and **N. Papanikolaou**. "QMC: A Model Checker for Quantum Systems". In: *Proceedings of Workshop on Quantum Cryptography and Security (LQCIL'07):*

Lisbon Quantum Computation, Information and Logic Meetings Series (Lisbon, Portugal, July 18—20, 2007). 2007.

- **N. Papanikolaou**. "Reasoning Formally About Quantum Systems: An Overview", ACM SIGACT News, **36**(3), pp. 51—66, 2005.

The development of the operational semantics incorporates several constructive suggestions from Hynek Mlnařík.

Although not related directly to the results of this thesis, I have contributed to the following additional publications. I wish to acknowledge this here since there are some theoretical connections to the work presented here.

- **Tim Davidson**, **Hynek Mlnařík**, **Rajagopal Nagarajan**, and **Nikolaos Papanikolaou**. "Verifying Communicating Quantum Processes using QMC". Unpublished technical report. 2008.
- **Pedro Baltazar**, **Paulo Mateus**, **Rajagopal Nagarajan** and **Nikolaos Papanikolaou**. "Exogenous Probabilistic Computation Tree Logic", In: *Proceedings of the Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL '07), (Braga, Portugal, 24—25 March, 2007)*

N.P.

ACKNOWLEDGEMENTS

This work would not have been possible without the support of my mentors and superiors, my colleagues and friends, and above all my parents. I wish to formally acknowledge all the support I have received here, including all sources of funding for my research.

I am highly indebted to my academic supervisor and friend, Rajagopal Nagarajan, who encouraged me to continue with my postgraduate studies and undertake an interesting project in an exciting and novel field. Raja employed me on his EPSRC and EU grants and gave me the opportunity to attend numerous workshops and conferences, and to meet and collaborate with other researchers including close collaborators of his. It was Raja who gave the initial thrust to this work, and who pointed out the necessity of verification techniques for quantum systems. Raja has been a generous supervisor who has definitively shaped my formative years as a researcher. Our discussions, which took place in Starbucks coffee shops almost exclusively ☕, of (among others) quantum bomb–testing ☢, Apple Macs ⌨, and various female antics 👩, will remain etched in my memory for years to come.

Simon Gay has had a substantial influence on the technical aspects of this work, and has been a great mentor to me. Simon has shown me the value of persistence and discipline, and

has provided a very comfortable atmosphere for collaboration at the University of Glasgow on numerous occasions. Our joint work has been very fruitful and I have enjoyed every minute of it. Simon's expertise in programming language theory, semantics and verification, as well as his shared fascination with quantum mechanics, have made him invaluable. His incisiveness has led to better papers, cleaner code and deeper understanding. I cannot thank him enough.

My advisor, Marcin Jurdziński, has encouraged me to look closely at purely theoretical issues in my field, and to consider algorithmic aspects of verification. He has been a source of encouragement, an astute advisor, and also a kind neighbour in Leamington Spa, whose company I have enjoyed.

Over the years I have been engaged in useful discussions with several professors in the Department of Computer Science at Warwick, including Mike Paterson, Ranko Lazic, Steve Matthews, Sara Kalvala, and Doron Peled. All of these people have had an influence on my thinking about computer science as a discipline.

Professor Paulo Mateus at the Instituto Supérior Tecnico in Lisbon, Portugal, invited me to give a seminar for the Security and Quantum Information Group (SQIG) in November 2006. Not only did this give me the opportunity to learn about his group's work on developing logics for reasoning about probabilistic and quantum systems, but it resulted in a long–term collaboration for which I am very grateful. Pedro Baltazar (also from IST Lisbon) also participates in this collaboration and I particularly benefitted from our discussions during his stay in Oxford in 2008.

I have merited from many discussions with Hynek Mlnařík (currently with the Warwick Institute for Financial Computing), whose background in design and implementation of programming languages has proved very helpful. Hynek helped me significantly with the development of the semantics for QMC, and gave sound advice on various technical aspects of this work. I look forward to further collaboration with him in the future.

As a result of participating in numerous workshops, meetings and conferences, I have had the opportunity to meet and discuss my work with several established computer scientists, with whom I have had valuable discussions; this includes Samson Abramsky, Graham Birtwistle, Bob Coecke,

Edmund Clarke, Anuj Dawar, Vincent Danos, Jozef Gruska, Philippe Jorrand, Elham Kashefi, Marta Kwiatkowska, Prakash Panangaden, Mark Ryan, Peter Selinger, and Joseph Sifakis.

My office mates and good friends, Hongyang Qu, Ashutosh Trivedi, Ritesh Krishna, Dimitris Vavoulis, and Timothy Davidson, have stood by me through all these years and we have had a great many laughs together in the *Formal Methods and Quantum Information Processing Lab* and beyond.

My research and research travel have been financially supported by the following sources:

- the Formal Methods Group, Department of Computer Science, University of Warwick (which partially funded my Ph.D. studies)
- the following grants from the Engineering and Physical Sciences Research Council (EPSRC): GR/S34090/01 (which partially funded my Ph.D. studies), EP/E006833/2 ("QNET"), and GR/S86037/01 ("SymNet")
- the European Union Sixth Framework Programme (Project "SECOQC")
- the FCT-POCI Project FEDER POCI/MAT/55796/2004 ("QuantLog")
- A CRUP/Treaty of Windsor grant (from IST Lisbon).

This support has made it possible for me to survive as a Ph.D. student while also enabling me to travel extensively for my work, within the UK as well as abroad.

It is a privilege to record here my thanks to my examiners Elham Kashefi and Ian Mackie for their precious comments and feedback.

I am extremely grateful to Professor Sadie Creese and Professor Michael Goldsmith for giving me the opportunity to start a challenging and rewarding research fellowship at the end of my doctoral studies.

I would like to thank each and every one of my ...muses for the pain and love they have given me at different times during my research; they *know* who they are.

Finally, I thank my parents for all their love and support, which has enabled me to get as far as I have in my life.

— N. P.

CONTENTS

Abstract iii

Declaration iv

Acknowledgements vi

List of Figures xii

Preface xiv

Chapter 1. Introduction 1

1.1. Context 3

1.2. Motivation 8

1.3. Methodology 9

1.4. Thesis Contribution 13

1.5. Previous Work by the Author and Collaborators 14

1.6. Related Work 15

1.7. Outline 17

Chapter 2. Background 18
2.1. The Postulates of Quantum Mechanics 19
2.2. Hilbert Space 20
2.3. Operators and Matrices 23
2.4. Projective Measurements 24
2.5. Quantum Gates, Circuits and Approximate Universality 24
2.6. The Stabilizer Formalism 27
2.7. Quantum Protocols 30
2.8. Concluding Remarks 40

Chapter 3. Probabilistic Model Checking and Efficient Simulation 41
3.1. Adapting the Probabilistic Model Checker PRISM 42
3.2. The Limitations of the PRISMGEN Approach 51
3.3. Efficiently Simulable Quantum Protocols 52
3.4. Concluding Remarks 54

Chapter 4. Specifying Quantum Protocols and Their Properties 56
4.1. QMCLANG: A Modelling Language 57
4.2. Syntax 60
4.3. Semantics 61
4.4. The Executability Predicate 69
4.5. Type System 71
4.6. The EQPL and QCTL Specification Logics 76
4.7. Interpreting QCTL Formulae over QMCLANG Models 81
4.8. Protocols and Properties Expressible Within the QMCLANG–QCTL Combined Framework 82
4.9. Concluding Remarks 82

Chapter 5. Implementation 84
5.1. QMC Tool Description 85

5.2. Parser Implementation 86
5.3. Scheduler Implementation 87
5.4. Interpreter Implementation 88
5.5. Model Checker Implementation 93
5.6. Special Extensions 104
5.7. Concluding Remarks 104

Chapter 6. Applications 106
6.1. Quantum Teleportation 106
6.2. Quantum Coin Flipping 110
6.3. Quantum Key Distribution with Error Correction 113
6.4. A Quantum Error Correcting Network 118
6.5. Quantum Secret Sharing 124
6.6. Concluding Remarks 127

Chapter 7. Review and Conclusion 128
7.1. Summary 129
7.2. Discussion 130
7.3. Conclusions 135

Bibliography 136

Index 150

LIST OF FIGURES

2.1 The principal quantum gates arising in protocols. 25

2.2 The effect of the Clifford group gates on the Pauli gates under Hermitian conjugation. 29

2.3 Quantum circuit diagram for the superdense coding protocol. 31

2.4 Quantum circuit diagram for the teleportation protocol. 32

2.5 Quantum circuit diagram for the qubit bit–flip code. 33

3.1 Internal probabilistic state transition system for superdense coding. 47

4.1 An example with sending and receiving of variables. 59

4.2 An example with looping. 59

4.3 Concrete Syntax for QMCLANG. 60

4.4 Internal Syntax for QMCLANG. 61

4.5 Transition Relations for the Operational Semantics. 62

4.6 Executability Predicates. 74

4.7 Definition of an evaluation function for expressions. 75

4.8 The syntax of EQPL (from [98]). 77

4.9 The semantics of EQPL. 79

4.10 The semantics of QCTL. 80

5.1 QMC's overall structure. 85

5.2 The Graphical User Interface of QMC. 86

5.3 The structure of a stabilizer tableau. 90

5.4 The binary tableau representation of a quantum state. 91

5.5 The Aaronson–Gottesman simulation algorithm 92

5.6 EQPL Verification Algorithm: Quantum and Classical Formulae. 95

5.7 EQPL Verification Algorithm: Real Terms. 96

5.8 EQPL Verification Algorithm: Complex Terms. 97

5.9 Fixed point model checking algorithm for QCTL. 101

5.10 The QCTL model checking algorithm (continues in Fig. 5.11). 102

5.11 The QCTL model checking algorithm (continued from Fig. 5.10). 103

PREFACE

A system is defined as a collection of components which work together as a whole, toward the attainment of a goal, the completion of a task, the execution of a computation. We are surrounded by a vast number of systems, man–made and natural, all of which exhibit varying degrees of complexity. The proper functioning of a modern society relies on a vast number of diverse systems (political, economic, technological) with many intricate relationships and interdependencies. Therefore it is essential to look for practical means of managing complexity, means of understanding, and for tools for analysing different types of systems.

Often the complexity of a system is so great that, in order to construct a theory which accounts for its behaviour, one must combine actual observations of some components with several hypotheses regarding the rest. Natural science is founded on a method of systematic observation, and a scientific theory cannot be fully accepted unless it agrees with experiment. The use of the scientific method provides a solid assurance that a theory is correct, and provides a defense against arbitrary conjecture and errors of human judgement.

Theoretical computer science relies heavily on the use of *abstraction*, namely the description of a system at a level of detail which is just sufficient for an analysis. Abstraction involves eliminating unnecessary information about a system from view, leaving only the ingredients which are considered relevant. The resulting description is known as a *model*. The benefits of abstraction are twofold. Firstly, it allows one to focus on the essential components of a system, thus reducing its apparent complexity. It also permits one to consider the interactions and dependencies between different components in a system.

It may seem downright foolish to start ignoring elements of a system while trying to understand it. However, a useful means of managing complexity should allow the user to separate and re–group these different elements in his mind, and to investigate each one in turn. Just which aspects of a system may be temporarily ignored is a matter which depends on the intended analysis. In spoken language, one is accused of "forgetting the forest for the trees" when he or she insists on looking at minor details and missing important points (the "big picture"). One does not study the patterns on the leaves when trying to decide whether a particular tree should be cut down. On the other hand, it is often argued that "the devil is in the detail;" this reminds us that leaf patterns can be extremely sophisticated and beautiful, and their study very difficult. System engineers are confronted with design decisions daily, and choosing the right abstraction usually involves a much finer distinction than this.

A *high–level* abstraction is one in which the emphasis is put not on individual components, but on the overall structure of a system and on the communication that occurs at the interface of the different components. This level of abstraction enables reasoning about interactions and dependencies. Oftentimes it is more important that components interact correctly with each other than it is that any particular one functions fully. For instance, a flight control system should be able to compensate for a failing aircraft engine by boosting all other engines, as well as by activating relevant safety mechanisms. Any flaw in the interactions between the control software and an aircraft's machinery could be fatal, while the absence of such a flaw would save many lives.

Clearly, an abstraction which is too high–level will probably be useless. Guaranteeing that all the machinery in an aircraft is well connected and that hence, communication between all

components is possible, is only useful if one can also show that adequate communication will happen when necessary. Thus, while the technique of abstraction can be helpful in managing complexity, it also has fundamental pitfalls. This is true of any method used in science, and it is the engineer's role to apply the method with care.

It can be argued that the development of suitable abstractions and, hence, models is the starting point for any serious scientific study of a given system. Once a model has been built, there are numerous methods of analysis; some of these require pencil and paper, others are entirely computer–based. It suffices to say here that the use of automated analysis techniques (including simulation and formal verification) has proved highly successful in recent years, and that such techniques have been used effectively for the design and validation of various real–world systems, ranging from simple communication schemes to commercial airliners. Computer–based systems modelling and validation is superior to manual proofs of correctness in the case of large and complex systems, for which assurances regarding system behaviour are required in short time frames.

Any method which is intended for the development of complex system models must provide a suitable level of abstraction, but it should also be rigorous and mathematically sound. In particular, there is much to be gained from the use of well–studied mathematical techniques in system design, especially when it is desirable to have formal proofs of correctness. There are many other benefits in using such techniques; for instance, it is possible to directly compare aspects of two different systems by examining the properties exhibited by the mathematical objects which are used to model them.

Of interest to us in this thesis are systems comprising communication, concurrency and computations involving certain specific physical phenomena (namely, phenomena of quantum physics such as entanglement and the behaviour caused by quantum measurement). In particular, we are interested in means of analysis of *protocols* for communication and cryptographic purposes which make use of quantum information. Quantum protocols have particularly important applications in cryptography. Several quantum protocols have been proposed for cryptographic tasks such as oblivious transfer, bit commitment and key distribution.

The so–called BB84 protocol for quantum key distribution, which allows two users to establish a common secret key using a single quantum channel, has been shown to be unconditionally secure against all attacks permitted by the laws of quantum mechanics. Such a degree of security has never been guaranteed by any classical cryptographic protocol, and the discovery has incited widespread interest in the properties of quantum protocols. Furthermore, practical quantum cryptographic devices are commercially available (*e.g.* from the companies *Id Quantique*[1] and *MagiQ*[2]).

We argue that detailed, automated analyses of protocols such as these facilitate our understanding of complex quantum behaviour and enable us to construct valuable proofs of correctness. Such analyses are especially important to manufacturers of commercial devices based on such protocols; the actual security of commercial quantum cryptographic systems, for example, is worth an in–depth investigation. Communication protocols have always been under scrutiny by computer scientists, who have developed numerous techniques for analysing and testing them, including process algebras, formal specification languages and automated verification tools. Automated verification techniques, such as *model-checking* and *theorem proving,* are frequently targeted at protocols and have been used to detect faults and subtle bugs. In this work we hope to obtain corresponding benefits for quantum protocols.

[1] `http://www.idquantique.com`
[2] `http://www.magiqtech.com`

CHAPTER

I

INTRODUCTION

A quantum computer, if built, will be to an ordinary computer as a hydrogen bomb is to gunpowder, at least for some types of computations.

— From [28]

Quantum theory is widely accepted as the most successful theory of natural science. Its precepts challenge our fundamental understanding of the universe, and are often in direct conflict with what our intuition leads us to believe. It provides us with a description of the behaviour of matter on the atomic scale. The implications of quantum theory for information processing are very hard to ignore; indeed, to harness the potential of the quantum world is to enable extremely powerful computational techniques, as well as novel means of data communication. This is the province of the emerging field of *quantum computation and quantum information*.

The study of this field is, all at once, a *challenge,* a *necessity,* a *potential* and a *drive,* as identified by Gruska [71]. Let's investigate these in turn.

The connections between information and its physical manifestation have raised certain fundamental theoretical questions at the interface of computer science and physics. A datum always has a physical representation, whether it be a written mark on a piece of paper, or a flag set in a computer register; but does it have an existence of its own, independent of the medium, and how do the physics of the medium influence its information content? On the other hand, are physical laws related to the rules that govern the flow of information in a system? Machines are inherently physical devices, and their operation is governed by natural laws; do these laws dictate the nature of computation itself? Indeed, Lloyd [91] of MIT has suggested that the universe itself is a vast quantum computer, and that it is the duty of physicists to discover the equations which govern its behaviour. Clearly, the study of the connections between computation and the very nature of matter poses many interesting *challenges.*

Since the inception and, especially, the manufacture of the transistor in the late 1940s, there has been a constant strive to develop ever smaller digital circuits. There has been an exponential growth of the number of transistors on computer chips in the last forty years or so, as predicted by Moore [107] in 1965. Keyes [85] has extrapolated that, by the year 2020, this constant miniaturisation of computer circuits will reach the atomic level. At that point it will be necessary to control and utilise quantum phenomena during the process of transcribing and manipulating individual bits of information. Thus, an understanding of how these phenomena can be used for information processing will become an absolute *necessity.*

Quantum computation and quantum information has been shown to have great *potential.* In particular, the use of quantum phenomena allows the implementation of novel algorithms and cryptographic protocols with no classical analogue. The quantum algorithms of Shor [137] and Grover [70] for prime factorisation and inverse database search, respectively, have significantly better complexity than the best known classical algorithms for the same tasks (in the former case, there is an exponential improvement). Protocols for quantum key distribution, such as BB84 [20],

have been shown to achieve unconditional security, a result which is not attainable using classical computational means.

Finally, the study of quantum information processing provides a *drive* to explore the fundamental limits of nature and the physical laws which govern the universe on the lowest scale. While quantum theory has been repeatedly shown to agree with experiment, physicists continue to be at odds regarding its philosophical basis. There are still many open questions in quantum mechanics, ranging from the origin of randomness in measurement to the issue of reconciling the theory with the general theory of relativity [11, 74, 91].

The focus in this thesis will be on the implications of this growing field for the development of communication and cryptographic schemes, namely, for systems and protocols combining classical computations with quantum phenomena, such as the superposition of states, entanglement, the randomness of quantum measurement, and the no–cloning property; we will have more to say about each of these in Section 1.1.1.1. More concretely, our aim is to describe formal verification techniques and tools for quantum protocols.

This chapter sets out the context for this work by giving a brief account of the field; the motivation for our approach, and the intended contribution, are then described. A review of previous and related work is also given. Finally, the contents of the remaining chapters are outlined.

1.1. Context

There are numerous texts on quantum computing [13, 15, 72, 75, 84, 94, 140, 141] and quantum information [16, 31, 92]; the standard references on the field as a whole are [71, 111]. The upsurge in public interest in quantum theory and its applications in recent years is evidenced by the significant number of special issues of popular periodicals, such as *Scientific American* and *New Scientist*, devoted to its different facets; these make a rewarding read. Here we review some pertinent aspects of quantum information and quantum computation.

1.1.1. Quantum Information. Landauer [89] characteristically observed that every kind of information has a physical manifestation with which its existence is inextricably linked. This implies that the applicability of physical laws extends to cover all sorts of data. Thus, physics

imposes restrictions on data representation and storage. For instance, if there is such a thing as a fundamental lower bound on the size of all physical objects, then this constrains the quantity of data that may *ever* be stored. In a widely publicised lecture, Feynman [55] argued that there is no fundamental physical obstacle to writing the entire contents of the *Encyclopaedia Britannica* on the head of a pin. Nevertheless, Feynman and his colleagues were aware of the difficulties of controlling matter on this scale.

Quantum information theory is concerned with how the phenomena which occur on the atomic level can be used constructively for the representation, storage and transmission of data. The potential of such phenomena for communication and cryptographic purposes was realised by Wiesner [148], who proposed the use of polarised photons to represent individual bits in a key string. The novelty here was to regard quantum mechanics not as an obstacle, but as a means for achieving a communication goal. Several other communication protocols have been proposed which make use of quantum phenomena, and the most characteristic ones will be detailed in Chapter 2.

1.1.1.1. *Properties of Quantum Systems.* In order to realise how quantum protocols work, it is necessary to understand quantum states and their properties. These are documented in detail in all the standard texts; we defer the mathematical treatment to the next chapter.

A quantum system such as an atom, an electron, or a photon are characterised by several physical attributes, such as position, momentum, spin and polarisation. The state of a particular system describes these attributes at a given time instant. In quantum physics, a state is described by a *wavefunction*. It is a well–known fact that quantum systems exhibit both particle–like and wave–like behaviour (this is known as *wave–particle duality* [40]) and the wavefunction is designed to reconcile these two views. Furthermore, the state of a quantum system evolves over time in accordance with a mathematical law known as *Schrödinger's equation*.

According to Williams and Clearwater [149, p.27] there are six essential properties of quantum states that distinguish quantum information from classical information. These are superposition, entanglement, interference, non–determinism, non–cloneability, and non–locality.

The so–called *superposition principle* [29, 40, 101] states that, if a quantum system can be in one of given states, then it can also be in a state that is a mixture, a simultaneous combination of these. This principle is central to quantum information, as it distinguishes qubits (*i.e.* quantum bits) from classical bits, which can only ever be in one of two states 0 and 1, but never in both. Superposition allows many classical states to be encoded efficiently in a single quantum state, leading to what is known as *quantum parallelism*.

When several states are combined together, they are said to have a *joint* quantum state. This state may be separable, so that the individual states of the different particles can be identified. However, it may contain *entanglement*, in which the states of particular particles are tied together and cannot be separated or identified individually. Entanglement is a key feature of quantum mechanics, used extensively in quantum information theory; indeed, it is the property of quantum states that most distinguishes them from the possible states of any classical system.

In order to extract information from a quantum system, an observer must perform a *measurement*. Measurement gives rise to *non–determinism,* in that the outcome is not predictable. For instance, electrons have a property known as *spin*; relative to an axis chosen by the observer, the value of spin may be 'up' or 'down' or, more generally, in a superposition of the two. A measurement would cause the electron's spin to collapse, at random, to either the 'up' or the 'down' state. If the electron in question is prepared in a known state, then the probability of either outcome can be computed, but the actual outcome is random. When one of a set of entangled particles is measured, the act of measurement influences the state of the other particles. Suppose there are two such particles; measuring the state of one alters and makes known the state of the other. This has also been confirmed by experiment.

Wootters and Zurek [151] showed that it is impossible to duplicate an unknown quantum state; this is known simply as the *no–cloning theorem*. This result is used in an essential way in quantum cryptographic protocols, as we will see in later sections.

Quantum entanglement is known to be maintained between particles even when they are physically separated; this is known as *non–locality*. Many quantum protocols rely on this property,

as measurements of entangled states, which may be miles apart, can be used to communicate information.

Quantum states are very fragile. As a particle interacts with its environment, its state tends to lose its stability and its wavefunction appears to collapse. This phenomenon is known as *decoherence*. In order to run a quantum protocol, or execute any quantum computation, it is necessary to combat decoherence. The phenomenon of decoherence is one of the main obstacles to building practical large–scale quantum computers today, but advances in experimental techniques and the development of quantum error correction help to overcome this particular problem.

The phenomena just described characterise a whole class of communication and cryptographic protocols with no direct analogue in classical computer science. A description of representative quantum protocols is given in Chapter 2.

1.1.2. Quantum Computation. Although quantum communication and cryptographic protocols are our focus in this thesis, we will now review certain aspects of quantum computation that are historically relevant.

A key feature of quantum mechanics is *reversibility*: it is always possible for a quantum system to evolve back to a previous state. Bennett [19] showed the existence of universal reversible Turing machines, thus proposing the first model of reversible computation. Subsequent work by Toffoli [144] and Fredkin and Toffoli [57] led to the discovery of universal classical reversible gates.

Benioff [17] proposed the first theoretical model of a quantum computer, showing it to be at least as powerful as a classical reversible computer; his proposal offered no computational speedup relative to a classical machine. Meanwhile Feynman [54] argued that, to simulate the behaviour of a general quantum mechanical system, a probabilistic Turing machine would require time exponential in the size of the system.

It was Deutsch [44] who proposed the first complete model of a quantum Turing machine, and conjectured that its computational power would be greater than that of a classical Turing machine for certain computations. Subsequently, Deutsch and Jozsa [45] demonstrated that there exist computational problems outside the complexity class **P** which can be solved with certainty in

polynomial time on quantum computers. Bernstein and Vazirani [25] developed quantum complexity theory, and conceived of the new complexity class **BQP** (bounded quantum polynomial time).

The discovery of efficient quantum algorithms for factoring integers and computing discrete logarithms by Shor [137] stimulated the interest of physicists and computer scientists alike, and gave the greatest impetus to the field of quantum computation. The best known classical factorisation algorithm for m–bit numbers has complexity $O(\exp(c(\ln m)^{1/3}(\ln\ln m)^{2/3})$, where $c = \left(\frac{64}{9}\right)^{1/3}$, according to [71]. Shor's quantum factorisation algorithm, on the other hand, requires $O(m^2 \log m \log\log m)$ steps on a quantum computer plus a polynomial number of steps on a classical computer. This is very significant, as some cryptographic systems such as RSA [132] rely on the belief that no polynomial time factorisation algorithm exists; a large scale quantum computer implementing Shor's algorithm, therefore, would be a threat to the security of such systems.

The code–breaking potential of quantum computers is, therefore, a cause for seeking cryptographic methods which are secure against such devices, and this is why studying the security of quantum key distribution and related protocols is so worthwhile. Quantum cryptography is designed to be secure even against quantum computers, as its security does not depend on the computational power of the enemy, but on fundamental quantum mechanical laws.

Grover [70] proposed an efficient quantum algorithm for unstructured database search. Classical solutions to this problem have an average time complexity of $O(\frac{N}{2})$ for a database with N elements, whereas Grover's algorithm takes $O(\sqrt{N})$ steps. The computational speedup in this case is not as substantial as with Shor's algorithm, but it demonstrates how quantum phenomena can be used to solve certain computational problems highly efficiently.

Quantum algorithms are designed to be implemented on a quantum computer, and such a device has yet to be built on a large scale. Experimental setups do exist, and there are several technologies that show some promise: ion traps, cavity quantum electrodynamics, and nuclear magnetic resonance stand out the most. These are discussed in all the standard texts [71, 111]. However, quantum cryptographic systems have been implemented with current–day technology

[49], and progress in experimental physics has led to experimental versions of several quantum protocols [31].

1.2. Motivation

Having reviewed the general setting of quantum computing and quantum information, we turn now to the issues and methods with which we are concerned.

It is a well–known fact that communication protocols are notoriously difficult to design. Holzmann [77] illustrates this by describing a scene from antiquity, taken from the writings of the Greek historian Polybius: it seems that even a simple communication protocol, involving the use of fire signals to relay messages during war, will be found to have fundamental flaws from which it is not possible to recover. This particular example was discussed in some detail in Papanikolaou [119].

Protocols are sets of rules that govern the way in which exchanges of data are performed in a communication system. The use of a protocol, as opposed to simple unstructured transmission of information from one party to another, is intended to prevent errors and to handle unexpected conditions, and, in the cryptographic setting, to satisfy certain security requirements for messages such as privacy, non–repudiation and more.

Formally, a protocol is characterised by five principal elements [77]:

(1) the service to be provided by the protocol,
(2) the assumptions about the environment in which the protocol is executed,
(3) the vocabulary of messages which are used to implement the protocol,
(4) the encoding of each message in the vocabulary, and
(5) the procedure rules guarding the consistency of message exchanges.

It is precisely the procedure rules mentioned in (5) above which are the most difficult to design; these rules give rise to a complex set of behaviours that must be modelled systematically and rigorously.

The procedure rules for a given protocol essentially define a sequence of interactions between concurrently executing processes; this concurrency is often the cause of subtle flaws to do with

timing, race conditions, deadlock and similar issues. According to Holzmann [77], unexpected events are the most common reason for a protocol failure, and by definition it is impossible for a system designer to account for such events in advance (although one can try to predict such events based on past experience). Also there are so many possible combinations of events that may occur during execution of a protocol that one cannot manually calculate them all (so as to ensure correct behaviour in each case).

Further, when a protocol is designed to provide a security–related service (we will use the terms "security protocol" and "cryptographic protocol" interchangeably) the issues just cited are compounded by the need to counter potential attacks. In fact, correcting flaws related to ordinary protocol behaviour may even have the effect of introducing new flaws that would enable an enemy to mount a successful attack. Also, even though a single protocol may be proven correct and secure, the way in which it is used in a particular system (esp. how it is combined with other protocols) may compromise its security.

The inevitable conclusion is that protocol design is a very difficult challenge, one which is beyond the capabilities of even the most skilled engineer. Proofs of correctness are sometimes carried out manually, by applying the mathematics relevant to the system at hand. The need for a systematic and rigorous analysis is apparent, yet such an analysis seems impractical, costly and very time–consuming unless some form of automation is used. Computer–based modelling and validation techniques, mentioned earlier, largely address this need and have been in widespread use for some years now. Section 1.3.1 reviews the field of automated verification, while Section 1.3.2 discusses formal modelling techniques.

1.3. Methodology

Methods for reasoning about protocols vary; we emphasise automated verification techniques here, as well as specification languages and logics. A survey of formal analysis techniques for communication protocols is given in Babich and Deotto [9].

1.3.1. Automated Verification. Automated verification techniques fall into two broad categories; *model checking* is fully automated, while *automated theorem proving* is partly automated.

Model checking [38, 130] is a method of computer–aided verification used widely in academia and industry for the analysis of software and communication systems. The method essentially consists of three phases: (i) modelling the system of interest at a suitable level of abstraction, usually by describing it in a formal specification language (see also section 1.3.2), (ii) specifying the properties that the system must satisfy — these properties are typically expressed with reference to the system model, using a logic designed specially for this purpose (*e.g.* linear or branching time temporal logic), and (iii) supplying the model and the properties to a model checking tool, which automatically constructs the state space of the model and checks whether or not the given properties are satisfied (and if not, usually providing a counter–example). Model checking involves exploring all possible behaviours arising from the model and, as such, it is an *exhaustive* method of analysis. The benefit of this is that all combinatorial possibilities are considered, and often particular scenarios are discovered which may not have been foreseen by a human user; this is especially true of larger models, which are too complex for analysis by hand.

Model checking software (*model checkers*) has been used to discover flaws in various industrial systems and protocols. For instance, the SPIN tool [78] was originally used to detect and correct signalling issues in telecommunications relays and switches at AT&T [77]. Similar tools have been used for the analysis of larger systems, ranging from flight software to CPUs, as discussed in the book by Huth and Ryan [80]. Further applications of model checking are found in the area of security, namely in the detection of subtle flaws in cryptographic protocols, relating to privacy and other security properties. For example, Lowe [93] used the FDR model checker [56, 133] to detect a flaw in the Needham–Schroeder PKCS protocol [110]; this is often heralded as one of the greatest successes of the technique, since it brought to the fore an issue with a protocol in widespread use.

Model checkers in widespread use include SMV [100], SPIN [78], UPPAAL [90]. The PRISM model checker [86, 121, 122] is designed for the analysis of probabilistic systems. The Formal Methods Wiki [32] contains an extensive database of verification tools, specification formalisms and related techniques, while the YAHODA database [87] focusses on tools, mainly model checkers.

Automated theorem proving tools such as Coq [143], HOL [66, 67], Isabelle [113] and PVS [117] are designed to assist a human user in the construction of formal proofs. For such tools to be used, a mathematical theory for a given problem must be sufficiently developed and suitably encoded in computer form. The correctness of a system, such as a particular protocol, is formulated as a theorem; the theorem has to be expressed in the logic of the automated theorem prover, and can then be proven in a stepwise manner. The main steps of the proof are guided by the user of the tool, while the tool itself applies certain proof rules automatically and does the necessary bookkeeping as the proof progresses. Automated theorem proving has substantially more generality than model checking, but it requires human intervention and the existence of several libraries of existing proofs and the development of specialised "theories" for the problem in question. A very notable application of automated theorem proving is Gonthier's proof [65] of the Four Colour Theorem using the tool Coq [143].

1.3.2. Specifying Systems. While verification techniques focus on automating proofs or parts of proofs of correctness for different systems, it is just as important to have adequate means of defining unambiguously, or *formally specifying,* system behaviour. For the description of protocols such formalisms are numerous, and usually model checking tools provide built–in specification languages. For instance, SPIN [78] uses the language PROMELA, while FDR uses CSP [76].

A large portion of the formal methods community is dedicated to the development of specification formalisms for different types of systems, taking the view that a suitable formalism can facilitate system design while also eliminating design flaws. Formalisms such as VDM [27] and Z [138] have been devised with this conviction in mind and have been used in several industrial applications.

Of most interest to us are *process algebras* [76, 105, 106, 128], which were originally designed for high level reasoning about multi–agent systems involving concurrency. Process algebras abstract away from certain features of programming languages, while providing various operators for expressing parallelism and communication.

CCS ("Calculus of Communicating Systems") [106] and CSP ("Concurrent Sequential Processes") [76] allow us to construct descriptions of system behaviour by providing an elegant, high level syntax for concurrent communicating processes. They differ slightly in syntax, and the latter has more operators. The π–calculus [105] was conceived as an extension to CCS which can be used to model mobility in processes and, as such, has more expressive power than its predecessor. For these formalisms, various notions of equivalence between processes have been defined, including bisimulation relations [120] of different strengths.

Equivalence relations between processes are particularly significant, since they may be used to prove that a given system satisfies its specification. Showing that a model of a system, expressed as a process M, satisfies a specification, expressed as another process S, amounts to proving the equivalence $P \cong S$, where $\cong$ is a suitably defined equivalence relation over the domain of all processes. Such a proof may be constructed automatically, and there are tools which implement such equivalence tests for processes (notably the *Concurrency Workbench of the New Century* [39], which handles the formalisms CCS, CSP, and LOTOS, among others).

Specialised extensions of process algebras have been proposed to allow for the description of particular classes of systems; for instance, probabilistic process algebras (see [81]) capture fault–tolerant systems and protocols, in which probabilities can be explicitly assigned to different events.

Since process algebras are particularly good in describing systems involving concurrency and communication, they have been used for the specification of both communication and cryptographic protocols since their inception. The process algebra LOTOS [30, 116], for example, was designed with the express intention of formalising communication protocols. Lowe's discovery of a flaw in the Needham–Schroeder protocol, cited earlier, involved expressing the protocol in the process algebra CSP.

1.3.3. Specifying System Properties. In the process algebraic approach to the specification of systems, the desired behaviour of a protocol is typically expressed as yet another process; in this case, process equivalences are used for verification. More commonly, system properties are described using formulae in a suitable logic, and in this case model checking algorithms are used.

An example of a setting in which a process algebra is used for system specification, but a logic is used for property specification, is provided by the framework of Hennessy–Milner logic (HML) [73]. HML is used to formalise properties of CCS processes.

Higher order logic is the logic used in the automated theorem prover HOL [66, 67], while most model checking tools use some form of modal logic, including temporal logics (linear [127] or branching time temporal logic [14]) and epistemic logics [52, 104].

1.4. Thesis Contribution

The goal in this thesis is to develop model checking algorithms and tools specifically for quantum protocols.

As discussed in Section 1.1, quantum protocols exhibit properties with no direct analogue in classical computer science, since they exploit phenomena such as superposition, entanglement, and probabilistic measurement, which are only manifested on the atomic scale.

On the other hand, we have seen in Section 1.2 that protocol design is inherently difficult in general, and that many methods have been devised to assist in the automated analysis of protocols. Model checking is a prominent method, which has the benefit of full automation.

The difficulties arising in designing correct communication and cryptographic protocols are likely to be exacerbated when a designer has to take into account behaviours caused by quantum phenomena. Today, while simulation tools for quantum information systems abound (see also section 1.6.4), to our knowledge no other authors have developed a tool aimed at verification, although the work of Sadrzadeh in connection with epistemic logic and Aximo [131] has some theoretical connections with this. It is our contention that a suitable modelling language and verification tool for quantum protocols is much needed, and will serve to address these difficulties.

We will present in this thesis a framework which includes:

- a modelling language (QMCLANG) for quantum protocols with a formal semantics and type system,

- an adaptation of a logic for describing properties of such protocols, interpreted over the semantics of the modelling language (our contribution lies in the adaptation of the logic to the needs of QMC and the redefinition of its semantics in terms of QMCLANG)
- an implementation of a model checker for a class of quantum protocols which are efficiently simulable on a classical computer, with
- a discussion of complexity issues, and how a classical model checker can be adapted to model such protocols, and
- a number of case studies, ranging from simple quantum protocols to a model of a larger system.

An important aspect of this work is our intention to build a software tool that is as efficient as possible; this justifies why have restricted the scope of our method to efficiently simulable protocols, namely those expressible within the stabilizer formalism.

1.5. Previous Work by the Author and Collaborators

As detailed below, in previous work we have considered the use of existing model checking tools for the analysis of quantum protocols, namely CWB–NC, SPIN and PRISM. Directly related to this was the design of the quantum process algebra "Communicating Quantum Processes (CQP)" by Gay and Nagarajan [61]. CQP was designed so as to provide a formal, high–level language for modelling quantum communicating systems, including protocols, and it possesses a formal semantics and type system.

Using Milner's CCS, Nagarajan and Gay [108] were able to develop a formal description of the BB84 protocol [20] and to verify a correctness property using the Concurrency Workbench of the New Century. This tool was used to prove the trace equivalence of the protocol with its specification.

Later, we chose to use the probabilistic model checker PRISM to develop and analyse models of protocols that explicitly accounted for the probabilism arising from quantum measurements. In [119], PRISM was used to show the correctness of part of BB84, while in [62] we described techniques for adapting PRISM to handle a specific set of quantum operations in a more direct

fashion. The development of QMC [63] was motivated by the large size of the resulting models (even for relatively small protocols) and the need to specify properties specific to quantum systems.

1.6. Related Work

1.6.1. Quantum Programming Languages and Quantum Process Calculi. Related to this work is the design of quantum programming languages, and there have been many proposals for such languages to date: QCL [114, 115], QPL [136], QGCL [135], the quantum λ–calculus of Van Tonder [147], the quantum process algebra QPAlg [82, 83] to name a few. See the survey by Gay [60] for an overview.

Lalire and Jorrand have developed their quantum process algebra by adding equivalence relations; see [88]. The group of Mingsheng Ying has developed qCCS, a quantum process algebra inspired by CCS (see [53, 152]).

The model of measurement–based quantum computation has given rise to the *measurement calculus* of Danos, Kashefi and Panangaden [43], and this formalism has been used to derive interesting properties of measurement–based circuits. A more recent extension of this work is described in [42].

1.6.2. Semantic Techniques. Abramsky and Coecke [3–5] have developed a category–theoretic formulation of the axioms of quantum mechanics. Their approach allows for a mathematical analysis of *information flow* in quantum protocols, especially that flow which is made possible through the use of entanglement, and they have demonstrated it with a correctness proof of the teleportation protocol (see Chapter 2 for a description).

Perdrix [124] has used abstract interpretation techniques to analyze entanglement properties of states, while Prost and Zerrari [129] use logical techniques for the same purpose.

1.6.3. Logics for Quantum Information. The increased interest in quantum computation and quantum information in recent years has made the subject of *quantum logic* very relevant today. Ever since research in quantum logic was initiated by Birkhoff and von Neumann [26], physicists have constantly been at odds over what its precise form should be. The emphasis was

mostly put on the *semantics* of quantum logic — in particular, on the mathematical structures that underlie quantum theory. A broad survey of quantum logic and logics for quantum information was given in Papanikolaou [118].

Mateus and Sernadas [96, 97] and Meyden and Patra [102, 103] (see also Patra [123]) have developed logics for reasoning about quantum information systems, including protocols. Their approaches are fundamentally different to the one of Birkhoff and von Neumann; both pairs of authors have designed quantum logics which are extensions of probabilistic logic.

Mateus and Sernadas have used the *exogenous* approach to design a logic for reasoning about quantum systems. This means that they have kept intact the classical model of propositional logic as the basis for their logic and simply augmented it to account for the probabilism inherent in quantum mechanics; in particular, the semantics of their logic is such that the denotation of a quantum proposition is given by *a superposition of the denotations of classical propositions.*

So, instead of building their logic atop the algebraic structures of quantum mechanics, Mateus and Sernadas have used models of propositional logic as their starting point. Their work is particularly inspired by the semantics of probabilistic logic, as given in [51, 112]. Their logic is named *exogenous quantum propositional logic* (EQPL). A more powerful variant, which allows reasoning about the dynamics of quantum systems, is DEQPL (*dynamic* EQPL).

More recently the same authors defined Quantum Computation Tree Logic (QCTL) [12, 98]. In [12] they present high–level model checking algorithms for QCTL and analyse their complexity (note that they do not mention any implementation; these algorithms were developed as this work was progressing and were published independently). They have done this for the most general possible setting, namely, for arbitrary quantum states which may arise in quantum algorithms and protocols.

Meyden and Patra [103] have focused on adapting the probabilistic logic in [51] to quantum systems, resulting in a logic for knowledge and time in quantum systems; they have separately proposed a logic for reasoning about the probabilities arising in quantum systems [102].

Other logics for the specification of quantum systems exist; D'Hondt and Panangaden [46] have proposed an epistemic logic for distributed quantum systems, and Danos and D'Hondt [41]

have proposed a temporal epistemic logic for reasoning about quantum protocols. Sadrzadeh [134] has considered the use of epistemic logics for the analysis of quantum cryptographic schemes. As mentioned in section 1.4, this last author has been involved in the development of a reasoning tool called Aximo [131].

1.6.4. Quantum Simulation Tools. Quantum simulation tools are intended for physicists and allow them to visualise the evolution of a quantum circuit. Existing simulation tools for quantum systems [64] are designed to help the user understand the function of a given quantum algorithm or protocol; some tools have a graphical user interface, and many allow the simulation of systems with arbitrary quantum gates, even if there is a substantial computational cost due to the limited power of the classical machine running the simulation. Simulators which allow only stabilizer operations include CHP [1] and GraphSim [7]. Quantum simulation tools are normally not efficient (the last two are an exception).

1.7. Outline

This thesis is organised as follows. Chapter 2 reviews background material and concepts which are used later. Chapter 3 describes a first approach to the verification of quantum protocols, which uses existing model checking tools; in particular, we describe techniques for adapting the probabilistic model checker PRISM to enable the analysis of small quantum protocols. This discussion leads to the observation that existing verification tools are fundamentally inadequate for the analysis of quantum systems, justifying a more specialised methodology. In Chapter 4 we describe a new specification language for quantum protocols and state its formal semantics; the logic QCTL is also described and related to the semantics of the specification language. The implementation of the QMC model checking tool is detailed in Chapter 5, and includes explanations of simulation and verification algorithms.

Examples of protocol analyses using QMC are provided in Chapter 6, and include quantum teleportation, quantum coin–flipping, a network involving quantum key distribution and quantum error correction, and quantum secret sharing. The work concludes with a final review and discussion, as well as directions for future work in Chapter 7.

CHAPTER

2

BACKGROUND

Bell's theorem ... proves that quantum theory requires connections that appear to resemble telepathic communication.

— Gary Zukav

In this chapter we will review those concepts and notations which are required for a proper understanding of the treatment that follows. We will focus on the mathematical framework of quantum mechanics, giving relevant definitions; our presentation aims to be as self–contained as possible. We will introduce Dirac notation for quantum states and discuss measurement and evolution of a quantum system. We will give a detailed account of several quantum protocols of interest, namely: superdense coding, quantum teleportation, a quantum error–correcting code, conjugate coding, quantum key distribution, and quantum coin–flipping. The stabilizer formalism will be introduced, along with the attendant algebraic ideas.

2.1. The Postulates of Quantum Mechanics

The main principles of quantum mechanics can be summarised in a few basic postulates, which are stated below.

Postulate 1. *All possible information about an isolated physical system can be obtained from a* ***state vector,*** *which is an element of an abstract vector space $\mathscr{H}$. We denote the elements of $\mathscr{H}$ using the notation $|\psi\rangle$, where ψ is some label.*

Postulate 2. *The state space of a composite quantum system is defined as the* ***tensor product*** *of the state spaces of its components.*

Postulate 3. *Physically measurable quantities are represented by Hermitian operators that act on state vectors in $\mathscr{H}$ and are known as* ***observables****. A linear operator A is* ***Hermitian*** *if it is equal to its adjoint, i.e. if $A^\dagger = (A^T)^* = A$.*

Postulate 4. *If an operator A has eigenvalues λ_i and eigenvectors $|a_i\rangle$, i.e. if $A|a_i\rangle = \lambda_i|a_i\rangle$, then the probability of obtaining measurement result λ_i when measuring a state $|\psi\rangle$ is given by $|c_i|^2$, where the c_i are the coefficients obtained when expanding $|\psi\rangle$ in terms of the eigenvectors $|a_i\rangle$, so that:*

$$|\psi\rangle = c_1|a_1\rangle + c_2|a_2\rangle + \cdots + c_n|a_n\rangle$$

In other words, the possible results of a measurement of a physical quantity are the eigenvalues of the corresponding observable.

Postulate 5. *The state of a system after a projective measurement yielding result λ_i is described by the basis vector $|a_i\rangle$ corresponding to the coefficient c_i.*

Postulate 6. *The time evolution of a quantum system is governed by the Schrödinger equation,*

$$i\hbar\frac{\partial}{\partial t}|\psi\rangle = H|\psi(t_0)\rangle$$

where H is the so–called Hamiltonian (which is specific to the system being studied) and t_0 is some initial time. The solution to this equation gives the wavefunction $|\psi(t)\rangle$, which is the quantum state of the system as a function of time.

The postulates dictate the basic rules of the theory and link together the concepts of state space, observable, measurement and evolution. We now discuss these concepts in more detail.

2.2. Hilbert Space

In quantum mechanics the possible states of a system (the term 'system' in this context is normally taken to refer to a particle or a closed collection of particles) are described by vectors belonging to a complex–valued, complete, vector space equipped with an inner product. Such a space is referred to as a *Hilbert space*. A complex vector space V is a set which contains a zero element 0, a unit element 1, and, for all vectors $|\psi\rangle, |\phi\rangle, |\chi\rangle \in V$ and for all $\alpha, \beta \in \mathbb{C}$ satisfies the following:

$$|\psi\rangle + |\phi\rangle = |\phi\rangle + |\psi\rangle \tag{2.1}$$

$$(|\psi\rangle + |\phi\rangle) + |\chi\rangle = |\psi\rangle + (|\phi\rangle + |\chi\rangle) \tag{2.2}$$

$$0 + |\psi\rangle = |\psi\rangle \tag{2.3}$$

$$|\psi\rangle - |\psi\rangle = 0 \tag{2.4}$$

$$\alpha(\beta|\psi\rangle) = (\alpha\beta)|\psi\rangle \tag{2.5}$$

$$(\alpha + \beta)|\psi\rangle = \alpha|\psi\rangle + \beta|\psi\rangle \tag{2.6}$$

$$\alpha(|\psi\rangle + |\phi\rangle) = \alpha|\psi\rangle + \alpha|\phi\rangle \tag{2.7}$$

$$1\,|\psi\rangle = |\psi\rangle \tag{2.8}$$

A Hilbert space is equipped with an *inner product* $\langle\psi|\phi\rangle$, which satisfies:

$$\langle\phi|\psi\rangle = \langle\psi|\phi\rangle^* \tag{2.9}$$

$$\langle\psi|\psi\rangle \geqslant 0 \tag{2.10}$$

$$(\alpha\langle\psi| + \beta\langle\psi|)|\chi\rangle = \alpha\langle\psi|\chi\rangle + \beta\langle\psi|\chi\rangle \tag{2.11}$$

Note that the symbol $\langle\psi|$ (known as a 'bra' in the literature) is defined as the complex conjugate transpose of $|\psi\rangle$. If we take the convention that a state is written as a column vector, then a bra is a row vector, so that their inner product is just a complex number.

Given a Hilbert space $\mathscr{H}$, we can obtain a *spanning set* $\{|\phi_i\rangle\}$, such that each $|\psi\rangle \in \mathscr{H}$ can be written as a linear combination of the $|\phi_i\rangle$:

$$|\psi\rangle = \sum_i \alpha_i|\phi_i\rangle$$

Definition 2.1 (Linear Independence)**.** *A set of n vectors $\{|\psi_i\rangle\}_{i=1}^n$ is **linearly independent** if, for any set of n nonzero complex numbers $\{\alpha_i\}_{i=1}^n$,*

$$\sum_i \alpha_i|\psi_i\rangle = 0$$

Definition 2.2 (Orthogonality)**.** *Two vectors $|\psi\rangle$ and $|\phi\rangle$ are **orthogonal** if $\langle\psi|\phi\rangle = 0$. A set of vectors is orthogonal if every vector is orthogonal to every other vector.*

Definition 2.3 (Basis)**.** *If $\mathscr{H}$ is a Hilbert space and S is a set of linearly independent vectors which span $\mathscr{H}$, then S is called a **basis** for $\mathscr{H}$.*

Definition 2.4 (Dimension)**.** *All bases for a Hilbert space $\mathscr{H}$ have the same size n, which is referred to as the **dimension** of $\mathscr{H}$.*

We conventionally label the basis vectors of an n–dimensional space using the integers 0 to $n-1$, so that $\{|0\rangle, |1\rangle\}$ is a basis for $\mathscr{H}_2$. Thus a vector $|\psi\rangle$ in an n–dimensional space may be expressed using the expansion:

$$|\psi\rangle = \sum_{i=0}^{n} c_i|i\rangle = c_0|0\rangle + \cdots + c_{n-1}|n-1\rangle \tag{2.12}$$

A two–dimensional Hilbert space (denoted $\mathscr{H}_2$) corresponds to the state space of a quantum system with two degrees of freedom, known as a *qubit*, or quantum bit. Such is the state space of a spin–$\frac{1}{2}$ electron, for example, or a hydrogen atom (which has two basis states, a ground state and an excited state). We can write the general state of a qubit as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad \text{where } \alpha, \beta \in \mathbb{C} \tag{2.13}$$

Systems of many quantum states have a state space which is the *tensor product* of the individual state spaces. Two quantum states $|\psi\rangle = \sum_i c_i|i\rangle$ and $|\phi\rangle = \sum_j d_j|j\rangle$ can be combined to form a joint state by taking the tensor product $\otimes$:

$$|\psi\rangle \otimes |\phi\rangle = |\psi, \phi\rangle = |\psi\phi\rangle = \sum_{i,j} c_i d_j \; |i\rangle \otimes |j\rangle$$

For an n–qubit system, we have a state space which is the tensor product of n copies of $\mathscr{H}_2$:

$$\mathscr{H}_n = \underbrace{\mathscr{H}_2 \otimes \cdots \otimes \mathscr{H}_2}_{n}$$

The state space $\mathscr{H}_n$ is spanned by 2^n basis vectors, so that the general state of an n–qubit system can be written

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i \; |b(i)\rangle$$

where $b(i)$ represents the integer i in binary notation. For example, the general state of a two–qubit system can be written:

$$|\psi\rangle = c_0|00\rangle + c_1|01\rangle + c_2|10\rangle + c_3|11\rangle \qquad \text{where } c_0, c_1, c_2, c_3 \in \mathbb{C}$$

Note that a system of two or more qubits may be in an *entangled state;* when this occurs the system as a whole has a known state, which cannot be decomposed, or separated, into individual

states for each of the qubits. For example, while the state

$$\frac{1}{\sqrt{2}}|00\rangle + |01\rangle \text{ can be rewritten as } \frac{1}{\sqrt{2}}|0\rangle \otimes (|0\rangle + |1\rangle)$$

it is not possible to do the same for the entangled state of two qubits below:

$$\frac{1}{\sqrt{2}}|10\rangle + |01\rangle$$

Finally, note that we conventionally take quantum state vectors in their *normalised* form, so that:

$$\text{for a state } |\psi\rangle = c_0|0\rangle + \cdots + c_n|n\rangle \text{ we have } \sum_i |c_i|^2 = 1$$

This justifies the normalisation coefficient of $\frac{1}{\sqrt{2}}$ which appears in the above examples.

2.3. Operators and Matrices

Transformations of quantum states are expressed mathematically by linear operators acting in the Hilbert space under consideration. An operator $T : \mathscr{H}_n \to \mathscr{H}_n$ is linear if, given $\alpha, \beta \in \mathbb{C}$ and vectors $|u\rangle, |v\rangle \in \mathscr{H}_n$, it satisfies:

$$T(\alpha|u\rangle + \beta|v\rangle) = \alpha T|u\rangle + \beta T|v\rangle$$

An operator has a matrix representation with respect to a set of basis vectors $\{|u_i\rangle\}_{i=0}^{2^n-1}$ whose elements $T_{i,j}$ are given by

$$T_{i,j} = \langle u_i| T |u_j\rangle$$

We define two significant kinds of operator:

Definition 2.5 (Hermitian operator)**.** *An operator T is **Hermitian** if is equal to its adjoint, i.e. if $T^\dagger = T$ (similarly for the matrix representation of T, where the adjoint is obtained by transposing and then taking the complex conjugate of all elements).*

Definition 2.6 (Unitary operator)**.** *A matrix or operator U is **unitary** if:*

$$UU^\dagger = U^\dagger U = I \qquad \text{where } I \text{ is the identity matrix}$$

Hermitian operators correspond to observables, namely measurable quantities. Hermitian matrices have real eigenvalues, and as per the third postulate of quantum mechanics, these represent the possible outcomes of an actual measurement.

Unitary operators describe a way in which a quantum system can evolve. To obtain the state of a system once such evolution has occurred, one applies the appropriate unitary operator to the state vector. Thus, if U describes a particular type of evolution, then a system in the quantum state $|\psi\rangle$ can evolve to the state $|\phi\rangle = U|\psi\rangle$.

2.4. Projective Measurements

The product of a ket $|\psi\rangle$ and a bra $\langle\phi|$ (in that order) is known as an *outer product* and it defines a *projection operator* $|\psi\rangle\langle\phi|$. A projection operator maps any state in the Hilbert space $\mathscr{H}_n$ to a state belonging to a subspace $\mathscr{H}_m$, where $m \leqslant n$. If this subspace is spanned by a set of basis vectors $|u_1\rangle, \ldots, |u_m\rangle$, then the projection operator P_m corresponding to this subspace is given by

$$P_m = \sum_{i=1}^{m} |u_i\rangle\langle u_i|$$

A projection operator is always Hermitian and describes the most common type of measurement arising in quantum mechanics (only such measurements concern us here).

The state of a system (which is initially in state $|\psi\rangle$) after a projective measurement P_j is given by

$$\frac{1}{\sqrt{\langle\psi| P_j|\psi\rangle}} \cdot P_j|\psi\rangle$$

The possible outcomes of a quantum measurement are associated with given probabilities as prescribed by Postulate 4.

2.5. Quantum Gates, Circuits and Approximate Universality

In quantum information science, and specifically in the quantum protocols of interest in this thesis, the specific unitary operators (known as *quantum gates* by analogy to the gates arising in boolean circuits) shown in Fig. 2.1 arise.

Operator	Symbol	Matrix Representation	Effect
Hadamard	H	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$	$H\|0\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$, $H\|1\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$
CNot	C	$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$	$C\|0,x\rangle = \|0,x\rangle$, $C\|1,x\rangle = \|1,1-x\rangle$, $x \in \{0,1\}$
Phase	S	$\begin{bmatrix}1 & 0\\ 0 & i\end{bmatrix}$	$S\|0\rangle = \|0\rangle$, $S\|1\rangle = i\|1\rangle$
σ_X	X	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$	$X\|0\rangle = \|1\rangle$, $X\|1\rangle = \|0\rangle$
σ_Y	Y	$\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}$	$Y\|0\rangle = -\|1\rangle$, $Y\|1\rangle = \|0\rangle$
σ_Z	Z	$\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$	$Z\|0\rangle = \|0\rangle$, $Z\|1\rangle = -\|1\rangle$

FIGURE 2.1. The principal quantum gates arising in protocols.

We can create versions of these operators suitable for higher–dimensional state spaces by taking the tensor product with the identity operator, so that, for instance, $I \otimes H \otimes I$ represents the Hadamard gate on the second qubit of a three–qubit system (which has a state space $\mathscr{H}_3$, spanned by $2^3 = 8$ basis vectors). Note that the CNot gate is a two–qubit gate, which has the effect of 'flipping' the state of the second qubit if the first qubit is in state $|1\rangle$.

Quantum gates can be depicted graphically so as to form *circuits;* we will make use of this graphical notation later in the chapter when describing several protocols. In a quantum circuit we represent a gate by a box containing the gate symbol, —$\boxed{H}$—, a measurement with respect to the computational basis $\{|0\rangle, |1\rangle\}^{\otimes n}$ with a meter symbol, —$\boxed{\nearrow}$—, and these will be linked by horizontal lines representing 'quantum wires.' The output of a measurement, which is a classical value, is represented by a double line. Examples of quantum circuits appear in Figs. 2.3,2.4, and 2.5.

A set of quantum gates is said to be *universal* if any quantum computation can be expressed in terms only of those gates. We define *approximate universality* below based on Gruska [71]:

Definition 2.7 (Approximate Universality)**.** *A set of quantum gates is* approximately universal *if any unitary transformation U on any qubit register (i.e. joint qubit state) can be performed, with arbitrary precision $\epsilon > 0$, by a quantum circuit $C_{U,\epsilon}$ consisting of the gates from that set.*

Theorem 2.1. *The set of quantum gates $\mathscr{G} = \{H, C, X, Y, Z, \frac{\pi}{8}\}$ is approximately universal, where $\frac{\pi}{8}$ denotes the operator represented by the matrix*

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

in the computational basis.

As we shall see in Section 3.3, we will be concerned with protocols involving a subset of $\mathscr{G}$, which have the property of being efficiently simulable on a classical computer. Of course, only a quantum computer can perform an arbitrary quantum computation efficiently, and hence simulate a quantum circuit involving a universal set of gates (see [111]).

2.5.1. Review of Fundamental Group–Theoretic Concepts. We now review basic ideas from group theory [79], including the definition of a group, and the concept of group action.

Definition 2.8. *A **group** is a set G equipped with an associative binary operation $\star$ in G and an identity element e, and is such that all elements have an inverse. Formally:*

$$\begin{aligned} \forall x, y, z \in G \quad &: \quad x \star (y \star z) = (x \star y) \star z \\ \exists e \in G \quad &: \quad x \star e = e \star x = x \\ \forall x \in G . \exists x^{-1} \in G \quad &: \quad x \star x^{-1} = x^{-1} \star x = e \end{aligned}$$

Definition 2.9. *A group $(G, \star)$ is commutative or **abelian** if $\forall x, y \in G : x \star y = y \star x$.*

Definition 2.10. *A **subgroup** of a group $(G, \star)$ is a subset of G equipped with the operation $\star$ which is also a group.*

In the following, let A be a subset of a group G.

Definition 2.11. *The **centralizer** $C(A)$ of A in G is defined by*

$$C(A) = \{c | c \in G \text{ and for all } a \in A,\ c \star a = a \star c\} \tag{2.14}$$

Definition 2.12. *The **normalizer** $N(A)$ of A in G is defined by*

$$N(A) = \{n | n \in G \text{ and } nAn^{-1} = A\} \tag{2.15}$$

Definition 2.13. *The **action** of a group G on a set X is a morphism $\phi : G \times X \to X$, $(g, x) \mapsto g \cdot x$, which satisfies:*

$$\forall g, g' \in G. \forall x \in X : \quad g' \cdot (g \cdot x) = (g' \cdot g) \cdot x$$

$$\forall x \in X : \quad e \cdot x = x$$

where e is the identity element.

Definition 2.14. *Given a set X and a group G, we define:*

- *the **orbit** of $x \in X$ as $Gx = \{y \in X, \exists g \in G : y = g \cdot x\}$*
- *the **stabilizer** of $x \in X$ as $G_x = \{g \in G : g \cdot x = x\}$*

2.6. The Stabilizer Formalism

The stabilizer formalism arises when we start to identify quantum states by the unitary operators which fix, or stabilize, them. For example, we can uniquely identify the EPR state

$$\left|\Psi^+\right\rangle = \frac{1}{\sqrt{2}} \left(|00\rangle + |11\rangle\right) \tag{2.16}$$

by the set of operators $\{X \otimes X, Z \otimes Z\}$. This set of operators is *unique* [111] for the quantum state $|\Psi^+\rangle$; we can also regard $|\Psi^+\rangle$ as the +1-eigenvector of these operators, since

$$\begin{aligned} (X \otimes X) \left|\Psi^+\right\rangle &= \left|\Psi^+\right\rangle \\ (Z \otimes Z) \left|\Psi^+\right\rangle &= \left|\Psi^+\right\rangle \end{aligned}$$

We can characterise not only specific states, but entire subspaces of the Hilbert space $\mathscr{H}_n$ by specifying the set of operators which stabilize elements of those subspaces. This has proved to be very useful in the design of quantum error–correcting codes [10, 68, 69, 111]. In order for quantum error correction to succeed, errors have to be easily distinguishable (the formal criterion for distinguishability of codewords is given in the works previously cited). Using the concept of stabilizer operators, it is possible to systematically construct good quantum error correction codes. In particular, one can design codes such that a single error will map $+1$-eigenvectors of a chosen operator to -1-eigenvectors. This is detailed in the notes by Bacon [10].

The unitary operators which characterise uniquely a subspace of $\mathscr{H}_n$ are elements of the *Pauli group* of dimension n, denoted $\mathscr{P}_n$. The Pauli group of dimension 1 is the group formed by the Pauli operators $\{X, Y, Z\}$ and the identity operator I, along with constant multiplicative factors of ± 1 and $\pm i$, under the operation of matrix multiplication. The multiplication table, or *Cayley table,* for this group $\mathscr{P}_1$ is partially shown below.

$\times$	I	X	Y	Z
I	I	X	Y	Z
X	X	I	iZ	$-iY$
Y	Y	$-iZ$	I	iX
Z	Z	iY	$-iX$	I

The Pauli group of dimension n is formally defined using the tensor product:

$$\mathscr{P}_n = \left\{ \bigotimes_{j=1}^{n} A_j \text{ where } A_j \in \mathscr{P}_1, j = 1, 2, \ldots n \right\} \tag{2.17}$$

A subgroup of $\mathscr{P}_n$ which stabilizes the states in a certain subspace of $\mathscr{H}_n$ (this being known as the *stabilizer subspace*) is a *stabilizer group*[1]. We can uniquely characterise a stabilizer group of interest by its generators or its *generating set*. The generating set of any group is defined as a set whose elements can be combined under the group operation to form the elements of the full group. It is a well-known fact of group theory that a finite group $(G, \star)$ has a generating set of length at most $\log_2 |G|$ where $|G|$ denotes the number of elements in G; the stabilizer group of

[1] In this thesis, we will consistently use the American spellings of the verb *stabilize* and the noun *stabilizer*, especially because the latter is a scientific term which we use in a specific sense.

an n–qubit state will thus have a generating set of length $\log_2 2^n = n$. Therefore, only n Pauli operators are needed to uniquely specify the stabilizer group of a particular state.

The most important aspect of the stabilizer formalism is the way in which certain unitary operators (namely the CNot gate, the Hadamard gate, and the phase gate) act on stabilizer groups. In particular, these operators are a *generating set* of the *Clifford group* in $\mathscr{H}_n$. The Clifford group of quantum gates (note the abuse of terminology here; the term "Clifford group" is used in a much more abstract way in mathematics texts, but we are using the term as it applies in the context of the stabilizer formalism) is the *normaliser* of the Pauli group. Thus, if S is a Clifford operator (*i.e.* a member of the Clifford group), and P is any Pauli operator,

$$SPS^\dagger = P' \tag{2.18}$$

where P' is also a Pauli operator and $S^\dagger$ denotes the adjoint of S (see also Definition 2.5). The effect of these Clifford group gates (under *Hermitian conjugation* as expressed in Eq. (2.18), with S corresponding to the operation, P to the input, and P' to the output) on the Pauli operators is shown below.

Operation	**Input**	**Output**
CNot	$X \otimes I$	$X \otimes X$
	$I \otimes X$	$I \otimes X$
	$Z \otimes I$	$Z \otimes I$
	$I \otimes Z$	$Z \otimes Z$
H	X	Z
	Z	X
S	X	Y
	Z	Z
X	X	X
	Z	$-Z$
Y	X	$-X$
	Z	$-Z$
Z	X	$-X$
	Z	Z

FIGURE 2.2. The effect of the Clifford group gates on the Pauli gates under Hermitian conjugation.

If one confines attention to stabilizer states and Clifford group operations, then they are working within the scope of the stabilizer formalism. Further details on operations, as well as measurements within the stabilizer formalism, are to be found in the standard references [10, 68, 69, 111].

2.7. Quantum Protocols

Quantum protocols can be succinctly described using the quantum circuit model [71, 111]. The general form of a quantum protocol involves a sequence of quantum transformations and measurements on an initial state $|\psi_0\rangle$. Communication in the quantum realm typically may refer to interactions that are caused by measuring entangled states, rather than physical transfers of data from one location to another. This is why quantum communication protocols can be described using circuits, whereas this is not possible for classical protocols.

In a typical quantum protocol, the different parties involved share an entangled state. In other words, each party possesses one of several particles which are entangled. The act of measuring part of an entangled state can be regarded as a transmission of information, since the effect of the measurement is to determine the other components of the state in a unique way. This is exemplified by the quantum teleportation protocol [23].

There are some quantum protocols which do not employ the phenomenon of entanglement at all. The BB84 protocol [20] for quantum key distribution is an example; it actually involves the physical transmission of qubit states over a quantum channel. This is also considered a form of quantum communication.

2.7.1. Superdense Coding. The simplest quantum protocol which we will use to illustrate our techniques is the superdense coding scheme [22]. This scheme makes it possible to encode a pair of classical bits on a single qubit. With superdense coding, a quantum channel with a capacity of a single qubit is all that is necessary to transmit twice as many bits as a serial classical channel.

The setting for superdense coding involves two parties, conventionally named Alice and Bob, who are linked by a quantum channel and share a pair of entangled qubits. The objective is for

Alice to communicate the binary number xy — henceforth termed the *message* and denoted by (x, y), with $x, y \in \{0, 1\}$ — by transmitting a single qubit to Bob. The superdense protocol takes advantage of the correlations between qubits P_1 and P_2, which are in an entangled quantum state. Alice essentially influences this state in such a way that Bob's measurement outcome matches the message of her choice. The quantum circuit diagram for the superdense coding procedure is shown in Fig. 2.3.

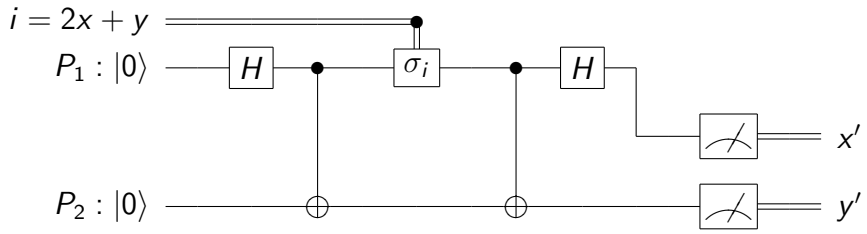


FIGURE 2.3. Quantum circuit diagram for the superdense coding protocol.

Quantum circuit diagrams are a convenient means for expressing computations on qubits; while these are mostly self–explanatory, the reader is referred to standard texts [71, 111] for explanations of the notation. For clarity, we describe superdense coding in full below.

(1) Two qubits, P_1 and P_2, are placed in an entangled state using the Hadamard and CNot operations. Alice is given P_1, and Bob is given P_2.

(2) Alice selects a message, (x, y), and applies the ith Pauli operator, σ_i, to P_1, where $i = y + x(2 + (-1)^y)$. She transmits this particle to Bob.

(3) Bob applies the CNot gate from P_1 to P_2, and then he applies the Hadamard gate to the former.

(4) Bob measures the two particles, thus obtaining a pair of classical bits, (x', y'). If no disturbance has occurred, this pair of bits will match the original message, *i.e.* $(x', y') = (x, y)$.

2.7.2. Quantum Teleportation. The quantum teleportation protocol [23] involves a series of operations on a three–qubit system. The quantum circuit for teleportation is shown in Fig. 2.4.

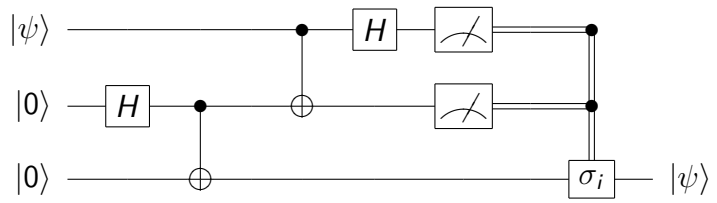


FIGURE 2.4. Quantum circuit diagram for the teleportation protocol.

The purpose of the protocol is to allow the transmission of qubit $|\psi\rangle$ from one user ('Alice') to another ('Bob') using only a classical communications line. Alice possesses a qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ to start with. An entangled pair of particles is created using an EPR source, and one particle (*qubit 2*) is given to Alice, the other (*qubit 3*) to Bob (creation of the EPR pair $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ can be performed by applying the Hadamard (H) and CNot operations to an initial two-particle state $|00\rangle$, represented by the second and third wire in the diagram of Fig. 2.4). Then the protocol proceeds as follows:

(1) Alice performs the CNot operation with qubit $|\psi\rangle$ as control and qubit 2 as target.

(2) Alice performs the Hadamard (H) operation on qubit 2.

After steps 1 and 2, qubit 2 is entangled with qubit $|\psi\rangle$.

(3) Alice measures qubit $|\psi\rangle$ in the standard basis and records the outcome, c_1.

(4) Alice measures qubit 2 in the standard basis and records the outcome, c_2.

(5) Alice transmits the values of c_1 and c_2 over a classical communication channel.

(6) Bob applies the Pauli operator σ_c to qubit 3, where $c = c_2 + c_1(2 + (-1)^{c_2})$, with the convention that $\sigma_0 \equiv I, \sigma_1 \equiv X, \sigma_2 \equiv Y, \sigma_3 \equiv Z$

Bob's qubit, *i.e.* qubit 3, is now in the state in which $|\psi\rangle$ was originally.

2.7.3. Quantum Error Correction. Our third example is the quantum bit–flip code for error correction [139]. In order to correct a single bit flip error, which may occur during the transmission of a single qubit state, this code represents the state by using a collection of three qubits. In particular, the qubit state $|0\rangle$ is encoded as $|000\rangle$ and the state $|1\rangle$ is encoded as $|111\rangle$. A bit flip error on the second qubit, for example, transforms $|000\rangle$ into $|010\rangle$.

In order to detect such an error, two additional qubits are used; they are known as *ancillas*. By applying a sequence of operations and measurements to the ancillas, the so–called *error syndrome* is obtained, which determines the location of the error. Then, the σ_1 operator is applied to the erroneous qubit, thus restoring the initial quantum state of the 3–qubit system. The quantum circuit for the bit–flip code is given in Fig. 2.5. In the figure, $\sigma_{1,i}$ denotes the operator corresponding to the application of σ_1 to the ith qubit.

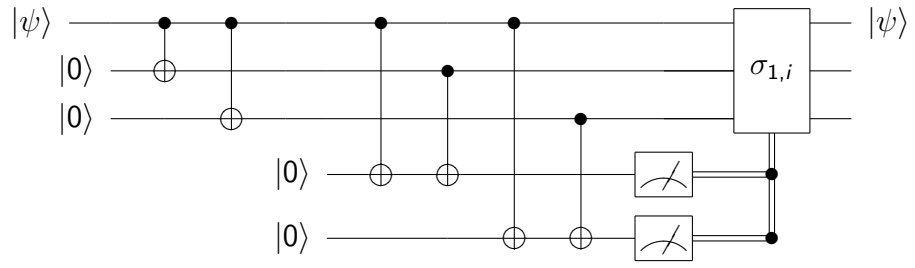


FIGURE 2.5. Quantum circuit diagram for the qubit bit–flip code.

For the diagram we have assumed that a bit–flip error does occur prior to the computation of the syndrome.

2.7.4. Conjugate Coding. In Wiesner [148] a scheme is proposed that allows a user to transmit two messages, either but not both of which may be received by another user; this task is known as *oblivious transfer* [33]. Let A denote the first message, and B the second message, where $A \in \{0,1\}^m$ and $B \in \{0,1\}^n$. The bits in each message are encoded into the polarisations of a stream of photons, which are transmitted over a quantum channel, such as a fibre optic cable.

Transmission proceeds as follows. At each time step, a coin is flipped, and depending on the outcome, a photon is prepared so as to represent a bit from one of the two sequences. If the outcome is heads, the ith bit in A is mapped to a photon with a *rectilinear* polarisation of $0°$ or $90°$. If the outcome is tails, the ith bit in B is mapped to a photon with a *circular* polarisation of $45°$ or $90°$. In the latter case, the polarisation can be expressed as a superposition of the two rectilinear polarisations (this will become clearer when the mathematical formalism is introduced in the next chapter).

The receiver uses an analyser (in practice, typically a birefringent crystal) to measure the bit sequence from the photon stream. The analyser is configured to separate the components of the stream into two different beams (corresponding to bit values of 0 and 1, respectively), either with respect to the rectilinear polarisation, or with respect to the circular polarisation. It is not possible to measure the rectilinear and circular component of polarisation simultaneously, and the process of measuring one component will collapse the state of the photons so that all information about the other component is lost, as dictated by the laws of quantum mechanics. Measuring a photon that is polarised horizontally or vertically with an analyser that is configured for circular polarisations will yield a random outcome.

If the transmitter encodes and sends all the bits in A and B, and the receiver measures each photon received individually with a fixed analyser configuration, then all the bits in one of the two sequences will be received correctly, while those of the other sequence will only be partly correct (*i.e.* when the measurement happens to produce an outcome that matches the original bit). Thus only one of the two messages is correctly transmitted.

Wiesner pointed out that this *conjugate coding* scheme is only practical if the receiver performs simple, individual measurements on the photons. There exist complex, collective measurements that would enable a receiver to correctly decode both bit sequences, although they are not easily implementable in hardware.

2.7.5. Quantum Key Distribution using BB84. Bennett and Brassard [20] proposed a protocol for establishing a secret random bit string between two users, with the intention that such a string could be used as a cryptographic key. Their protocol, referred to simply as BB84, involves a variant of conjugate coding. The use of quantum channels, which cannot be tapped or monitored without causing a noticeable disturbance, makes it possible to achieve unconditionally secure key establishment between two users. The presence of an enemy is made manifest to the users of such channels through an unusually high error rate.

BB84 assumes that the two legitimate users are linked by two specific channels, which the enemy also has access to a classical, possibly public channel (which may be passively monitored

but not tampered with by the enemy) and a quantum channel (which may be tampered with by an enemy). By its very nature, this channel prevents passive monitoring.

The first phase of BB84 involves transmissions over the quantum channel, while the second phase takes place over the classical channel. The steps in the protocol are the following:

(1) Alice generates a random binary sequence s.

(2) Alice chooses which type of photon to use (rectilinearly polarised or circularly polarised) in order to represent each bit in s. Let b denote the sequence of choices of basis for each photon.

(3) Alice uses an analyser to create a stream p of polarised photons whose polarisations represent the bits in s. Alice sends the photon sequence p to Bob over a suitable quantum channel, such as an optical fibre.

(4) For each photon received, Bob makes a guess as to whether it is rectilinearly or circularly polarised, and sets up his measurement device accordingly. Let b' denote his choices of basis.

(5) Bob measures each photon with respect to the basis he has chosen, producing a new sequence of bits s'.

(6) Alice and Bob communicate over a classical, possibly public channel. Specifically, Alice tells Bob her choice of basis for each bit, and he tells her whether he made the same choice. The bits for which Alice and Bob have used different bases are discarded from s and s'.

What is important to understand about this protocol is that, only if Bob's guess is correct is it certain that he will make an accurate measurement. If Bob attempts to measure a rectilinearly polarised photon with a circularly oriented measurement device (and vice versa), the outcome will be, at random, either 0 or 1; the original bit value represented by the photon is encoded in its rectilinear polarisation, all information about which is lost. So, an incorrect choice of measurement basis randomises the outcome of a measurement, which is only accurate in this case with probability $\frac{1}{2}$. If n photons are transmitted in total, there is a probability $\left(\frac{1}{2}\right)^n$ that Bob will measure all of them correctly.

A similar logical argument allows Alice and Bob to detect the presence of an eavesdropper, Eve. Just as Bob, Eve is incapable of knowing which type of photon is used to represent each bit. Therefore Eve must guess which measurement basis to use and, since it is impossible for her to duplicate the state of each received photon (due to the no–cloning theorem), she must prepare a new photon to send to Bob. Eve's presence is made manifest to Alice and Bob because her measurements necessarily cause a disturbance to the states of the transmitted photons.

The criterion for detecting Eve's presence can be formulated as follows. For the ith bit chosen by Alice, s_i, there will correspond a choice of polarisation basis, b_i, which is used to encode the bit to a photon. If Bob's chosen measurement basis is b_i' and the outcome of his measurement is s_i', then $b_i' = b_i$ should necessarily imply $s_i' = s_i$. If an eavesdropper tries to obtain any information about s_i, a disturbance will result and, even if Bob and Alice's bases match, $s_i' \neq s_i$. This allows Alice and Bob to detect an eavesdropper's presence on a noiseless channel, and to reschedule their communications accordingly.

The basic BB84 procedure is incomplete in the following sense: whether an eavesdropper is present or not, there will still be errors in Bob's key sequence. The final step of BB84, which was described above merely as a comparison of encoding and measurement bases, is usually much more elaborate. There are two parts involved: *secret key reconciliation* and *privacy amplification*.

Secret key reconciliation [35] is a special error correction procedure which eliminates:

- errors due to incorrect choices of measurement basis,
- errors induced by eavesdropping, and
- errors due to channel noise, if any exists.

Reconciliation is performed as an interactive binary search for errors. Alice and Bob divide their bit sequences into blocks and compare each other's parity for each block. Whenever their respective parities for any given block do not match, they divide it into smaller blocks and compare parities again, repeating this process until the exact location of the error is found. When an error has been located, Alice and Bob may decide to discard the corresponding bit, or agree on the correct value. During this process, Alice and Bob can communicate over a classical channel, which is by definition insecure and accessible to an eavesdropper.

Since valuable information about the key may be obtained by an eavesdropper during reconciliation, Alice and Bob must perform a final step in order to establish a perfectly secret key: this is known as privacy amplification [21]. The process of reconciliation results in a bit sequence which is common to Alice and Bob, but some of its bits may be known to an eavesdropper who has tapped the classical channel. To eliminate this leaked information, Alice and Bob must apply, in common, a binary transformation (usually, a random permutation) to their sequences, and discard a subset of bits from the result. The precise choice of transformation and the number of bits discarded, of course, determine the amount of secrecy of the final key. The objective of this step is to minimise the quantity of correct information which the eavesdropper may have obtained about Alice and Bob's common bit sequence. At the end of the privacy amplification procedure, Alice and Bob's bit sequences can be proven to be identical and absolutely secret, with arbitrarily high probability.

Mayers [99] proved the unconditional security of quantum key distribution under all attacks permitted by quantum mechanics. That is to say, there is no general attack that would allow an enemy to successfully eavesdrop and obtain a key established using BB84 with secret–key reconciliation and privacy amplification.

Variations of the BB84 protocol have been proposed; these include B92 (proposed by Bennett [18]) and Ekert's protocol [48]. The latter makes use of the properties of entanglement.

2.7.6. Quantum Coin–Flipping. Quantum coin–flipping [20, 34] enables two users, Alice and Bob, to establish a common random bit x through the transmission of a single qubit q and its measurement. The protocol relies on the principle that, if Alice and Bob use compatible bases for preparation and for measurement of this qubit, their bit values will be guaranteed to match by the laws of quantum mechanics. Incompatible bases will produce a matching bit value only with probability $\frac{1}{2}$, although in this case the protocol is repeated and the bit discarded. There are various possible attacks that may be performed by an enemy, which would enable him or her to compromise the final bit value.

The detailed steps of the protocol are as follows:

(1) Alice selects b, x uniformly at random, and sends $|\psi_{b,x}\rangle$ to Bob.

(2) Bob selects bit g uniformly at random and sends it to Alice.

(3) Bob selects b; he measures with respect to basis $\hat{b}$ and obtains measurement result $\hat{x}$.

(4) Alice sends b, x to Bob.

(5) If $b = \hat{b}$ and $x \neq \hat{x}$, Bob aborts the protocol because Alice has cheated. If $b \neq \hat{b}$ then the protocol is restarted.

(6) The final common bit is $c = b \oplus g$, where $\oplus$ denotes the exclusive-OR binary operator.

2.7.7. Quantum Secret Sharing. Quantum secret sharing protocols [95] are designed to address the following classical problem.

Problem 2.1 (Threshold Secret Sharing). *A dealer holds a secret* s*, and must send this secret to* n *players such that:*

- *any* k *or more players can reconstruct the secret*
- *all sets of fewer than* k *players as well as eavesdroppers are denied any access whatsoever to the secret.*

The value of k is known as the *threshold*, and is a parameter of the protocol in question. Classical solutions to this problem do achieve *unconditional security*, but only on the assumption that the channels between the dealer and the player, and the channels between players, are private. Quantum protocols for this problem to address this problem have two main features:

(i) the use of quantum channels to protect against eavesdropping (thus fulfilling the requirement for private channels)

(ii) the use of quantum error correction to protect quantum secrets.

Markham and Sanders [95] developed a formalism that embodies both classical and quantum secret sharing protocols in a uniform fashion. In their setting, there are three categories of protocols:

CC: protocols handle a classical secret (*i.e.* a single bit) and assume private point-to-point channels

CQ: protocols handle a classical secret but use quantum key distribution to allow for public channels

QQ: protocols assume a quantum secret (*i.e.* a qubit), quantum channels between the dealer and each of the players, and quantum or classical channels between the players.

Of interest for our purposes is only the CC case, and we note here that a classical secret can be trivially represented using a qubit. Thus a CC protocol is not necessarily a classical protocol; it can be a quantum protocol – in the sense that qubits, rather than classical bits, are being exchanged. All three cases can be presented uniformly using quantum information (and in particular, graph states). The local equivalence of graph states and stabilizer states [146] makes these protocols particularly relevant for our analyses, since we will be focussing on protocols involving the latter class of quantum states.

The formalism in [95] is used to build CC, CQ, and QQ protocols for specific values of k and n, as well as a general case where $k = n$. In Section 6.5 we will look at a model of a CC protocol with $k = n = 3$.

We will need some background on graph states in order to understand the presentation of these protocols.

2.7.7.1. *Graph States.* A *graph state* is a multi-qubit state which is presentable in the form of a graph such that:

- a node corresponds to a qubit in the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
- an edge between two nodes corresponds to the application of a Controlled–Z operation.

The reader is referred to [7] for formal definitions and details.

Example 2.1. *The graph state*

can be expressed in state vector form as follows (where $\wedge Z$ denotes the Controlled-Z gate):

$$\wedge Z \mid ++\rangle = \wedge Z \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle - |11\rangle\right)$$

2.7.7.2. *The Structure of the Protocol.* The structure of a CC type, graph state–based secret sharing protocol with $k = n$ is as follows.

(1) The dealer prepares a graph state $|\psi\rangle$.

(2) The dealer selects a random vertex v of the graph state, and encodes onto it the secret bit s. To do this, he applies the Z gate to the qubit corresponding to this vertex.

(3) The dealer distributes the qubits in the state to the players (by sending the ith qubit to the ith player).

(4) The players measure the qubits as follows:

- the player receiving the qubit corresponding to vertex v measures his qubit in the diagonal basis $\{|+\rangle, |-\rangle\}$
- the other players measure their qubits respectively in the computational basis.

(5) The players communicate their respective measurement outcomes to each other.

(6) The final outcome of the protocol is obtained by computing the XOR of all the players' outcomes together.

Thus the players can obtain the secret bit s which the dealer has originally encoded onto a chosen vertex of the initial graph state. The idea here is that, unless all players coöperate, they will not be able to correctly obtain the original secret. Variations of the protocol do not require all n players to collaborate, as we have assumed here, but we will restrict ourselves to the $k = n$ case.

2.8. Concluding Remarks

In this chapter we have presented the main background concepts relevant to the subject of this thesis. In particular we have focussed on the formalism of quantum mechanics and the basic notions of qubit, quantum gate, and measurement. We have also surveyed a number of quantum protocols and seen how the use of quantum phenomena is used for information exchange. The rest of the thesis will be devoted to a particular method of formalising such protocols and verifying their correctness in an automated manner.

CHAPTER

3

PROBABILISTIC MODEL CHECKING AND EFFICIENT SIMULATION

Electricity is actually made up of extremely tiny particles called electrons, that you cannot see with the naked eye unless you have been drinking.

— Dave Barry

This chapter sets out a first attempt at modelling and verifying simple quantum protocols, such as superdense coding, quantum teleportation, and quantum error correction. We focus on the use of the probabilistic model checker PRISM and describe fundamental techniques for adapting this tool to the analysis of such protocols; a PRISM model of superdense coding is presented as a demonstration of this approach. While this approach does

produce the expected results, it has significant shortcomings, namely, it can handle only very small protocols (with up to 3 qubits in practical terms).

Our purpose here is to demonstrate in which ways classical model checking tools are unsuitable for analyses of quantum protocols. In the process we will discuss the issue of representing a finite, closed set of quantum states and quantum operators. This will lead is in a natural way to consider the Gottesman–Knill theorem [69] and the more general question of efficiently simulating quantum protocols on a classical computer. As we shall see, efficient simulation precludes the possibility of modelling arbitrary quantum protocols; such protocols require the ability to represent a universal set of quantum gates, and the quantum states which arise from their application.

This chapter is based on material from the paper [62]. The proof of closure in Section 3.1 is originally due to Simon Gay.

3.1. Adapting the Probabilistic Model Checker PRISM

PRISM [122] is an acronym for *probabilistic symbolic model checker,* and is designed for modelling and validating systems which exhibit probabilistic behaviour. Whereas a logical model checker, such as SPIN [78], only states whether a system model σ satisfies a temporal formula Φ, a tool such as PRISM computes the probability with which such a formula is satisfied, *i.e.* the value of $\mathsf{P}_{\sigma,\Phi} = \mathsf{Pr}\{\sigma \models \Phi\}$ for given σ and Φ. The models catered for by PRISM may incorporate specific probabilities for various behaviors and so may the formulas used for verification. Probabilistic models and PRISM–like tools find applications in numerous areas of computer science where random behaviour is involved. Oft–cited applications are randomized algorithms, real–time systems and Monte Carlo simulation. The application of probabilistic model checking to quantum systems is deemed appropriate, since such systems manifest probabilistic behaviour during the process of measurement; to reason about such phenomena one should account for this.

PRISM has a built–in specification language based on *Reactive Modules*, originally due to Alur and Henzinger [6]. Using this language the user can describe probabilistic behaviour. Internally, a PRISM model is represented by a *probabilistic transition system.* In such a system, each step in a

computation is represented by a *move,* or *transition,* from a particular state s to a distribution π of successor states [121, 122].

The probabilistic temporal logic PCTL [37] is used as the principal means for defining properties of systems modelled in PRISM. It suffices for our purposes to remind the reader of the meaning of the operator $\mathscr{U}$, known as "unbounded until". The formula $\Phi_1 \mathscr{U} \Phi_2$ expresses the fact that Φ_1 holds continuously from the current state onward, *until eventually* Φ_2 becomes true. The property $P \geqslant 1\ [\Phi_1 \mathscr{U} \Phi_2]$ states that the formula $\Phi_1 \mathscr{U} \Phi_2$ is true with certainty, *i.e.* with a probability of unity; we use PRISM to check whether such a property holds in a given model.

In order to use a classical probabilistic model checker to verify quantum protocols, we need to model the quantum states that arise in a given protocol, and the effect of specific quantum operations on these states. PRISM itself only allows positive integer and boolean variables to be used in models. So how can we model the states of quantum systems, and the quantum operations arising in protocols, using only classical data types and arithmetic?

Single qubits can be in a superposition of two states, while classical variables can only take on a single value in any given state. The coefficients of these states can be any two complex numbers whose moduli squared sum to unity, and there is an uncountable infinity of these; of course, PRISM can only work with a finite state space. Furthermore, quantum systems consisting of many qubits can be in entangled states, which, unlike classical systems, cannot be decomposed into products of individual states. What is needed, therefore, is a means of representing quantum states fully and consistently, in a form that PRISM can handle.

Of all the possible quantum states of an n–qubit system, we identify the finite set of states which arise by applying the operations CNot, Hadamard (H), and $\sigma_0, \sigma_1, \sigma_2, \sigma_3$ to the input state $|0\rangle^{\otimes n}$, and all the states that arise from subsequent applications of these gates to the resulting states. We confine our analyses to protocols that involve *only* this restricted set of operations. At present, determining which states belong to this set is done manually.

Consider a very simple system: a single qubit, being acted upon through the Hadamard gate and through measurement in the standard basis. For our purposes, the state of the qubit may be $|0\rangle$, $|1\rangle$, or an equal superposition of the two. In fact, these states are sufficient to model the BB84

protocol for quantum key distribution [20]. The quantum states which we need to represent in order to model this simple system are thus:

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

To model this small, finite set of quantum states, which is closed under the operation of the Hadamard gate and the Pauli operators, we represent each state by assigning a unique integer from 0 to 3 to it, and we use straightforward transitions from one integer value to another to model the action of the gate. We call this *state space enumeration*.

The actual PRISM model for this, as well as the possible results of measurement in the computational basis, is listed below.

```
probabilistic
module Qubit
state : [0..3]; // 0 is |0>, 1 is |1>, 2 is |+>, 3 is |->
result : [0..1]; // Result of measurement in standard basis
[hadamard] (state=0) -> (state'=2);
[hadamard] (state=1) -> (state'=3);
[hadamard] (state=2) -> (state'=0);
[hadamard] (state=3) -> (state'=1);
[measure] (state=0) -> (state'=0) & (result'=0);
[measure] (state=1) -> (state'=1) & (result'=1);
[measure] (state=2) -> 0.5 : (state'=0) & (result'=0)
                     + 0.5 : (state'=1) & (result'=1);
[measure] (state=3) -> 0.5 : (state'=0) & (result'=0)
                     + 0.5 : (state'=1) & (result'=1);
endmodule
```

A protocol such as superdense coding can be expressed as a step-by-step interaction with a *two–qubit system*. In order to model the states of 2– and 3–qubit systems, the quantum operators

and the measurements which arise in this and related protocols such as teleportation, we have developed a code generation tool called PRISMGEN. This tool generates a PRISM code fragment, or *module*, in which each quantum state is represented by a unique positive integer. Every quantum operator used in a particular protocol is coded as a set of deterministic transitions from one quantum state to another. PRISMGEN calculates these transitions by multiplying the unitary matrix, which corresponds to a particular operator, with each quantum state vector of interest. A measurement is modelled by a set of probabilistic transitions, leading to the various possible outcomes with equal probability. For simplicity, we have only considered states whose measurement outcomes are all equiprobable, although PRISM does allow us to model the more general case.

From the overall state space for a 2–qubit system, a certain subset is closed under the CNot, Hadamard and Pauli operations. This subset consists of the following states, which are 24 in total:

- 4 states corresponding to the four basis vectors, *i.e.* $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.
- 12 states which are sums of two basis vectors, *i.e.* of the form $\frac{1}{\sqrt{2}}|xy\rangle \pm \frac{1}{\sqrt{2}}|x'y'\rangle$ with $x \neq x', y \neq y'$ and $x, x', y, y' \in \{0, 1\}$.
- 8 states which are sums of all four basis vectors.

Remark 3.1. *The above set of 24 states is closed under the* CNot, *Hadamard and Pauli operations. This is a direct consequence of the stabilizer formalism.*

PROOF. These states can be expressed in the following way.

(1) The single basis vectors: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$

(2) The states containing two basis vectors can be separated into three subclasses:
   (a) $\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \otimes |0\rangle$, $\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \otimes |1\rangle$
   (b) $|0\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$, $|1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$
   (c) $\frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$, $\frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$

(3) The states containing four basis vectors can be expressed in any of the forms:
   (a) $\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \otimes |0\rangle \pm \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \otimes |1\rangle$
   (b) $|0\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \pm |1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$
   (c) $\frac{1}{2}(|00\rangle \pm |01\rangle \pm |10\rangle \pm |11\rangle)$

It is obvious that each set (1.)–(3.) individually is closed under each σ_i (applied to either qubit) and CNot these operations are permutations of the basis vectors. Each set has an evident symmetry among the basis vectors (taking (3.a) and expanding (2.) into an explicit list of states). Applying H to the first qubit gives a bijection between (1.) and (2.a), between (2.b) and (3.a), and between (2.c) and (3.b). Applying H to the second qubit is similar. △

Our PRISMGEN tool enumerates these states and calculates the transitions corresponding to the various operations. The resulting PRISM module can be included as part of any model which involves measurements and the application of these operations to a system of two qubits.

The situation with a system of three qubits is similar. We have developed a 3–qubit version of PRISMGEN, which gives us the ability to model protocols such as those for quantum teleportation and quantum error correction, as detailed in [62].

3.1.1. Analysing the Superdense Coding Protocol with PRISMGEN. The simplest quantum protocol which we will use to illustrate our techniques is the superdense coding scheme [22]. This scheme makes it possible to encode a pair of classical bits on a single qubit. With superdense coding, a quantum channel with a capacity of a single qubit is all that is necessary to transmit twice as many bits as a serial classical channel. Superdense coding is essentially a computation on a two–qubit system; therefore, the PRISM model of this protocol uses the 2–qubit version of PRISMGEN. We begin with a description of the protocol, and proceed to show how it is modelled and verified with PRISM.

The model of superdense coding consists of four PRISM modules. Of these four, one module is generated automatically by PRISMGEN and describes the possible states of the two qubits. There is a module specifying Alice's actions, and similarly one for Bob's. The actions in these two modules correspond exactly to the successive application of quantum gates in Fig. 2.2.3.

Before we examine the workings of this model in detail, consider the following observations, which highlight the capabilities of PRISM. In the PRISM model, Alice's first action is to select one of the four possible messages (represented by the integers 0, 1, 2, 3); each message has an equal probability, $\frac{1}{4}$, of being chosen. This is an assumption we made when constructing this

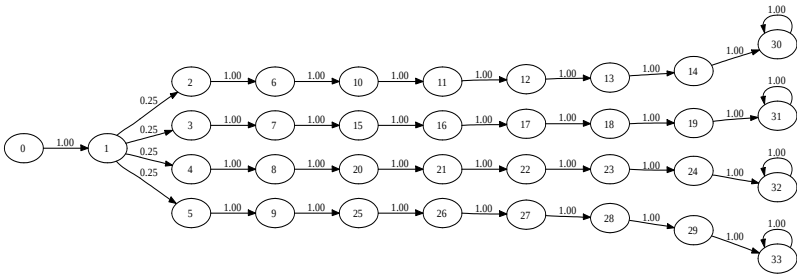


FIGURE 3.1. Internal probabilistic state transition system corresponding to the PRISM model of superdense coding.

model, but it is possible to specify different respective probabilities for the four choices. Another point worth noting is that, depending on which message is chosen, the protocol proceeds in one of four distinct ways; PRISM actually considers *all* these possibilities when testing the validity of a property. This is precisely why we advocate the use of model-checking for these analyses, as opposed to simulation of quantum protocols, proposed elsewhere; simulators only treat one of several possible executions at a time.

PRISM interprets the superdense coding model as the probabilistic transition system[1] depicted in Fig. 3.1. The nodes in the graph correspond to the internal state numbers which PRISM assigns to each step in the protocol. Each internal state number corresponds to a tuple with the states of all variables in a particular model.

Read from left to right, Fig. 3.1 shows the succession of internal state numbers for the four possible messages that Alice may send to Bob in superdense coding, The initial state, labelled 0, is where all variables are first set. In internal state 1, Alice makes her choice of message, setting the `msg` variable accordingly and leading to one of the internal states 2, 3, 4, or 5 with equal probability. The succession of PRISM's internal states in Table 1, which includes the value of each variable in the model, corresponds to the case in which Alice has chosen the message **(1, 0)** and, hence, applied the σ_1 operator to her qubit. The quantum state of the two-qubit system is represented by the variable `state`, which corresponds to the actual quantum state indicated in the final column of this table.

[1]Note that the probabilities in this diagram arise from Alice's choice of message, not from measurement outcomes. In general, it is measurement that produces probabilistic transitions.

State number	`alice_step`	`msg`	`bob_step`	`result`	`state`	Quantum State
0	0	0	0	0	0	$\|00\rangle$
1	1	0	0	0	8	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$
4	2	2	0	0	8	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$
18	3	2	0	0	9	$\frac{1}{\sqrt{2}}(\|01\rangle + \|10\rangle)$
19	3	2	1	0	9	$\frac{1}{\sqrt{2}}(\|01\rangle + \|10\rangle)$
20	3	2	2	0	7	$\frac{1}{\sqrt{2}}(\|00\rangle - \|10\rangle)$
21	3	2	3	0	2	$\|10\rangle$
22	3	2	4	0	2	$\|10\rangle$
23	3	2	5	2	2	$\|10\rangle$

TABLE 1. The transitions of the PRISM model for superdense coding for the case when the message chosen by Alice is **(1,0)**.

When Bob has finished his measurement, and the dense coding protocol terminates, one of the internal states 11, 17, 23, 29 is reached, corresponding to the final nodes in the graph in Fig. 3.1. The property required for verification must be expressed in terms of the final state. When the dense coding protocol terminates, Bob's measurement result, *i.e.* the pair of classical bits (x', y'), must match Alice's original choice (x, y). This requirement is expressed using PCTL as follows:

$$\mathrm{P} \geqslant 1 \, [\, \textbf{true} \, \mathscr{U} \, ((\texttt{protocol\_finished}) \, \wedge \, (\texttt{result} = \texttt{msg}))\,] \tag{3.1}$$

The PCTL formula in Eq. 3.1 stipulates that the probability of Bob's result matching Alice's choice is 1. Model checking with PRISM confirms that this property holds (*i.e.* this property is true in all executions of the model). We have thus proven, using the PRISM model checker, that the dense coding protocol always succeeds in transmitting two classical bits using a single qubit. Clearly, this is not difficult to prove by hand; however, we have used superdense coding as a simple demonstration of this first approach to the verification of quantum protocols.

3.1.2. Analysing Quantum Teleportation using PRISMGEN. Our next example is the quantum teleportation protocol [23], which involves a computation on three qubits.

The PRISM model of teleportation is similar in appearance to that for superdense coding, and it is not included here due to lack of space. It is a transformation on a collection of three qubits, as opposed to the two for superdense coding. This calls for the 3-qubit version of PRISMGEN. Other

than this, the PRISM model itself is unremarkable, and matches the structure of the quantum circuit for teleportation, given in the appendix. Verifying the teleportation protocol with PRISM is more involved. Short of manual calculation, it is not possible to predict what the quantum state of the entire 3-qubit system will be at the end of the teleportation protocol; indeed, there are several possible final states, depending on which quantum state Alice chooses to transmit to start with. We are interested in checking that the state of Bob's qubit matches Alice's original qubit state, $|\psi\rangle$, which is assumed to be one of $|0\rangle$, $|1\rangle$, $|+\rangle$ $|-\rangle$. To formulate a usable property for verification, we need to express this requirement in terms of the overall state of the 3–qubit system.

Formally, the specification of the teleportation protocol is this: if the initial state of the 3-qubit system is of the form $|\psi\rangle \otimes |00\rangle$, then the final state will be of the form $|\phi\rangle \otimes |\psi\rangle$, where $|\phi\rangle$ is a two–qubit state. Let's consider this in more detail. If Alice chooses to teleport $|\psi\rangle = |0\rangle$, the final state of the 3–qubit system will be of the form $|\phi\rangle \otimes |0\rangle$. Similarly, if Alice chooses to teleport $|\psi\rangle = |1\rangle$, the final state of all three qubits will be of the form $|\phi\rangle \otimes |1\rangle$. Finally, if Alice chooses to teleport the superposition $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, the final state of the three qubits will be of the form $|\phi\rangle \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)$.

Clearly, the PCTL property necessary for verification will depend on the choice of $|\psi\rangle$; it will stipulate that, when the teleportation protocol has completed, the final state of the 3-qubit system will have one of the forms given above. In particular, if the input state is $|0\rangle$, the necessary property is

$$\mathrm{P} \geqslant 1 \, [\, \mathbf{true} \, \mathscr{U} \, ((\texttt{telep\_end}) \wedge ((\texttt{st} = s_1) \vee \cdots \vee (\texttt{st} = s_n))) \,] \tag{3.2}$$

where `telep_end` is a predicate which is `true` when the protocol completes, and the values $s_1, \ldots, s_n$ represent quantum states of the form $|\phi\rangle \otimes |0\rangle$. If the input state is $|1\rangle$, the necessary property has exactly the same form as equation 3.2, but the values $s_1, \ldots, s_n$ represent quantum states of the form $|\phi\rangle \otimes |1\rangle$; similarly for the case when the input state is the superposition $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$.

In other words, in order to formulate the property needed to verify the protocol, we need to choose the input states and determine the possible final states of the three–qubit system in advance. This may be seen as begging the question; there is little point in verifying a protocol whose final outcome has already been calculated by hand. We have developed an auxiliary tool to PRISMGEN, which computes the internal state numbers $s_1, \ldots, s_n$ corresponding to the desired final states. When the PCTL property for a particular input is supplied to PRISM, the tool proves that the teleportation model works as expected. Since the model–checker necessarily constructs a finite state space for the model, the teleportation protocol can only be verified for a specific, known set of inputs, rather than an arbitrary quantum state.

3.1.3. Analysing the Quantum Bit–Flip Code Using PRISMGEN. Our third and final example is the quantum bit–flip code for error correction [139]. In order to correct a single bit flip error, which may occur during the transmission of a single qubit state, this code represents the state by using a collection of three qubits. In particular, the qubit state $|0\rangle$ is encoded as $|000\rangle$ and the state $|1\rangle$ is encoded as $|111\rangle$. A bit flip error on the second qubit, for example, transforms $|000\rangle$ into $|010\rangle$.

In order to detect such an error, two additional qubits are used; they are known as *ancillas*. By applying a sequence of operations and measurements to the ancillas, the so–called *error syndrome* is obtained, which determines the location of the error. Then, the σ_1 operator is applied to the erroneous qubit, thus restoring the initial quantum state of the 3–qubit system.

Our PRISM model of the protocol for the quantum bit–flip code includes a channel which perturbs the transmitted qubit with a chosen probability; this probability is a parameter of the model, and can be varied as required. The model uses the output from the 3–qubit version of the PRISMGEN tool. When the syndrome computation is taken into account, there are in total five qubits whose states need to be modelled; since we have not yet implemented a code generator for 5–qubit quantum systems, the state transitions for the syndrome computation are calculated in advance and manually coded into PRISM.

To verify the correctness of the quantum bit–flip code, we need to ensure that: independently of which of the three qubits is perturbed and with what probability this occurs, the protocol does succeed in correcting the error. Thus, at the end of the protocol, the state of the 3–qubit system should be in one of the following forms (where $|\phi\rangle$ is a two–qubit state):

$$|0\rangle \otimes |\phi\rangle\,, \text{ if the input state was } |0\rangle \tag{3.3}$$

$$|1\rangle \otimes |\phi\rangle\,, \text{ if the input state was } |1\rangle \tag{3.4}$$

$$\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes |\phi\rangle\,, \text{ if the input state was } \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.5}$$

The properties used in PRISM to verify the protocol are analogous to those for teleportation, taking the form

$$\mathrm{P} \geqslant 1\,[\,\mathbf{true}\;\mathscr{U}\;((\mathtt{qbf\_end})\;\wedge\;((\mathtt{st} = s_1)\;\vee\;\cdots\;\vee\;(\mathtt{st} = s_n)))\,] \tag{3.6}$$

where `qbf_end` is a predicate which holds when the protocol completes, and the values $s_1, \ldots, s_n$ represent quantum states of one of the forms given in Eqs. 3.3—3.5. PRISM confirms that the protocol does indeed leave the 3–qubit system in one of these forms, depending on the input, as expected.

3.2. The Limitations of the PRISMGEN Approach

The PRISMGEN tool identifies which quantum operations arise in a given protocol (these are chosen from a finite set which has been defined in advance), and applies the corresponding operator matrices to a quantum state vector of suitable dimension. The quantum state vector is initially in the basis state $|0\rangle^{\otimes n}$, and the results of applying the operators of interest successively to this state are recorded internally in PRISMGEN. Each new state which arises is given a label, and this label is represented by the value of an integer variable in the generated PRISM model.

PRISMGEN was a first attempt at devising a model checking approach targeted at quantum protocols, but suffered from the following shortcomings:

- the generation of PRISM code representing the quantum operations arising in a protocol was quite inefficient, as quantum states were represented internally using the state vector representation (whose size grows exponentially with the number of qubits in the system being modelled). The approach was only successful for systems of 2-3 qubits, and became unwieldy for 4-qubit systems.
- the allowed quantum operations were limited to: CNot, Hadamard (H), and the Pauli operators X, Y, Z. These operations were the most common ones arising in applications, although confining ourselves to these operations seemed arbitrary and became quite limiting.
- the semantics of protocol models were necessarily described by classical probabilistic transition systems, as all models were targeted at the PRISM model checker. This meant that the two key features of quantum information, superposition and entanglement, could not be described properly.
- the properties for verification were expressed in PCTL, and in most cases, these had to be automatically generated also.
- the use of PRISMGEN was impractical, since verification of any protocol would be a three-stage process. Firstly, the quantum protocol of interest would be described and manually coded in PRISM. Secondly, PRISMGEN would be executed to generate state transition code for all quantum operations arising in the protocol. Then, the generated file would be combined with the manually coded protocol description and submitted to PRISM for verification along with a property expressed in PCTL.

3.3. Efficiently Simulable Quantum Protocols

Developing a self-contained verification tool would clearly solve the practical problems described above, although certain issues, such as which states and operations to represent and how, still needed further consideration. There exists a fundamental theoretical result, referred to as the Gottesman–Knill theorem [1, 68, 69], which states that there is a specific, restricted set of

quantum operations which can be simulated efficiently (and hence used in a verification tool) on a classical computer. This theorem may be stated as follows.

Theorem 3.1 (Gottesman–Knill)**.** *There is a restricted class of n–qubit quantum states in Hilbert space $\mathscr{H}_n$, such that each state $|\psi\rangle$ in this class satisfies the following:*

- *$|\psi\rangle$ can be obtained from $|0\rangle^{\otimes n}$ by* CNot*, Hadamard and phase operators only.*
- *$|\psi\rangle$ can be obtained from $|0\rangle^{\otimes n}$ by* CNot*, Hadamard and phase and measurement operators only.*
- *$|\psi\rangle$ is stabilized by exactly 2^n Pauli operators.*
- *$|\psi\rangle$ is uniquely determined by the stabilizer group $S(|\psi\rangle) = \mathsf{Stab}(|\psi\rangle) \cap \mathscr{P}_n$ (i.e. the group of Pauli operators that stabilize $|\psi\rangle$)*
- *The total number of classical bits needed to specify $|\psi\rangle$ is $n(2n+1)$, and these bits can be updated in **polynomial time** after a* CNot*, Hadamard, phase or measurement operation is applied to $|\psi\rangle$.*

Any quantum circuit consisting entirely of Clifford group gates (Hadamard, CNot and the *phase gate* [71, 111]) and single–qubit measurements, is referred to as a *stabilizer circuit,* and any state arising by applying a stabilizer circuit to $|0\rangle^{\otimes n}$ is a *stabilizer state*. Stabilizer circuits are important in quantum error correction and fault-tolerant quantum computation. We have previously set out the mathematics of the associated *stabilizer formalism* in Section 2.6.

The existence of a restricted class of quantum operations whose effect can be efficiently simulated classically is extremely important for our purposes, since this result makes it clear which quantum protocols we can effectively analyse using current technology. However, this restricted class of operations falls short of the full power of quantum computation, and there are certainly protocols and attacks which cannot be expressed using only these. There are several related results in this space, including the matrix product state formalism (see e.g. [125] and the Valiant's class of efficiently simulable quantum circuits [145].

3.4. Concluding Remarks

We have considered in this chapter how an existing model checking tool might be adapted to model small quantum protocols. The PRISMGEN code generation tool has been discussed. We considered the limitations of this approach and related it to the Gottesman–Knill theorem, which states which quantum gates can be efficiently represented and simulated classically.

Now, the set of gates which PRISMGEN considers is certainly not universal. This tool was implemented before the author became aware of the Gottesman–Knill theorem and its ramifications; it is clear that the results obtained with this approach could have been slightly generalised to take into account also the phase gate S, without losing the closure property for the set of quantum states allowed in protocols.

It is also interesting to note that, due to the explicit enumeration of the quantum state space which PRISMGEN performs, it is not possible to make use of PRISM's efficient symbolic model checking algorithms. In effect, PRISMGEN serves to build an explicit representation of all the quantum states which may arise in a protocol, while PRISM itself computes the possible communications and interleavings for all these possible states.

Concerning protocol properties, we have clearly not used the probabilistic fragment of the specification logic and have confined ourselves to "with probability 1" formulae. However, there may be a case to be made for analysing protocols involving many measurements, where formulae involving explicit probabilities would be useful.

The PRISMGEN approach only allowed the analysis of very simple quantum protocols, and did not provide the best means of specifying properties for such; however, it formed the first experimental foundation for building a dedicated model checking tool. As it was realised at a later stage, the quantum gates allowed for by PRISMGEN are a subset of the Clifford group gates and protocols which involve only these can be simulated in polynomial time using the algorithm of Aaronson and Gottesman.

We will put simulation and verification matters aside in the next chapter with a view to presenting a specification language for describing quantum protocols formally. We will see examples

of its use and its formal syntax and semantics. Also we will consider a logic for describing key properties of quantum protocols.

CHAPTER

4

SPECIFYING QUANTUM PROTOCOLS AND THEIR PROPERTIES

See the little phrases go,

Watch their funny antics.

The men who make them wiggle so

Are teachers of semantics.

— F. Winsor (Space Child's Mother Goose 1)

In this chapter we will present a modelling language for quantum protocols with a formal semantics and type system. We will also discuss quantum computation tree logic and how this logic may be interpreted over models of protocols expressed in our language.

4.1. QMCLANG: A Modelling Language

Various programming and specification formalisms have been proposed in order to address the shortcomings of the quantum circuit model when describing quantum protocols. These include quantum programming languages and quantum process algebras (see [60] for a survey). We have built an imperative-style concurrent specification language, QMCLANG , which is intended to be used as an input language for the Quantum Model Checker (QMC) (see Chapter 5). We will proceed to discuss the nature of this specification language, stating its formal syntax and semantics.

The language is designed to allow for the description of systems with classical data and communication as well as manipulation and transmission of a finite number of qubits. A QMCLANG model consists of: (1) a set of process declarations, and (2) a set of global (channel) variable declarations.

Each process defines the behaviour of a system component and has its own local variables (classical variables of integer, real or boolean type, and qubit variables) as well as access to all global variables (which represent channels of communication between processes).

The minimal QMCLANG program consists of a single process with no actions:

```
program Minimal;
process SingleProcess;                    2
begin
end;                                      4
endprogram.
```

Classical variables are declared and used just as in conventional computer languages such as Pascal [150]; the language includes assignment statements and simple expressions (*e.g.* `a:=b+2;` is a valid statement assuming `a, b` are suitably declared). Qubit variables store references to qubits

in the global quantum state and are used in one of four ways (assuming `a, q` have been declared as qubit variables):

- the statement `q:=newqubit;` allocates a new qubit in the global quantum state and sets the value of `q` to represent the index of that qubit. Before such a statement is issued, although a qubit variable `q` may have been declared it does not stand for an actual qubit in the quantum state.
- the statement `a:=savequbit q;` stores the global quantum state at that point in execution. The ability to create *history variables* such as `a` is a special feature of the language which is useful for property verification.
- in statements which apply quantum operators (H, *CNot*, S, X, Y, Z) to particular qubits, the corresponding qubit variables are used, *e.g.*: `had q; X q; cnot a q;`.
- Measurements of qubits in the standard basis are performed using expressions of the form `meas q`. Normally these expressions are assigned to an integer value, so that the measurement outcome (0 or 1 for a single–qubit measurement) is recorded.

Apart from assignments and the statements of the forms discussed above, processes are able to send and receive the values of variables on channels. The sample program below demonstrates this feature of the language. Note that the syntax for sending and receiving is reminiscent of the formalism CSP [76].

This example also demonstrates two other characteristics of the language: *parallelism* and *executability*. The `SendReceive` system model declares one channel variable and two parallel processes `Sender` and `Receiver`. All processes in a QMCLANG program are supposed to be executed in parallel, leading to several possible interleavings. However, the action `ch?b` cannot be executed before the channel `ch` has received a value, so there is only possible execution of this particular program. There is no explicit parallelism operator for processes (which is common in other specification languages), as it is implicit.

Simple statements can be combined together to form *blocks*, which are executed in a single step. A block is simply a sequence of statements enclosed in braces (‘`{`’ and ‘`}`’). The semantics of

```
program SendReceive;
var ch: channel of integer;
process Sender;
var a: integer;
begin
   a := 2;
   ch!a;
end;
process Receiver;
var b: integer;
begin
   ch?b;
end;
endprogram.
```

FIGURE 4.1. An example with sending and receiving of variables.

the language is such that one step in the execution of a process consists of either a single statement by itself or a block.

QMCLANG models can also include non–deterministic choices and loops. In particular, a process can perform a choice between several sequences of statements, and such a choice can be performed repeatedly. Here is an example of a process involving a loop (this loop is repeated up to a maximum number of times, which is implementation-dependent):

```
process Looping;
var a: integer; q: qubit;
begin
   q := newqubit;
   do
   □a:=0; had q;
   □a:=1; X q;
   od
end;
```

FIGURE 4.2. An example with looping.

The `Looping` process repeatedly chooses, either to set `a` to 0 and then apply the H gate to `q`, or to set `a` to 1 and then apply the X gate to `q`. The first statement in an option (an option is a sequence of statements preceded by the symbol '□' – in ASCII form this is represented by '`::`') can be an expression stating a condition; this condition is used to determine whether the option

is executable or not. These conventions are habitual in guarded–command languages and are inspired by a proposal of Dijkstra [47].

Next we define the formal syntax of QMCLANG along with its operational semantics.

4.2. Syntax

The *concrete syntax* of QMCLANG is given in Fig. 4.3. The actual parser for the language uses a slight variation of the grammar shown, which also takes into account the precedence of arithmetic operators. The grammar defines the four base types for variables and the channel type (T), expressions (E), statements (S), sequences of statements (Q), nondeterministic choices (H), guarded commands (G), commands (C), variable declarations (V and D), processes (P) and programs (M).

$$
\begin{array}{rcl}
T & ::= & \texttt{integer} \mid \texttt{bool} \mid \texttt{real} \mid \texttt{qubit} \\
 & \mid & \texttt{channel of } T \\
E & ::= & n \mid r \mid x \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 * E_2 \mid E_1/E_2 \\
 & \mid & \texttt{true} \mid \texttt{false} \mid \texttt{not } E \mid E_1 \texttt{ and } E_2 \mid E_1 \texttt{ or } E_2 \\
 & \mid & E_1 = E_2 \mid E_1 < E_2 \mid E_1 > E_2 \mid \texttt{meas } x \mid \texttt{newqubit} \\
S & ::= & E \mid x := E \mid x_1 ! x_2 \mid x_1 ? x_2 \mid \texttt{cnot } x_1 x_2 \mid \texttt{had } x \\
 & \mid & \texttt{ph } x \mid \texttt{X } x \mid \texttt{Y } x \mid \texttt{Z } x \\
Q & ::= & S;\ Q \mid S \\
H & ::= & \square Q\ H \mid \square Q \\
G & ::= & \texttt{if } H \texttt{ fi} \mid \texttt{do } H \texttt{ od} \\
C & ::= & S\ C \mid G\ C \mid \epsilon \\
V & ::= & \texttt{var } D \mid \epsilon \\
D & ::= & x : T;\ D \mid \epsilon \\
P & ::= & \texttt{process } p\ V \texttt{ begin } C \texttt{ end } P \mid \epsilon \\
M & ::= & \texttt{program } p\ V \texttt{ begin } P \texttt{ end}
\end{array}
$$

FIGURE 4.3. Concrete Syntax for QMCLANG.

Note that QMCLANG does not include a universal set of quantum operators; thus, by design, it is not a universal quantum programming language. (Adding the $\pi/8$ gate would allow for approximate universality and this only requires adding an extra semantic rule for this case.)

4.3. Semantics

The semantics are defined over the *internal syntax* of the language, given in Fig. 4.4. The mapping from concrete to internal syntax should be intuitively clear (note that declarations of variables have been omitted and that certain additional notations for operators have been introduced). The metavariables used in the different productions belong to the following syntactic categories: $n \in \mathsf{Num}$, $r \in \mathsf{Real}$, $x \in \mathsf{Var}$, $c \in \mathsf{Chan}$.

$$
\begin{array}{rll}
op & ::= + \mid - \mid * \mid / & op \in \mathsf{Op} \\
bop & ::= \wedge \mid \vee & bop \in \mathsf{BOp} \\
qop & ::= had \mid ph \mid X \mid Y \mid Z & qop \in \mathsf{QOp} \\
rel & ::= > \mid < \mid = & rel \in \mathsf{Rel} \\
E & ::= n \mid r \mid x \mid E_1\ op\ E_2 \mid E_1\ bop\ E_2 \mid E_1\ rel\ E_2 & \\
 & \quad \mid true \mid false \mid \neg\, E & E \in \mathsf{Exp} \\
S & ::= E \mid x := E \mid x := measure\ x_1, \ldots, x_k \mid x := newqubit & \\
 & \quad \mid c!x \mid c?x \mid cnot\ x_1\ x_2 \mid qop\ x & S \in \mathsf{Stmt} \\
Q & ::= S \mid S;\ Q \mid \epsilon & Q \in \mathsf{Seq} \\
H & ::= Q \mid Q \square H \mid \epsilon & H \in \mathsf{Choice} \\
G & ::= H \mid do(H) \mid \epsilon & G \in \mathsf{GCom} \\
C & ::= S;\ C \mid G;\ C \mid \epsilon & C \in \mathsf{Com} \\
P & ::= C \parallel P \mid \epsilon & P \in \mathsf{Program}
\end{array}
$$

FIGURE 4.4. Internal Syntax for QMCLANG.

Definition 4.1 (Values)**.** *The set of possible values which arise in the language is*

$$\mathsf{ValueSet} = \mathbb{R} \cup \mathbb{B} \cup \{\mathsf{null}\} \cup \mathbb{Q}$$

where $\mathbb{Q} = \{\mathsf{qb}_1, \mathsf{qb}_2, \ldots\}$, $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$, *and the values* $\mathsf{qb}_1, \mathsf{qb}_2, \ldots$ *represent each of the qubits. The special value* null *is used as the content of an empty channel variable.*

$$
\begin{aligned}
\longrightarrow_E &: \mathsf{Exp} \times \mathsf{Store} \times \mathscr{H} \hookrightarrow \mathsf{ValueSet} \times \mathscr{H} \\
\longrightarrow_S &: \mathsf{Stmt} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \hookrightarrow \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \\
\longrightarrow_Q &: \mathsf{Seq} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \hookrightarrow \mathsf{Seq} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \\
\longrightarrow_H &: \mathsf{Choice} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \hookrightarrow \mathsf{Choice} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \\
\longrightarrow_G &: \mathsf{GCom} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \times \mathbb{N} \hookrightarrow \mathsf{Com} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \times \mathbb{N} \\
\longrightarrow_C &: \mathsf{Com} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \times \mathbb{N} \hookrightarrow \mathsf{Com} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \times \mathbb{N} \\
\longrightarrow_P &: \mathsf{Program} \times \mathsf{ChanStore} \times \mathsf{StoreSet} \times \mathscr{H} \hookrightarrow \mathsf{Program} \times \mathsf{ChanStore} \times \mathsf{StoreSet} \times \mathscr{H}
\end{aligned}
$$

FIGURE 4.5. Transition Relations for the Operational Semantics.

Definition 4.2 (Store)**.** *A **store** is a partial mapping from variables to values (similarly a channel store):*

$$\mathsf{Store} = \{f : \mathsf{Var} \cup \mathsf{QVar} \hookrightarrow \mathsf{ValueSet}\}$$

$$\mathsf{ChanStore} = \{f : \mathsf{Chan} \hookrightarrow \mathsf{ValueSet}\}$$

A process store represents the local state of an individual process; we denote by StoreSet a set of stores:

$$\mathsf{StoreSet} = \mathsf{Store} \times \mathsf{Store} \times \cdots$$

We can now define the transition relations which give the operational semantics of the language. The types of these relations are given in Fig. 4.5. In rules we will use $\sigma \in \mathsf{Store}$, $\Sigma \in \mathsf{StoreSet}$, $\kappa \in \mathsf{ChanStore}$.

Definition 4.3 (State)**.** *The overall state at any time during a computation is given by an element of the set*

$$\mathsf{State} = \mathscr{H} \times \mathsf{StoreSet} \times \mathsf{ChanStore}$$

Definition 4.4 (Substitution)**.** *Substitution of variables is as follows:*

$$\sigma[x \mapsto v](y) = \begin{cases} v & \textit{if } y = x \\ \sigma(y) & \textit{otherwise} \end{cases}$$

We will require some auxiliary functions in the definitions of the semantics of expressions. In particular, we will require:

$$f_A : \mathsf{Op} \times \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$$

$$f_B : \mathsf{BOp} \times \mathbb{B} \times \mathbb{B} \mapsto \mathbb{B}$$

$$f_R : \mathsf{Rel} \times \mathbb{R} \times \mathbb{R} \mapsto \mathbb{B}$$

which are defined as follows:

$$f_A(op, a_1, a_2) = \begin{cases} a_1 + a_2 & \text{if } op = + \\ a_1 - a_2 & \text{if } op = - \\ a_1 \cdot a_2 & \text{if } op = * \\ a_1 \div a_2 & \text{if } op = / \end{cases}$$

$$f_B(bop, b_1, b_2) = \begin{cases} b_1 \wedge b_2 & \text{if } bop = \wedge \\ b_1 \vee b_2 & \text{if } bop = \vee \end{cases}$$

$$f_R(rel, a_1, a_2) = \begin{cases} \text{true if } a_1 > a_2, \text{false otherwise} & \text{when } rel \text{ is } > \\ \text{true if } a_1 < a_2, \text{false otherwise} & \text{when } rel \text{ is } < \\ \text{true if } a_1 = a_2, \text{false otherwise} & \text{when } rel \text{ is } = \end{cases}$$

Let $\mathcal{IR} : \mathsf{Stmt} \times \mathsf{State} \to \mathsf{Choice}$ denote a syntactic translation function with

$$\mathcal{IR}(x := \textit{measure } x_1, \ldots, x_k, s) = (x := \alpha_1 \,\square\, x := \alpha_2 \,\square\, \cdots \,\square\, x := \alpha_m) \tag{4.1}$$

so that the non–deterministic choice construct produced by this function reflects all possible outcomes $\{\alpha_i \mid 0 \leqslant \alpha_i \leqslant 2^{k-1}\}$ of the measurement, from the current state $s \in \mathsf{State}$, of the qubits represented by variables $x_1, \ldots, x_k$ (see also Section 5.3). The probabilities of the different outcomes are not taken into account, since for stabilizer states the measurement outcomes 0 and 1 are equiprobable.

4.3.1. Expressions. The semantics of expressions is quite straightforward. Base values include integers (n), real numbers (r), and the boolean values *true* and *false*. Qubit values are not explicitly represented in the language, so that it is not possible to assign an explicit value to a qubit variable. Variables of all types are stored in the process stores, and transition relations such as $\longrightarrow_E$ take into account the store σ of the process currently being executed. Transition relation $\longrightarrow_E$ is defined inductively over all expressions below.

$$\frac{}{(n, \sigma, |\psi\rangle) \longrightarrow_E (n, |\psi\rangle)} \quad \text{(R-NAT)}$$

$$\frac{}{(r, \sigma, |\psi\rangle) \longrightarrow_E (r, |\psi\rangle)} \quad \text{(R-REAL)}$$

$$\frac{}{(x, \sigma, |\psi\rangle) \longrightarrow_E (\sigma(x), |\psi\rangle)} \quad \text{(R-VAR)}$$

$$\frac{}{(\mathit{true}, \sigma, |\psi\rangle) \longrightarrow_E (\mathsf{true}, |\psi\rangle)} \quad \text{(R-TRUE)}$$

$$\frac{}{(\mathit{false}, \sigma, |\psi\rangle) \longrightarrow_E (\mathsf{false}, |\psi\rangle)} \quad \text{(R-FALSE)}$$

$$\frac{(E, \sigma, |\psi\rangle) \longrightarrow_E (\mathsf{true}, |\psi\rangle)}{(\neg E, \sigma, |\psi\rangle) \longrightarrow_E (\mathsf{false}, |\psi\rangle)} \quad \text{(R-NEG}_1\text{)}$$

$$\frac{(E, \sigma, |\psi\rangle) \longrightarrow_E (\mathsf{false}, |\psi\rangle)}{(\neg E, \sigma, |\psi\rangle) \longrightarrow_E (\mathsf{true}, |\psi\rangle)} \quad \text{(R-NEG}_2\text{)}$$

$$\frac{(E_1, \sigma, |\psi\rangle) \longrightarrow_E (v_1, |\psi\rangle), \quad (E_2, \sigma, |\psi\rangle) \longrightarrow_E (v_2, |\psi\rangle)}{(E_1 \ op \ E_2, \sigma, |\psi\rangle) \longrightarrow_E (f_A(op, v_1, v_2), |\psi\rangle)} \quad \text{(R-OP)}$$

$$\frac{(E_1, \sigma, |\psi\rangle) \longrightarrow_E (v_1, |\psi\rangle), \quad (E_2, \sigma, |\psi\rangle) \longrightarrow_E (v_2, |\psi\rangle)}{(E_1 \ bop \ E_2, \sigma, |\psi\rangle) \longrightarrow_E (f_B(bop, v_1, v_2), |\psi\rangle)} \quad \text{(R-BOP)}$$

$$\frac{(E_1, \sigma, |\psi\rangle) \longrightarrow_E (v_1, |\psi\rangle), \quad (E_2, \sigma, |\psi\rangle) \longrightarrow_E (v_2, |\psi\rangle)}{(E_1 \ rel \ E_2, \sigma, |\psi\rangle) \longrightarrow_E (f_R(rel, v_1, v_2), |\psi\rangle)} \quad \text{(R-REL)}$$

4.3.2. Statements. The semantics of statements is given by transition relation $\longrightarrow_S$. An isolated expression is treated as a statement; such a statement can control the flow of execution as it may or may not be executable from the current state, as we shall see, but it has no effect on the state, as shown in rule R-EXP. An assignment statement is defined as having the effect of substituting the value of a variable in the process store. Allocating a new qubit enlarges the overall quantum state and introduces a new internal reference L, which indexes it. Sending $(c!x)$ and receiving $(c?x)$ are similar to assignment, but they are specific to channel variables; an empty channel has the special value null. Applying the unitary operators (*cnot*, H, *ph*, X, Y, Z) involves multiplying the global quantum state by a suitably–dimensioned unitary matrix, and measurement is treated as a source of non–deterministic choice, where the outcome is stored. Note that $P_{v_1,\ldots,v_k}$

denotes the projection operator corresponding to the computational basis measurement of the k qubits.

$$(E, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa, \sigma, |\psi\rangle) \tag{R-EXP}$$

$$\frac{(E, \sigma, |\psi\rangle) \longrightarrow_E (v, |\psi\rangle)}{(x := E, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa, \sigma[x \mapsto v], |\psi\rangle)} \tag{R-ASSN}$$

$$\frac{L \in \mathbb{Q} \text{ is fresh}}{(x := \mathit{newqubit}, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa, \sigma[x \mapsto L], (|\psi\rangle \otimes |0\rangle))} \tag{R-NEW}$$

$$\frac{(x, \sigma, |\psi\rangle) \longrightarrow_E (v, |\psi\rangle)}{(c!x, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa[c \mapsto v], \sigma[x \mapsto \text{null}], |\psi\rangle)} \tag{R-SEND}$$

$$\frac{\kappa(x) = v}{(c?x, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa[c \mapsto \text{null}], \sigma[x \mapsto v])} \tag{R-RECV}$$

$$(\mathit{cnot}\ x_1\ x_2, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa, \sigma, C_{\sigma(x_1),\sigma(x_2)}|\psi\rangle) \tag{R-CNOT}$$

$$(\mathit{qop}\ x, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa, \sigma, U_{\mathit{qop}}|\psi\rangle) \tag{R-QOP}$$

$$\frac{(\mathfrak{IR}(x := \mathit{measure}\ x_1, \ldots, x_k, s), \kappa, \sigma, |\psi\rangle) \longrightarrow_H (\epsilon, \kappa', \sigma', |\psi\rangle), \quad \sigma(x_i) = v_i \text{ for } 1 \leqslant i \leqslant k}{(x := \mathit{measure}\ x_1, \ldots, x_k, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa', \sigma', P_{v_1,\ldots,v_k}|\psi\rangle)} \tag{R-MEASURE}$$

4.3.3. Sequences and Non-deterministic Choices. Transition relation $\longrightarrow_Q$ expresses the effect of sequencing of statements; it should require no further explanation.

$$\frac{(S, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa', \sigma', |\psi'\rangle)}{(S, \kappa, \sigma, |\psi\rangle) \longrightarrow_Q (\epsilon, \kappa', \sigma', |\psi'\rangle)} \tag{R-SEQ$_1$}$$

$$\frac{(S, \kappa, \sigma, |\psi\rangle) \longrightarrow_S (\kappa', \sigma', |\psi'\rangle)}{(S;\ Q, \kappa, \sigma, |\psi\rangle) \longrightarrow_Q (Q, \kappa', \sigma', |\psi'\rangle)} \tag{R-SEQ$_2$}$$

Non–deterministic choices contain several sequences of statements, one of which may be selected for execution. Once the choice has been made, the chosen sequence must be executed to completion. One step of the transition relation $\longrightarrow_H$ is designed to correspond to the execution of a choice (and the first step thereof).

$$\frac{(Q, \kappa, \sigma, |\psi\rangle) \longrightarrow_Q (Q', \kappa', \sigma', |\psi'\rangle)}{(Q, \kappa, \sigma, |\psi\rangle) \longrightarrow_H (Q', \kappa', \sigma', |\psi'\rangle)} \quad \text{(R-CHOICE}_1\text{)}$$

$$\frac{(Q, \kappa, \sigma, |\psi\rangle) \longrightarrow_Q (Q', \kappa', \sigma', |\psi'\rangle)}{(Q \square H, \kappa, \sigma, |\psi\rangle) \longrightarrow_H (Q', \kappa', \sigma', |\psi'\rangle)} \quad \text{(R-CHOICE}_2\text{)}$$

$$\frac{}{(Q \square H, \kappa, \sigma, |\psi\rangle) \longrightarrow_H (H, \kappa, \sigma, |\psi\rangle)} \quad \text{(R-CHOICE}_3\text{)}$$

$$\frac{(Q, \kappa, \sigma, |\psi\rangle) \longrightarrow_Q (\epsilon, \kappa', \sigma', |\psi'\rangle)}{(Q, \kappa, \sigma, |\psi\rangle) \longrightarrow_H (\epsilon, \kappa', \sigma', |\psi'\rangle)} \quad \text{(R-CHOICE}_4\text{)}$$

4.3.4. Guarded Commands. A guarded command is either an explicit non–deterministic choice (the case *if* …*fi*) or a loop (enclosed in *do* …*od*). The rules for guarded commands reduce the former into an instance of the relation $\longrightarrow_H$, and provide the steps for executing a loop. Note that a loop is simply a repetition of a non–deterministic choice, and that executing a loop means transforming it into a command sequence consisting of the non–deterministic choice by itself, followed by another iteration of the whole loop (see rule R-GUARDED$_2$).

There is an upper bound i_{MAX} on the number of repetitions of a loop, which is implementation dependent. Future extensions to the language might include an explicit mechanism for terminating such loops early; there are constructs for this purpose *e.g.* in the language PROMELA (the syntax and functionality of guarded commands used here is inspired by that language and its implementation in the SPIN model checker).

Note that we use a transition relation $\longrightarrow_N$ to reduce a choice H to another choice $H^\sharp$ before performing a step. This is defined and explained in Section 4.4.

$$\frac{(H,\kappa,\sigma,|\psi\rangle) \longrightarrow_N (H^\sharp,\kappa,\sigma,|\psi\rangle), \quad (H^\sharp,\kappa,\sigma,|\psi\rangle) \longrightarrow_H (H',\kappa',\sigma',|\psi'\rangle)}{(H,\kappa,\sigma,|\psi\rangle,i) \longrightarrow_G (H',\kappa',\sigma',|\psi'\rangle,i)} \quad (\text{R-GUARDED}_1)$$

$$\frac{(H,\kappa,\sigma,|\psi\rangle) \longrightarrow_N (H^\sharp,\kappa,\sigma,|\psi\rangle), \quad (H^\sharp,\kappa,\sigma,|\psi\rangle) \longrightarrow_H (H',\kappa',\sigma',|\psi'\rangle)}{(do(H),\kappa,\sigma,|\psi\rangle,i) \longrightarrow_G (H';\ do(H),\kappa',\sigma',|\psi'\rangle,i)} \quad (\text{R-GUARDED}_2)$$

$$\frac{i < i_{MAX}, \quad (H,\kappa,\sigma,|\psi\rangle) \longrightarrow_N (H^\sharp,\kappa,\sigma,|\psi\rangle), \quad (H^\sharp,\kappa,\sigma,|\psi\rangle) \longrightarrow_H (\epsilon,\kappa',\sigma',|\psi'\rangle)}{(do(H),\kappa,\sigma,|\psi\rangle,i) \longrightarrow_G (\epsilon,\kappa',\sigma',|\psi'\rangle,i+1)} \quad (\text{R-GUARDED}_3)$$

$$\frac{i = i_{MAX}}{(do(H),\kappa,\sigma,|\psi\rangle,i) \longrightarrow_G (\epsilon,\kappa,\sigma,|\psi\rangle,0)} \quad (\text{R-GUARDED}_4)$$

4.3.5. Processes. A process C is simply a sequence of commands, including guarded commands and single statements. (Note that we ignore variable declarations and process names here for simplicity.) Executing a step of a process means executing the next statement available. The transition relation $\longrightarrow_C$ expresses the meaning of a process step and is detailed below.

$$\frac{\mathsf{Exe}_S(S,\kappa,\sigma,|\psi\rangle) = \mathsf{true}, \quad (S,\kappa,\sigma,|\psi\rangle) \longrightarrow_S (\kappa',\sigma',|\psi'\rangle)}{(S;\ C,\kappa,\sigma,|\psi\rangle,i) \longrightarrow_C (C,\kappa',\sigma',|\psi'\rangle,i)} \quad (\text{R-COM}_1)$$

$$\frac{(G,\kappa,\sigma,|\psi\rangle,i) \longrightarrow_G (G',\kappa',\sigma',|\psi'\rangle,i')}{(G;\ C,\kappa,\sigma,|\psi\rangle,i) \longrightarrow_C (G';\ C,\kappa',\sigma',|\psi'\rangle,i')} \quad (\text{R-COM}_2)$$

$$\frac{(G,\kappa,\sigma,|\psi\rangle,i) \longrightarrow_G (\epsilon,\kappa',\sigma',|\psi'\rangle,i')}{(G;\ C,\kappa,\sigma,|\psi\rangle,i) \longrightarrow_C (C,\kappa',\sigma',|\psi'\rangle,i')} \quad (\text{R-COM}_3)$$

where n is initialised to 0 in the first application of R-COM$_2$.

4.3.6. Programs. A single step of a program corresponds to a single process transition. At each time instant, there are many possible process choices, *i.e.* many possible process transitions.

Transition relation $\longrightarrow_P$ expresses the execution of a single program step.

$$\frac{(C, \kappa, \sigma, |\psi\rangle, i) \longrightarrow_C (C', \kappa', \sigma', |\psi'\rangle, i')}{(C \mid\mid P, \kappa, \Sigma, |\psi\rangle) \longrightarrow_P (C' \mid\mid P, \kappa', \Sigma', |\psi'\rangle, i')} \quad \text{(R-PROGRAM}_1\text{)}$$

$$\frac{(C', \kappa, \sigma, |\psi\rangle, i) \longrightarrow_C (C'', \kappa', \sigma', |\psi'\rangle, i')}{(C \mid\mid P, \kappa, \Sigma, |\psi\rangle) \longrightarrow_P (C \mid\mid P', \kappa', \Sigma', |\psi'\rangle, i')} \quad \text{(R-PROGRAM}_2\text{)}$$

where $\sigma \in \Sigma$, $\sigma' \in \Sigma'$, $P = \cdots \mid\mid C' \mid\mid \cdots$ and $P' = \cdots \mid\mid C'' \mid\mid \cdots$.

Definition 4.5 (Complete Program Run)**.** *Starting from an empty initial state* $(\kappa_0, \Sigma_0, \mathbf{1})$ *(where* $\kappa_0 = \Sigma_0 = \emptyset$ *and* $\mathbf{1}$ *is a unit vector of* $\mathscr{H}$*) and a program* P*, we define a complete program run as a sequence of transitions*

$$(P, \kappa_0, \Sigma_0, \mathbf{0}) \longrightarrow_P^* (\epsilon, \kappa, \Sigma, |\psi\rangle)$$

It should be noted that a run of a QMCLANG program may terminate prematurely if all remaining statements are non–executable (this means that a run of a program may end in a state where the original program P has not been reduced completely to the empty string ϵ as above). The concept of executability is detailed in the following section.

4.4. The Executability Predicate

At each step during execution of a QMCLANG program, there are several possibilities caused by the interleaving of processes. As we have seen, a single execution step comprises a single statement. However, execution can only occur if the statement in question (and, by extension, the process containing it) is *executable*. Executability is defined formally using a predicate Exe_S for statements. Similar predicates can be defined for the other constructs in the language; the definitions are given in Fig. 4.6. Note that we have used the executability predicates here in the semantic rules.

Communication actions (sending and receiving of variables) can be used to control the synchronisation of processes, since these are performed with statements that are only executable if a

condition is satisfied (*e.g.* a "receive" statement cannot be executed if the channel from which a value is being retrieved is empty). This fact is expressed in Eqs. 4.4 and 4.4.

Note that in QMCLANG there exist *expression–statements*, namely, statements consisting of a single expression which is simply evaluated. This feature, inspired by the language PROMELA, may not seem useful at first, but turns out to be a very helpful mechanism for controlling the interleaving of processes. A boolean expression–statement which is false is simply not executable. In practice this means that one can introduce an arbitrary condition inside a process which blocks that process until the condition is satisfied. One must be careful not to misconstrue expression–statements with expressions that are inside an assignment statement; an assignment is always executable.

Notation 4.1 (Definitions). *We use the symbol* $:=$ *instead of the equality symbol* $=$ *to denote a definition, namely that the expression on the left hand side of this symbol can be replaced by the expression on the right hand side.*

Executability is the mechanism by which conditions in programs are checked, since conditions are simply written as expression–statements which are placed at suitable points in the text of a process. A guarded command comprises a non–deterministic choice between a number of options; each option consists of a sequence of statements which may or may not be executable at a given point during execution. We define the set of *enabled options* inside a choice H as follows.

$$\mathit{Enabled}(H) = \{Q \mid Q \text{ is an option in } H \text{ such that } \mathsf{Exe}_{\mathsf{Q}}(Q)\} \tag{4.2}$$

Definition 4.6 (Enabled Options). *If* $\mathit{Enabled}(H) = \{Q_1, \ldots, Q_k\}$ *then we define* $H^\sharp$ *to be the corresponding syntactic element:*

$$H^\sharp := Q_1 \Box \cdots \Box Q_k$$

This symbol is used to represent the piece of program text consisting only of the enabled options in H.

In order to execute a step inside a non–deterministic choice, the enabledness of the options inside the choice must be checked; this is performed via a simple transition relation $\longrightarrow_N$ which

reduces a choice H to a choice $H^\sharp$ between enabled options.

$$\longrightarrow_N \ : \mathsf{Choice} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H} \hookrightarrow \mathsf{Choice} \times \mathsf{ChanStore} \times \mathsf{Store} \times \mathscr{H}$$

$$\frac{}{(H, \kappa, \sigma, |\psi\rangle) \longrightarrow_N (H^\sharp, \kappa, \sigma, |\psi\rangle)} \quad \text{(R-REDUCE)}$$

Next we turn to the type system of the language.

4.5. Type System

The type system for QMCLANG is given by the rules detailed next. QMCLANG is a *statically typed* language. The presentation of the type system is in the style of [36]. We assume that the reader is familiar with the concepts of typing judgement and type environment.

In the type system there are two main kinds of judgement, one for expressions and types, and one for statements. We write

$$\Gamma \vdash \diamond \tag{4.3}$$

to express the fact that Γ is a well–formed environment. A judgement of the form

$$\Gamma \vdash E : T \tag{4.4}$$

states that the expression E is of type T in environment Γ. Finally, a judgement of the form

$$\Gamma \vdash S \tag{4.5}$$

states that the statement S is well–typed in environment Γ.

The base rules for the type system define how environments are constructed. The form $\Gamma, x : T$ denotes an environment which is extended with the declaration of the variable x of type T.

$$\emptyset \vdash \diamond \qquad \frac{\Gamma \vdash T \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : T \vdash \diamond} \qquad \frac{\Gamma_1, x : T, \Gamma_2 \vdash \diamond}{\Gamma_1, x : T, \Gamma_2 \vdash x : T} \quad \text{(T-EMPTY,T-NEW,T-UNION)}$$

Given a well–formed environment Γ the admissible types are integers, booleans, real numbers, qubits, and channels containing values of these types.

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{integer}} \qquad \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{bool}} \qquad \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{real}} \qquad (\textsc{t-int},\textsc{t-bool},\textsc{t-real})$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{qubit}} \qquad \frac{\Gamma \vdash \diamond \quad \Gamma \vdash T}{\Gamma \vdash \mathit{channel\ of}\ T} \qquad (\textsc{t-qubit},\textsc{t-chan})$$

The following rules give the types of simple expressions.

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash n : \mathit{integer}} \qquad \frac{\Gamma \vdash \diamond}{\Gamma \vdash r : \mathit{real}} \qquad (\textsc{t-exp}_1,\textsc{t-exp}_2)$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{true} : \mathit{bool}} \qquad \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathit{false} : \mathit{bool}} \qquad \frac{\Gamma \vdash E : \mathit{bool}}{\Gamma \vdash \neg E : \mathit{bool}} \qquad (\textsc{t-exp}_3,\textsc{t-exp}_4,\textsc{t-exp}_5)$$

Rules $\textsc{t-exp}_6$ to $\textsc{t-exp}_8$ give the types of composite expressions.

$$\frac{\Gamma \vdash E_1 : \mathit{integer} \quad \Gamma \vdash E_2 : \mathit{integer}}{\Gamma \vdash E_1\ \mathit{op}\ E_2 : \mathit{integer}} \qquad (\textsc{t-exp}_6)$$

$$\frac{\Gamma \vdash E_1 : \mathit{bool} \quad \Gamma \vdash E_2 : \mathit{bool}}{\Gamma \vdash E_1\ \mathit{bop}\ E_2 : \mathit{bool}} \qquad (\textsc{t-exp}_7)$$

$$\frac{\Gamma \vdash E_1 : \mathit{integer} \quad \Gamma \vdash E_2 : \mathit{integer}}{\Gamma \vdash E_1\ \mathit{rel}\ E_2 : \mathit{bool}} \qquad (\textsc{t-exp}_8)$$

Well–typed statements are defined by rules $\textsc{t-stmt}_1$ to $\textsc{t-stmt}_7$.

$$\frac{\Gamma \vdash x : \mathit{qubit}}{\Gamma \vdash x := \mathit{newqubit}} \qquad \frac{\Gamma \vdash x : \mathit{qubit}}{\Gamma \vdash \mathsf{Op}\ x} \qquad (\textsc{t-stmt}_1,\textsc{t-stmt}_2)$$

$$\frac{\Gamma \vdash x_1 : \mathit{qubit} \quad \Gamma \vdash x_2 : \mathit{qubit}}{\Gamma \vdash \mathit{cnot}\ x_1\ x_2} \tag{T-STMT$_3$}$$

$$\frac{\Gamma \vdash x : \mathit{integer} \quad \Gamma \vdash x_i : \mathit{qubit} \text{ for all } i}{\Gamma \vdash x := \mathit{measure}\ x_1, \ldots, x_k} \tag{T-STMT$_4$}$$

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash x := E} \tag{T-STMT$_5$}$$

$$\frac{\Gamma \vdash T \quad \Gamma \vdash c : \mathit{channel\ of}\ T \quad \Gamma \vdash x : T}{\Gamma \vdash c!x} \tag{T-STMT$_6$}$$

$$\frac{\Gamma \vdash T \quad \Gamma \vdash c : \mathit{channel\ of}\ T \quad \Gamma \vdash x : T}{\Gamma \vdash c?x} \tag{T-STMT$_7$}$$

The rules of the type system restrict the set of valid QMCLANG programs as defined by the grammar in Fig. 4.4. The types of all variables in a QMCLANG program must be explicitly declared, although we have omitted type declarations from the abstract syntax for simplicity. The semantics of type declarations are subsumed by the static typing rules T-EMPTY, T-NEW, and T-UNION given in this section.

$$\mathsf{Exe_S} : \mathsf{Stmt} \times \mathsf{State} \to \{\mathsf{true}, \mathsf{false}\}$$
$$\mathsf{Exe_S}(E, \kappa, \sigma, |\psi\rangle) := \mathit{eval}(E, \kappa, \sigma, |\psi\rangle)$$
$$\mathsf{Exe_S}(x := E, \kappa, \sigma, |\psi\rangle) := \mathsf{true}$$
$$\mathsf{Exe_S}(x := \mathit{newqubit}, \kappa, \sigma, |\psi\rangle) := \mathsf{true}$$
$$\mathsf{Exe_S}(x := \mathit{measure}\ x_1, \dots, x_k, \kappa, \sigma, |\psi\rangle) := \forall i \in \{1, \dots, k\}.\sigma(x_i) \neq \mathsf{null}$$
$$\mathsf{Exe_S}(c!x, \kappa, \sigma, |\psi\rangle) := \kappa(c) = \mathsf{null}$$
$$\mathsf{Exe_S}(c?x, \kappa, \sigma, |\psi\rangle) := \kappa(c) \neq \mathsf{null}$$
$$\mathsf{Exe_S}(\mathit{cnot}\ x_1\ x_2, \kappa, \sigma, |\psi\rangle) := \sigma(x_1) \neq \mathsf{null}\ \text{and}\ \sigma(x_2) \neq \mathsf{null}$$
$$\mathsf{Exe_S}(\mathit{qop}\ x, \kappa, \sigma, |\psi\rangle) := \sigma(x) \neq \mathsf{null}$$

$$\mathsf{Exe_Q} : \mathsf{Seq} \times \mathsf{State} \to \{\mathsf{true}, \mathsf{false}\}$$
$$\mathsf{Exe_Q}(S, \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_S}(S, \kappa, \sigma, |\psi\rangle)$$
$$\mathsf{Exe_Q}(S;\ Q, \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_S}\kappa, \sigma, |\psi\rangle(S, \kappa, \sigma, |\psi\rangle)$$

$$\mathsf{Exe_H} : \mathsf{Choice} \times \mathsf{State} \to \{\mathsf{true}, \mathsf{false}\}$$
$$\mathsf{Exe_H}(Q, \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_Q}(Q, \kappa, \sigma, |\psi\rangle)$$
$$\mathsf{Exe_H}(Q \,\square\, H, \kappa, \sigma, |\psi\rangle) := \begin{cases} \mathsf{Exe_Q}(Q, \kappa, \sigma, |\psi\rangle) \\ \text{or } \exists Q' \in H.\mathsf{Exe_Q}(Q', \kappa, \sigma, |\psi\rangle) \end{cases}$$

$$\mathsf{Exe_G} : \mathsf{GCom} \times \mathsf{State} \to \{\mathsf{true}, \mathsf{false}\}$$
$$\mathsf{Exe_G}(H, \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_H}(H, \kappa, \sigma, |\psi\rangle)$$
$$\mathsf{Exe_G}(\mathit{do}(H), \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_H}(H, \kappa, \sigma, |\psi\rangle)$$

$$\mathsf{Exe_C} : \mathsf{Com} \times \mathsf{State} \to \{\mathsf{true}, \mathsf{false}\}$$
$$\mathsf{Exe_C}(S;\ C, \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_S}(S, \kappa, \sigma, |\psi\rangle)$$
$$\mathsf{Exe_C}(G;\ C, \kappa, \sigma, |\psi\rangle) := \mathsf{Exe_G}(G, \kappa, \sigma, |\psi\rangle)$$

FIGURE 4.6. Executability Predicates.

$$\mathit{eval}(n, \kappa, \sigma, |\psi\rangle) := \mathsf{true}$$
$$\mathit{eval}(r, \kappa, \sigma, |\psi\rangle) := \mathsf{true}$$
$$\mathit{eval}(x, \kappa, \sigma, |\psi\rangle) := \begin{cases} \mathsf{true\ if}\ \sigma(x) > 0\ \text{or}\ \sigma(x) = \mathsf{true} \\ \mathsf{false\ otherwise} \end{cases}$$
$$\mathit{eval}(E_1\ \mathit{op}\ E_2, \kappa, \sigma, |\psi\rangle) := \mathsf{true}$$
$$\mathit{eval}(E_1\ \mathit{bop}\ E_2, \kappa, \sigma, |\psi\rangle) := f_B(\mathit{eval}(E_1, \kappa, \sigma, |\psi\rangle), \mathit{eval}(E_2, \kappa, \sigma, |\psi\rangle))$$
$$\mathit{eval}(E_1\ \mathit{rel}\ E_2, \kappa, \sigma, |\psi\rangle) := f_R(\mathit{eval}(E_1, \kappa, \sigma, |\psi\rangle), \mathit{eval}(E_2, \kappa, \sigma, |\psi\rangle))$$
$$\mathit{eval}(\mathit{true}, \kappa, \sigma, |\psi\rangle) := \mathsf{true}$$
$$\mathit{eval}(\mathit{false}, \kappa, \sigma, |\psi\rangle) := \mathsf{false}$$
$$\mathit{eval}(\neg E, \kappa, \sigma, |\psi\rangle) := \begin{cases} \mathsf{true\ if}\ \mathit{eval}(E, \kappa, \sigma, |\psi\rangle) = \mathsf{false} \\ \mathsf{false\ if}\ \mathit{eval}(E, \kappa, \sigma, |\psi\rangle) = \mathsf{true} \end{cases}$$

FIGURE 4.7. Definition of an evaluation function for expressions. This is used for defining executability of expression–statements.

4.6. The EQPL and QCTL Specification Logics

The properties of quantum protocols which we are interested in reasoning about are specific to quantum states (*e.g.* which qubits are 'active' in a given state, which qubits are entangled). We are also interested in outcomes of different measurements, and the way in which the values of classical variables evolve. For this purpose we have chosen to use Exogenous Quantum Propositional Logic (EQPL) and its temporal extension, Quantum Computation Tree Logic (QCTL). EQPL is a subset of QCTL [12], although the former has different variants, appearing in [96–98].

We will present here the syntax and semantics of EQPL as defined in [96], giving a satisfaction relation for the logic while relating it to the semantics of our modelling language. The syntax and semantics of QCTL is detailed in Section 4.6.2.

With the modelling language and specification logic formally defined, we will have an elegant framework with which to produce uniform descriptions of quantum protocols and their properties.

4.6.1. EQPL: Exogenous Quantum Propositional Logic. EQPL was designed so as to enable reasoning about systems comprising a finite set of qubits as well as a classical state. The main novelty of this logic is that it uses *combinations* of classical valuations (which provide the semantics of classical propositional logic) to give semantics to 'quantum formulae'. In other words, propositional logic is extended to produce quantum propositional logic by taking combinations of models of the former to build models of the latter; the authors call this the *exogenous approach to enriching logics*. In [97], classical propositional logic is first generalised to incorporate probability, leading to the logic EPPL, and is then further generalised (by defining mathematical structures embodying the postulates of quantum mechanics) to EQPL, which we now describe.

The formulae of EQPL [98] allow one to reason about the state of individual qubits, and involve usual logical connectives such as negation and implication. There are two levels of formulae: *classical formulae,* which hold only if all valuations in a state satisfy them, and *quantum formulae,* which are essentially logical combinations of classical formulae. The syntax of EQPL is given in Fig. 4.8.

$$
\begin{array}{rcl}
\alpha & := & \mathbf{p_k} \mid \bot \mid (\alpha \Rightarrow \alpha) \\
t & := & x \mid r \mid (\int \alpha) \mid (t + t) \mid (t\, t) \mid \mathsf{Re}(u) \mid \mathsf{Im}(u) \mid \mathsf{arg}(u) \mid |u| \\
u & := & z \mid |\top\rangle_{FA} \mid (t + it) \mid te^{it} \mid \overline{u} \mid (u + u) \mid (u\, u) \mid (\alpha \triangleright u;\, u) \\
\gamma & := & \alpha \mid (t \leqslant t) \mid [F] \mid (\gamma \sqsupset \gamma)
\end{array}
$$

FIGURE 4.8. The syntax of EQPL (from [98]).

The starting point for defining the semantics of EQPL is the establishment of a denumerable set of propositional constants $\mathsf{qB} = \{\mathbf{p_k} : k \in \mathbb{N}\}$, where we associate each $\mathbf{p_k}$ to a single qubit in the protocol under consideration. A *classical valuation* v is a truth value assignment for all propositional constants, *i.e.* a mapping

$$v : \mathsf{qB} \to \{0, 1\}$$

Given a set of classical valuations V, it is possible to construct a Hilbert space $\mathscr{H} = H(V)$ of which the $v \in V$ constitute an orthonormal basis. In the definition of EQPL, a set of *admissible valuations* V is established, and then the Hilbert space $\mathscr{H}(V)$ is constructed by making each vector $v \in V$ into a basis vector of $\mathscr{H}(V)$; in other words, $\mathscr{H}(V)$ is spanned by the vectors in V. For all intents and purposes we take V to consist of all 2^n valuations for the n propositional symbols; each of these valuations corresponds to a computational basis vector of the Hilbert space associated with an n–qubit system. A *quantum valuation* is a unit vector of $\mathscr{H}$. A quantum state $|\psi\rangle$ is none other than a quantum valuation, and so can be expressed as a linear combination of classical valuations (which are computational basis states). An n–qubit state $|\psi\rangle$ is specified by 2^n complex numbers $\{\ \langle v|\psi\rangle \mid v \subset 2^{\mathsf{qB}}\ \}$. The complex number $\langle v|\psi\rangle$ (termed a 'logical amplitude') is the projection of the unit vector ψ on the basis vectors $|v\rangle$.

Example 4.1. *For a two–qubit system we have four possible valuations for the two propositional constants* $\mathbf{p_0}$ *and* $\mathbf{p_1}$*. These valuations correspond exactly to the four computational basis vectors* $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$*. We have the set* $V = \{v_{00}, v_{01}, v_{10}, v_{11}\}$ *where:*

$$v_{00} : \mathbf{p_0} \mapsto 0, \mathbf{p_1} \mapsto 0 \tag{4.6}$$

$$v_{01} : \mathbf{p_0} \mapsto 0, \mathbf{p_1} \mapsto 1 \tag{4.7}$$

$$v_{10} : \mathbf{p_0} \mapsto 1, \mathbf{p_1} \mapsto 0 \tag{4.8}$$

$$v_{11} : \mathbf{p_0} \mapsto 1, \mathbf{p_1} \mapsto 1 \tag{4.9}$$

The semantics of EQPL is given over a *quantum interpretation structure;* this structure is defined as follows (the reader can refer to [97] for the details of the particular notations used).

Definition 4.7. ***A quantum interpretation structure*** *is a tuple*

$$\mathbf{w} = (V, \mathscr{S}, |\psi\rangle, \nu)$$

where:

- V *is a nonempty subset of* 2^{qB} *(for our purposes we take* $V = 2^{\mathsf{qB}}$*)*
- $\mathscr{S}$ *is a finite partition of* qB *(this is actually an entanglement partition, containing those qubits which are not entangled with the rest of the state)*
- $|\psi\rangle = \{|\psi\rangle_{[R]}\}_{R \in \bigcup S}$ *where each* $|\psi\rangle_{[R]}$ *is a unit vector of* $\mathscr{H}_{[R]}$ *and such that:*
  (1) $|\psi\rangle_{[\emptyset]} = \mathsf{exp}(i0)$
  (2) $|\psi\rangle_{[R]} = \bigotimes_{S \in \mathscr{S}, S \subseteq R} |\psi\rangle_{[S]}$ *for each nonempty* $R \in \bigcup S$
  (3) $|\psi\rangle_{[S]}$ *is non–factorisable for each* $S \in \mathscr{S}$
  (4) $\langle v|\psi\rangle_{[\mathsf{qB}]} = 0$ *if* $v \notin V$
- $\nu : \{\nu_{FA}\}_{F \subseteq_{fin} \mathsf{qB}, A \subseteq F}$ *where each* $\nu_{FA} \in \mathbb{C}$ *and* $\nu_{FA} = \langle v_A^F|\psi\rangle_{[F]}$ *if* $F \in \bigcup S$.

Note that a quantum interpretation structure consists of a quantum state as well as additional information about its structure, including the entanglement partition $\mathscr{S}$ and the coefficients ν. In addition to this, a probability space is defined and is used to assign meaning to terms in the logic representing measurement of a set F of qubits; it suffices to say here that this space comprises a

probability measure $\mu^F_{\mathbf{w}}$, as this symbol is used in the definition of the satisfaction relation for the logic.

Given a set S of qubit symbols and a set V of valuations, the *extent* at V of a classical formula α over S is defined as $|\alpha|^S_V = \{v \in V_{[S]} : v \Vdash \alpha\}$. The satisfaction relation for the logic also makes use of the concept of an *assignment* ρ, which is simply a store defined for all variables of interest (where x denotes a real variable and z a complex variable), so that $\rho(x) \in \mathbb{R}$, $\rho(z) \in \mathbb{C}$.

A classical formula α evaluates to true if and only if α is true for every valuation corresponding to a basis vector that appears in the quantum state with a non-zero coefficient. For example, if the quantum state is $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ then $\mathsf{q}_0 \Leftrightarrow \mathsf{q}_1$ is true but $\mathsf{q}_0 \wedge \mathsf{q}_1$ is false. Note that commonly encountered connectives such as $\Leftrightarrow$ and $\wedge$ can be defined (at both the classical and quantum level) in terms of falsum ($\bot$) and implication ($\Rightarrow$ and $\sqsupset$ respectively).

A quantum formula γ is either the constant $\bot$, a comparison formula $t_1 \leqslant t_2$, a formula involving a quantum connective such as $\gamma_1 \sqsupset \gamma_2$, a classical formula α, or an entanglement formula $[\mathsf{q}_i, \mathsf{q}_j, \ldots]$. The formula $t_1 \leqslant t_2$ is evaluated by evaluating t_1 and t_2 to real values. Quantum connectives are defined by the usual truth tables for boolean operators applied to the values of the operands.

Denotation of terms:

$[\![x]\!]_{\mathbf{w},\rho} = \rho(x)$

$[\![r]\!]_{\mathbf{w},\rho} = r$

$[\![(\int \alpha)]\!]_{\mathbf{w},\rho} = \mu_{\mathbf{w}}(|\alpha|_V)$

$[\![z]\!]_{\mathbf{w},\rho} = \rho(z)$

$[\![|\top\rangle_{G,A}]\!]_{\mathbf{w},\rho} = \nu_{G,A}$

$$[\![(\alpha \triangleright u;\ u)]\!]_{\mathbf{w},\rho} = \begin{cases} [\![u_1]\!]_{\mathbf{w},\rho} \text{ if } |\alpha|_V = V \\ [\![u_2]\!]_{\mathbf{w},\rho} \text{ otherwise} \end{cases}$$

$[\![t_1 + it_2]\!]_{\mathbf{w},\rho} = [\![t_1]\!]_{\mathbf{w},\rho} + \delta[\![t_2]\!]_{\mathbf{w},\rho}$

Satisfaction of quantum formulae:

$\mathbf{w}, \rho \Vdash \alpha$ iff $v \Vdash \alpha$ for every $v \in V$

$\mathbf{w}, \rho \Vdash (t_1 \leqslant t_2)$ iff $[\![t_1]\!]_{\mathbf{w},\rho} \leqslant [\![t_2]\!]_{\mathbf{w},\rho}$

$\mathbf{w}, \rho \Vdash [F]$ iff $F \in \bigcup \mathscr{S}$

$\mathbf{w}, \rho \Vdash (\gamma_1 \supset \gamma_2)$ iff $\mathbf{w}, \rho \nVdash \gamma_1$ or $\mathbf{w}, \rho \Vdash \gamma_2$

FIGURE 4.9. The semantics of EQPL.

4.6.2. QCTL: Quantum Computation Tree Logic. We have implemented quantum computation tree logic (QCTL [12]) in the QMC model checker; QCTL adds the usual temporal connectives (AF, EF, EU) of computation tree logic (CTL) [50] to EQPL.

The syntax of temporal formulae is given below.

$$\theta ::= \gamma \mid \theta \sqsupset \theta \mid (\mathsf{EX}\theta) \mid ([\theta\ \mathsf{EU}\ \theta]) \mid (\mathsf{AF}\theta) \tag{4.10}$$

The temporal operators EX, EU, AF allow us to build formulae which express, with reference to the tree of all possible program executions, in which paths and in which states a quantum formula should hold:

- EX θ expresses the notion "there exists a path (*i.e.* a run of the program) in which, if θ holds in a state of this path, it also holds in the immediately following state (the next state)."
- $[\theta_1\ \mathsf{EU}\ \theta_2]$ expresses the fact that "there exists a path (*i.e.* a run of the program) in which θ_1 holds for a sequence of states, and immediately after θ_2 holds until the end of the run."
- AF θ expresses the fact that "for all runs of the program, there is a future state in which θ is satisfied."

Other common CTL operators, such as EF, can be expressed as combinations of the above operators (see [12]).

$\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} \gamma$ iff $\mathbf{w}, \rho \Vdash \gamma$
$\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} (\theta_1 \supset \theta_2)$ iff $\mathscr{T}, \mathbf{w}, \rho \nVdash_{\text{QCTL}} \theta_1$ or $\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} \theta_2$
$\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} \mathsf{EX}\theta$ iff $\mathscr{T}, \mathbf{w}', \rho' \Vdash_{\text{QCTL}} \theta$ for some $(\mathbf{w}, \rho) \in S$ such that $((\mathbf{w}, \rho), (\mathbf{w}', \rho')) \in R$
$\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} \mathsf{AF}\theta$ iff for all paths $(\mathbf{w}_1, \rho_1), (\mathbf{w}_2, \rho_2), \ldots$ with $\mathbf{w}_1 = \mathbf{w}, \rho_1 = \rho$ there is a $i \geqslant 1$ such that $\mathscr{T}, \mathbf{w}_i, \rho_i \Vdash_{\text{QCTL}} \theta$
$\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} \mathsf{E}[\theta_1 \mathsf{U} \theta_2]$ iff there is a path $(\mathbf{w}_1, \rho_1), (\mathbf{w}_2, \rho_2), \ldots$ with $\mathbf{w}_1 = \mathbf{w}, \rho_1 = \rho$ such that for some $i \geqslant 1$, $\mathscr{T}, \mathbf{w}, \rho \Vdash_{\text{QCTL}} \theta_2$ and $\mathscr{T}, \mathbf{w}_j, \rho_j \Vdash_{\text{QCTL}} \theta_1$ for $1 \leqslant j < i$.

FIGURE 4.10. The semantics of QCTL.

The semantics of QCTL are defined over a *quantum Kripke structure*, defined thus:

Definition 4.8. *A finite quantum Kripke structure over the set of qubits* qB *and variables* $\mathbf{X}$ *is a pair* $\mathscr{T} = (S, R)$ *where:*

- $S \subset \mathscr{H}_{\mathsf{qB}} \times \mathbb{R}^{\mathbf{X}}$ *is a set of pairs* (ψ, ρ) *such that* ψ *is a unit vector in* $\mathscr{H}_{\mathsf{qB}}$ *and* ρ *is an assignment, and*
- $R \subseteq S \times S$ *is a relation such that for any* $(\psi, \rho) \in S$*, there is a* $(\psi', \rho') \in S$ *such that* $((\psi, \rho), (\psi', \rho')) \in R$.

If S *is finite then* $\mathscr{T}$ *is said to be finite, and* $|S|$*, which is the number of elements in* S*, is said to be the size of* $\mathscr{T}$.

4.7. Interpreting QCTL Formulae over QMCLANG Models

The key ingredients for evaluating EQPL formulae are the quantum state $|\psi\rangle$and the classical assignment function ρ for variables. Entanglement information about the quantum state is also needed (the sets V and S in the EQPL paper [96]), but as we shall see this information can be extracted from the quantum state directly, at least in the cases of interest. The classical assignment ρ corresponds to the union of all classical stores $\sigma \in \Sigma$ in our setting.

We consider the graph that results from joining all possible runs of a QMCLANG program, referred to as the *program tree* or *execution tree*, and use it *in lieu* of the quantum Kripke structure which the authors of QCTL have defined. The elements of the program tree $\mathscr{T}$ corresponding to a QMCLANG program P are states $(|\psi\rangle, \Sigma, \kappa)$ (see Definition 4.3) linked by a relation $\Rightarrow$, such that:

$$\frac{\text{For all transitions } (P, \kappa, \Sigma, |\psi\rangle) \longrightarrow_P (P', \kappa', \Sigma', |\psi'\rangle)}{(|\psi\rangle, \Sigma, \kappa) \Rightarrow (|\psi'\rangle, \Sigma', \kappa')} \tag{4.11}$$

We are able to evaluate QCTL formulae over a QMCLANG program tree $\mathscr{T}$ computationally, as detailed in the next chapter. Since a program tree contains *all* possible runs of a program, our method is exhaustive and corresponds essentially to an explicit model checking procedure.

4.8. Protocols and Properties Expressible Within the QMCLANG–QCTL Combined Framework

In a nutshell, the framework for verification described in this chapter and extended with a full implementation in the next provides capability to express protocols involving Clifford group operators and computational basis measurements on pure states and to check the following classes of property:

- properties described by simple EQPL formulae
- entanglement of a qubit in any state of the protocol with the other qubits in that state
- "equality" of qubit states at different times during a computation, subject to the requirement that the qubits in question are disentangled from the other qubits in the overall states at those times
- properties described by temporal QCTL formulae.

4.9. Concluding Remarks

In this chapter we have described a specification language for quantum protocols which includes primitives that model the construction, manipulation and transmission of qubits. The operations permitted in the language, which is tentatively referred to as QMCLANG, are Clifford operations and so the scope of the language is restricted to protocols expressible within the stabilizer formalism. This is the setting in which we will operate.

We have presented a formal operational semantics and a static type system for QMCLANG. The language models concurrency, asynchronous communication via named channels and involves dynamic allocation of qubit variables. Aspects of the semantics are inspired by the language PROMELA (especially the construct referred to here as an expression–statement, and the attendant definition of executability). The semantic definitions here are completely original work, and even for PROMELA only a partial semantic definition exists [109] which is not as detailed as ours.

The logics EQPL (exogenous quantum propositional logic) and QCTL (quantum computation tree logic) have been discussed and their semantics given. Our implementation of the QMC model

checker (see Chapter 5) uses an enriched version of QCTL (in particular, a version of QCTL which treats classical formulae as a subset of quantum formulae, and includes amplitudes and entanglement formulae as in the original EQPL) for the specification of properties to be verified.

We believe that QMCLANG is apt for the description of quantum protocols since it has an intuitive, procedural syntax. The development and use of QMCLANG does not preclude possible future extensions to the QMC model checker which would allow the use of other formalisms. In particular, it is envisaged that the quantum process algebra CQP [61] could be used instead; however, CQP has a richer syntax and would require extending the existing implementation significantly.

In the next chapter, we will discuss the functionality of the QMC tool and the algorithms which it implements.

CHAPTER

5

IMPLEMENTATION

One man's constant is another man's variable.

— Alan Perlis

The core part of this work is the development of a dedicated model checker for quantum protocols. In previous chapters we have alluded to the QMC tool several times; here we will describe QMC and aspects of its implementation. The implementation of QMC took up the greatest part of the research work described, and the resulting program has around 100,000 lines of Java code overall[1].

[1] The JavaNCSS metric was used to determine the size of the program, which comprises the QMC front–end (30,833 LOC) and the "StabSim" component (58,442 LOC). These measurements were made on 06/07/2008, and the program has grown in size since this date.

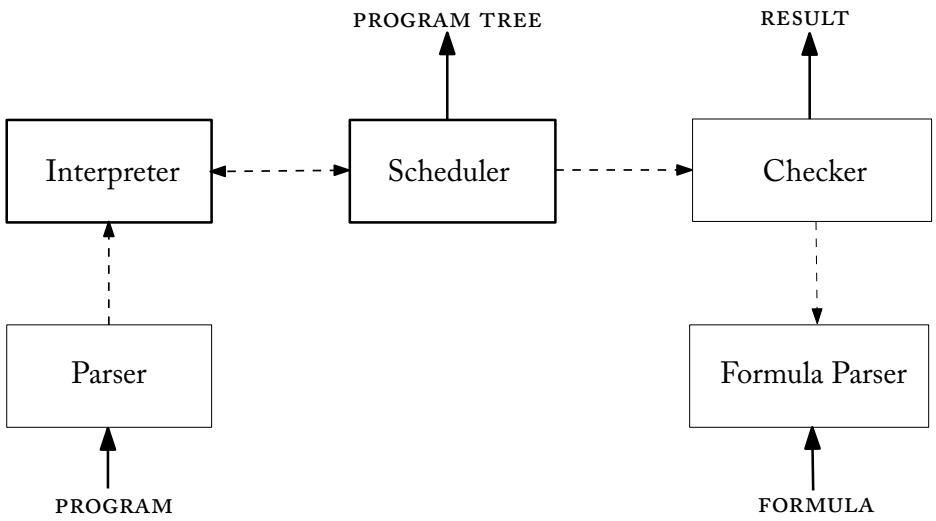


FIGURE 5.1. QMC's overall structure.

5.1. QMC Tool Description

QMC is a tool that automatically explores all possible behaviours arising from a protocol description expressed in QMCLANG and enables QCTL properties to be checked over the resulting structure.

A protocol model consists of definitions of one or more processes. The commands performed by these processes are interleaved, in order to simulate concurrent execution. Nondeterministic choices are resolved in all possible ways, producing a tree of all possible executions of the protocol.

The QMC tool has three main components: (1) a *process scheduler*, (2) a *language interpreter*, and (3) a *model checker*. The role of component (1) is essentially to perform the tasks described in the previous paragraph. The language interpreter handles the execution of individual commands and keeps track of the overall classical and quantum state at each step. Finally, the verifier is responsible for evaluating QCTL formulae over the structure generated by (1) and (2). This structure is depicted in Fig. 5.1.

QMC has a graphical user interface which includes a user–friendly editor for models and properties (see Fig. 5.2).

Given a model and a QCTL formula for verification, QMC outputs the following:

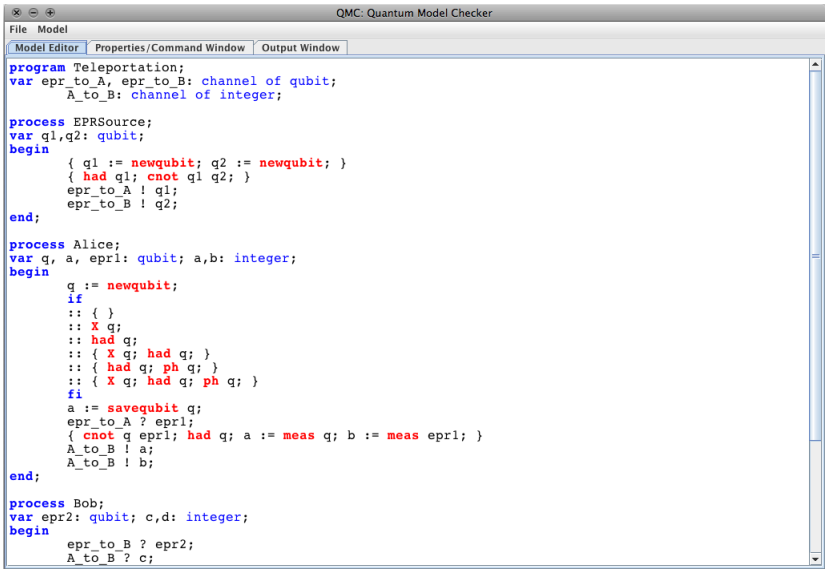


FIGURE 5.2. The Graphical User Interface of QMC.

- a summary of the files provided as input and basic feedback on the model provided, including the number of states and the total number of runs,
- a verification result for the formula provided,
- a list of states (if any) that serve as counter-examples to the formula,
- and a prompt which allows the user to track individual states and trace runs of interest, as well as to verify further formulae.

QMC also provides an interface to the DOT graph layout tool [59], and the ZGRVIEWER application [126], which can display the program execution tree.

The remainder of this chapter describes in turn various implementation aspects, and notably the specific algorithms used in each of the components of QMC as presented in Fig. 5.1.

5.2. Parser Implementation

The inputs to QMC are QMCLANG source files and property description files consisting of EQPL/QCTL formulae and various instructions. These inputs are passed to suitable parsers, one for QMCLANG models and one for formulae. Both parsers have been implemented using SableCC [58], which is a LALR(1) parser generator.

The benefit of SableCC is the flexibility to write many different "tree–walker" classes (in Java) to traverse the parse tree of an input program. QMC includes a tree–walker for interpreting each step in a QMCLANG program, a separate tree–walker for evaluating state formulae (indeed, several of these, one for each class of formula in EQPL), another for evaluating temporal formulae.

Note that the grammar for QMCLANG presented in Chapter 4 is ambiguous. This ambiguity has been resolved by using suitable transformations in the parser grammar.

5.3. Scheduler Implementation

The purpose of the scheduler in QMC is to simulate the interleaving of all executable processes in a given input model, at each step of the simulation. The result of the scheduler–interpreter interaction is the generation of an execution tree (see Section 4.7) which is passed to the model checker.

The scheduler needs to store and update several variables as execution of a model proceeds:

- the "program counter" for each process, *i.e.* the current command (a guarded command, a single statement or atomic block) to be executed,
- the current state of all variables and the current state of the tableau,
- the current position in the execution tree so as to store the result of the next execution,
- the current option and current statement within that option if executing a non–deterministic choice,
- the index of the current loop iteration, which has to be less than an implementation–dependent upper bound i_{MAX} (see rules R-GUARDED$_3$ and R-GUARDED$_4$ in Chapter 4),
- the number of processes which have completed,
- the number of processes which are blocked (non–executable),
- the total number of processes.

The implementation of the scheduler is lengthy and complex, as it has to handle a large number of possibilities, including explicit and implicit non–determinism in models. As we have seen in Chapter 4, we treat indeterminate quantum measurements as being equivalent to a non–deterministic choice between measurement outcomes (see also Eq. 4.1).

5.4. Interpreter Implementation

The purpose of the interpreter is to perform the computational steps corresponding to the execution of a QMCLANG statement. The interpreter is invoked by the scheduler as execution proceeds and is responsible for invoking the Aaronson–Gottesman algorithm (detailed next) whenever a quantum operation or measurement is performed.

5.4.1. Simulation Algorithm. At every point during execution of a model QMC keeps track of $(\psi, \sigma, \gamma_1, \ldots, \gamma_n)$ (for all n processes) where $|\psi\rangle$ in particular is the overall quantum state. The quantum state $|\psi\rangle$ is represented internally in an implicit way: rather than storing the so-called *state vector representation* of $|\psi\rangle$ (which grows exponentially in length as a function of the total number of qubits in $|\psi\rangle$), we use the *stabilizer array representation,* which is a binary representation of the set of Pauli operators that fix (or stabilize) $|\psi\rangle$. Using the stabilizer array representation, we gain significant computational benefits in terms of both space and time when simulating a given protocol, given that simulation of stabilizer circuits is performed using a polynomial time algorithm [1], and the representation of the state grows polynomially with the number of total qubits. Using this approach, the application of each unitary operator or measurement involves updating the stabilizer array by applying suitable row or column operations. It is worth reminding the reader here that only Clifford group gates may ever be applied; clearly it would not be possible to update the tableau representation if a gate outside this group were to be used.

Consider the five–qubit basis state $|\psi_1\rangle = |00000\rangle$. The operators $ZIIII$, $IZIII$, $IIZII$, $IIIZI$, $IIIIZ$ all stabilize $|\psi_1\rangle$ since $Z|0\rangle = |0\rangle$:

$$
\begin{aligned}
(Z \otimes I \otimes I \otimes I \otimes I)\,|00000\rangle &= |00000\rangle \\
(I \otimes Z \otimes I \otimes I \otimes I)\,|00000\rangle &= |00000\rangle \\
(I \otimes I \otimes Z \otimes I \otimes I)\,|00000\rangle &= |00000\rangle \\
(I \otimes I \otimes I \otimes Z \otimes I)\,|00000\rangle &= |00000\rangle \\
(I \otimes I \otimes I \otimes I \otimes Z)\,|00000\rangle &= |00000\rangle
\end{aligned}
$$

We can represent the state $|\psi_1\rangle$ uniquely by writing the list of these particular operators, which generate its stabilizer group. To store the generators of a stabilizer group in a computer program we use a *check matrix* [111]. Each line of the check matrix is a boolean array $\{t_i\}$ of length $2n+1$ which represents one generator. To represent the stabilizer operator

$$P = \bigotimes_{i=1}^{n} P_i = P_1 \otimes \cdots \otimes P_n, \text{ where } P_i \in \{I, X, Y, Z\} \tag{5.1}$$

we set the values of t_i as follows:

$$t_i \leftarrow 0 \text{ and } t_{i+n} \leftarrow 0 \text{ if } P_i = I$$

$$t_i \leftarrow 1 \text{ and } t_{i+n} \leftarrow 0 \text{ if } P_i = X$$

$$t_i \leftarrow 1 \text{ and } t_{i+n} \leftarrow 1 \text{ if } P_i = Y$$

$$t_i \leftarrow 0 \text{ and } t_{i+n} \leftarrow 1 \text{ if } P_i = Z$$

The value of the last entry, t_{2n+1}, is set to 1 to represent a positive overall sign or to 0 to represent a negative overall sign. In practice the last entry is actually a pair of bits, as there are four possible global phases ($+1$, -1, $+i$ and $-i$).

Thus, for example, the generating set of the stabilizer group of $|\psi_1\rangle$ above is represented using the following check matrix:

$$\mathcal{M}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 1 \end{bmatrix} \tag{5.2}$$

Aaronson and Gottesman [1] use a generalised version of the check matrix which they refer to as a *tableau*. A tableau corresponding to a stabilizer state is an augmented check matrix containing a set of *destabilizer generators* atop the stabilizer generators. The stabilizer generators for a

quantum state $|\psi\rangle \in \mathscr{H}^n$ generate a subset of the Pauli group $\mathscr{P}^n$; when these are taken together with the corresponding destabilizer generators, they generate no longer a subset but the full Pauli group $\mathscr{P}^n$. The destabilizer generators are necessary in the simulation algorithm to enable more efficient computation of the effect of a measurement on the quantum state.

Thus the representation of a quantum state which is used by the simulation algorithm is a matrix of the form shown in Figure 5.3.

$$T = [T_{i,j}] = \underbrace{\left[\begin{array}{ccc:c} T_{1,1} & \cdots & T_{1,n} & T_{1,n+1} \\ \vdots & \ddots & \vdots & \vdots \\ T_{n,1} & \cdots & T_{n,n} & T_{n,n+1} \\ \hdashline T_{n+1,1} & \cdots & T_{n+1,n} & T_{n+1,n+1} \\ \vdots & \ddots & \vdots & \vdots \\ T_{2n,1} & \cdots & T_{2n,n} & T_{2n,n+1} \\ \hdashline T_{2n+1,1} & \cdots & T_{2n+1,n} & T_{2n+1,n+1} \end{array}\right]}_{n \text{ columns}\quad \text{phase}} \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} \text{Destabilizer Rows } (n) \\ \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} \text{Stabilizer Rows } (n) \\ \left.\vphantom{\begin{array}{c}a\end{array}}\right\} \text{Scratch Row} \end{array}$$

FIGURE 5.3. The structure of a stabilizer tableau.

Each row in the matrix of Fig. 5.3 corresponds to a Pauli operator of dimension n. In particular, $T_{i,j} \in \{X, Y, Z, I\}$ and the operator in row i is the tensor product of the row entries (with the overall phase factor in front):

$$Op_i := T_{i,n+1} \cdot T_{i,1} \otimes \cdots \otimes T_{i,n}$$

The binary representation of T is a matrix $\mathcal{T} = [\mathcal{T}_{i,j}]$ which takes the form shown in Fig. 5.4.

The Clifford group gates, which are the only operations we will allow in protocol models, can be expressed as boolean functions of the entries in the check matrix. We assume that the check matrix is $\mathcal{T} = [\mathcal{T}_{i,j}]$, where $\mathcal{T}_{i,j}$ represents the entry in row i, column j. We denote the bitwise exclusive OR by $\oplus$ and conjunction by $\cdot$.

Using our notation for the binary tableau, the Aaronson–Gottesman simulation algorithm may be stated as shown in Fig. 5.5. Note that we have used a single symbol $\mathcal{T}_{i,j}$ to denote all matrix

$$\mathcal{T} = [\mathcal{T}_{i,j}] = \left[\begin{array}{ccc:ccc:c} \mathcal{T}_{1,1} & \cdots & \mathcal{T}_{1,n} & \mathcal{T}_{1,n+1} & \cdots & \mathcal{T}_{1,2n} & \mathcal{T}_{1,2n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathcal{T}_{n,1} & \cdots & \mathcal{T}_{n,n} & \mathcal{T}_{n,n+1} & \cdots & \mathcal{T}_{n,2n} & \mathcal{T}_{n,2n+1} \\ \hdashline \mathcal{T}_{n+1,1} & \cdots & \mathcal{T}_{n+1,n} & \mathcal{T}_{n+1,n+1} & \cdots & \mathcal{T}_{n+1,2n} & \mathcal{T}_{n+1,2n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathcal{T}_{2n,1} & \cdots & \mathcal{T}_{2n,n} & \mathcal{T}_{2n,n+1} & \cdots & \mathcal{T}_{2n,2n} & \mathcal{T}_{2n,2n+1} \\ \hdashline \mathcal{T}_{2n+1,1} & \cdots & \mathcal{T}_{2n+1,n} & \mathcal{T}_{2n+1,n+1} & \cdots & \mathcal{T}_{2n+1,2n} & \mathcal{T}_{2n+1,2n+1} \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} \text{Destabilizer Rows } (n) \\ \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} \text{Stabilizer Rows } (n) \\ \left.\vphantom{\begin{array}{c}a\end{array}}\right\} \text{Scratch Row} \end{array}$$

$$\underbrace{\qquad\qquad}_{\text{X-part } (n \text{ columns})} \quad \underbrace{\qquad\qquad}_{\text{Z-part } (n \text{ columns})} \quad \underbrace{\qquad}_{\text{phase}}$$

FIGURE 5.4. The binary tableau representation of a quantum state.

elements, while Aaronson and Gottesman denote the "X part" of the tableau (*i.e.* elements $\mathcal{T}_{1,1}$ to $\mathcal{T}_{2n,n}$) by $x_{i,j}$, the "Z part" of the tableau (*i.e.* $\mathcal{T}_{1,n}$ to $\mathcal{T}_{2n,2n}$) by $z_{i,j}$, and the phase elements $T_{1,2n+1}$ to $T_{2n,2n+1}$ by r_i.

Note that the Pauli group operation is implemented as a special function `rowsum(h,i)` which multiplies row i onto row h while taking into account the overall phase factor. This function is detailed in [1].

5.4.1.1. *Complexity of Simulating Stabilizer Circuits.* According to [1], the problem of simulating stabilizer circuits is complete for the classical complexity class $\oplus L$ (parity-L). According to [2], the complexity class $\oplus L$ (Parity L) has the same relation to class L as $\oplus P$ has to P. Complexity class L is defined below.

Definition 5.1 (Complexity class L: Logarithmic Space)**.** *The class of decision problems solvable by a Turing machine restricted to use an amount of memory logarithmic in the size of the input, n. (The input itself is not counted as part of the memory.)* L *contains* $NC1$ *and is contained in generalizations including* NL*,* $L/poly$*,* SL*,* RL*,* $\oplus L$*, and* $ModkL$*.*

It is important to note that the efficiency of the simulation is due to the compactness of the binary representation of the tableau, and the ability to express quantum operators and measurements as simple binary operations on the tableau. In our work it is often necessary to convert the tableau representation to the more usual state vector representation of quantum states (as used in

Algorithm AARONSON-GOTTESMAN($\mathcal{T}$, *operation*, *arguments*)

1. **for all** rows $1 \leqslant i \leqslant 2n$ **do**
2. **if** applying the CNot gate from qubit a to qubit b **then**
3. Set $\mathcal{T}_{i,(2n+1)} \leftarrow \mathcal{T}_{i,(2n+1)} \oplus \mathcal{T}_{i,a} \cdot \mathcal{T}_{i,(n+b)} \cdot (\mathcal{T}_{i,b} \oplus \mathcal{T}_{i,(n+a)} \oplus 1)$.
4. Set $\mathcal{T}_{i,b} \leftarrow \mathcal{T}_{i,b} \oplus \mathcal{T}_{i,a}$.
5. Set $\mathcal{T}_{i,(n+a)} \leftarrow \mathcal{T}_{i,(n+a)} \oplus \mathcal{T}_{i,(n+b)}$.
6. **else if** applying the Hadamard gate to qubit a **then**
7. Set $\mathcal{T}_{i,(2n+1)} \leftarrow \mathcal{T}_{i,(2n+1)} \oplus \mathcal{T}_{i,a} \cdot \mathcal{T}_{i,(n+a)}$.
8. Swap $\mathcal{T}_{i,a}$ and $\mathcal{T}_{i,(n+a)}$.
9. **else if** applying the Phase $(\frac{\pi}{4})$ gate to qubit a **then**
10. Set $\mathcal{T}_{i,(2n+1)} \leftarrow \mathcal{T}_{i,(2n+1)} \oplus \mathcal{T}_{i,a} \cdot \mathcal{T}_{i,(n+a)}$.
11. Set $\mathcal{T}_{i,(n+a)} \leftarrow \mathcal{T}_{i,(n+a)} \oplus \mathcal{T}_{i,a}$.
12. **if** measuring qubit a in the standard basis **then**
13. **if** there exists a stabilizer row $p \in \{n+1, \ldots, 2n\}$ such that $\mathcal{T}_{p,a} = 1$ **then**
14. **for all** rows i such that $i \neq p$ and $\mathcal{T}_{i,a} = 1$ **do**
15. Multiply row i by row p (using the Pauli group operation).
16. Set the $(p-n)$th row equal to row p.
17. Set the pth row to 0 (which corresponds to the identity operator).
18. Set the phase of the pth row, $\mathcal{T}_{p,2n+1} \leftarrow 0$ or $\mathcal{T}_{p,2n+1} \leftarrow 1$ with equal probability.
19. Set $T_{p,n+a} \leftarrow 1$.
20. **return** $T_{p,2n+1}$ as measurement outcome.
21. **else**
22. Set the $2n+1$th row to 0.
23. **for all** rows $1 \leqslant j \leqslant n$ such that $\mathcal{T}_{j,a} = 1$ **do**
24. Multiply row $2n+1$ by row $j+n$.
25. **return** $T_{p,2n+1}$ as measurement outcome.

FIGURE 5.5. The Aaronson–Gottesman simulation algorithm for applying Clifford operations to the tableau representation of a quantum stabilizer state.

Chapter 2). The procedure to perform this conversion is implemented in Aaronson's CHP simulator and has also been used in QMC, but it involves a maximum of 2^n iterations for an n–qubit state vector. In other words the conversion of an n–qubit tableau into the corresponding quantum state vector has a cost which is exponential in n. Unfortunately this conversion becomes necessary when checking several classes of EQPL formula, although there is the notable exception of entanglement partitions, which we will describe later in this chapter.

5.4.1.2. *Optimising the Implementation of the Simulation Algorithm.* There is actually scope for optimising the implementation of the simulation algorithm, notably by using low-level binary operations and by packing the stabilizer array representation into 32-bit registers, as Aaronson and Gottesman chose to do in the final implementation of their CHP simulator (see [1] for details).

The current version of QMC uses binary operations on entire integer registers, where every register corresponds to a bit in the stabilizer array; the Java language does not provide modifiers such as `short` and `unsigned` (which are used in C), which would have helped to make the representation more space–efficient. However, a more efficient implementation of the binary representation of stabilizer generators is likely to be possible using Java's `BitSet` class [142].

5.5. Model Checker Implementation

We now discuss the algorithms that QMC uses for the verification of properties of models. The core requirement is to evaluate quantum formulae γ of EQPL on states of the model, and to use this as a component of the model checking algorithm for QCTL.

5.5.1. EQPL Verification Algorithms and Complexity. The reader is reminded that the semantics of EQPL is defined [98] in terms of a *quantum interpretation structure*, which includes a quantum state $|\psi\rangle$, a classical state ρ and a means of specifying entanglement partitions of $|\psi\rangle$. In our setting ρ includes the global classical state, and we omit the explicit representation of entanglement partitions, instead calculating them on demand.

Note that in QMC we have implemented the full EQPL as described in [98] and in Fig. 4.8, not just the fragment of EQPL ("efficient EQPL") which is contained in QCTL in [12]. The differences between the two are that: (i) full EQPL treats classical formulae (α) as a special case of quantum formulae whereas "efficient EQPL" allows classical formulae only in probability terms ($\int \alpha$), (ii) full EQPL includes complex–valued terms (u), and (iii) full EQPL includes entanglement partition terms ($[F]$). The authors of the logic emphasise that, while full EQPL is designed to embody the first two postulates of quantum mechanics (see Section 2.1), the restricted fragment of the logic contained in QCTL only embodies the first postulate. This is significant as it affects the complexity of model checking QCTL formulae, as there are more cases in which a conversion from the tableau representation to the state vector representation of quantum states is necessary.

Evaluating EQPL formulae over any state $|\psi\rangle$ arising from the simulation of a protocol model requires being able to determine all the valuations in that state, so that the truth value of any

propositional constant (*e.g.*: qb_i where $0 \leqslant i \leqslant N$ for an N-qubit system - this constant corresponds to the state of the ith qubit in the quantum state) can be computed. What this means in more practical terms is that, in order to determine whether a given qubit has valuation *true* (1) or *false* (0) in the current state, it is necessary to extract all the basis vectors which are present in the state vector expansion of $|\psi\rangle$. The process of extracting all the basis vectors requires converting from the space–efficient stabilizer array representation to the state vector corresponding to $|\psi\rangle$, and this conversion can take up to a maximum of 2^N steps if all the 2^N basis vectors appear in $|\psi\rangle$ (as mentioned in Section 5.4.1.1). Even when $|\psi\rangle$ is a stabilizer state, it may contain all of the basis vectors with non-zero coefficients. Therefore in general, evaluating a classical formula requires solving a SAT problem, and of course this is NP-complete. This observation seems rather discouraging given that the process of verifying a state formula requires us to lose the efficient state representation which is used during simulation. A useful direction for future work would be to integrate an off-the-shelf SAT-solver into QMC.

However, there are cases for which we can avoid the conversion from stabilizer array to state vector; for certain classes of formula we can extract the necessary valuation information by processing the stabilizer array directly. We have observed that certain classical EQPL formulae, which do not involve the conjunction operator $\wedge$, may be checkable on a given state $|\psi\rangle$ by just examining the contents of those columns in the stabilizer array corresponding to the qubits in the formula. We are still investigating optimisations and heuristics such as this, bearing in mind that the most general EQPL formulae still require performing a state vector conversion.

It is also worth noting that the algorithm for checking qubit equality does not require extracting the full expansion of the quantum state, but can be checked by operating on the internal stabilizer representation in an efficient way. Further note that checking qubit equality includes an implicit but significant check that the qubits whose states are being compared are disentangled from one another, and that in the implementation the whole quantum states to which these qubits belong are compared directly.

Checking whether a given set of qubits constitutes a partition of a stabilizer state $|\psi\rangle$ is possible using the polynomial time algorithm of Audenaert and Plenio [8] for the so-called "bipartite

Algorithm SAT-EQPL(γ):

1. **if** γ is of the form α **then**
2. **return** VERIFY-CLASSICAL(α)
3. **else if** γ is of the form $t_1 \leqslant t_2$ **then**
4. $r_1 \leftarrow$ EVAL(t_1)
5. $r_2 \leftarrow$ EVAL(t_2)
6. **return** $r_1 \leqslant r_2$
7. **else if** γ is of the form $[F]$ **then**
8. Call CNFP(F) to transform tableau into normal form
9. Search for matching XZ pairs in the two portions of the tableau
10. **if** such pairs are found **then**
11. **return** true
12. **else**
13. **return** false
14. **else if** γ is of the form $\gamma_1 \sqsupset \gamma_2$ **then**
15. $b_1 \leftarrow$ SAT-EQPL(γ_1)
16. $b_2 \leftarrow$ SAT-EQPL(γ_2)
17. **return** $(b_1 \Rightarrow b_2)$ { *where* $\Rightarrow$ *denotes logical implication.* }

Algorithm VERIFY-CLASSICAL(α) :=

1. **Compute all the valuations (basis vectors) in the quantum state, *i.e.* the full state vector $|\psi\rangle$.**
2. **if** α is of the form qB **then**
3. **for all** valuations v in $|\psi\rangle$ **do**
4. $acc \leftarrow acc \wedge v(qB)$
5. **return** acc
6. **else if** α is of the form $\bot$ **then**
7. **return** false
8. **else if** α is of the form $\alpha_1 \Rightarrow \alpha_2$ **then**
9. **for all** valuations v in $|\psi\rangle$ **do**
10. $b_1 \leftarrow$ VERIFY-CLASSICAL(α_1)
11. $b_2 \leftarrow$ VERIFY-CLASSICAL(α_2)
12. $acc \leftarrow acc \wedge (b_1 \implies b_2)$

FIGURE 5.6. EQPL Verification Algorithm: Quantum and Classical Formulae.

Clifford Normal Form (CNFP)," which works directly with the stabilizer array. This algorithm is discussed in Section 5.5.2.

5.5.1.1. *Complexity.* We now formalise the complexity of model checking EQPL formulae. For this calculation we consider only closed formulae (*i.e.* formulae without variables), following [12]. We assume that basic arithmetic operations take $O(1)$ time, and we use $|\phi|$ to denote the length of a formula ϕ, namely, the number of symbols required to write it.

In general, EQPL formulae are evaluated over the explicit state vector representation of a quantum state $|\psi\rangle$ on n qubits, which is a 2^n array of complex numbers. To evaluate a quantum formula γ over a quantum state $|\psi\rangle$, we need to evaluate all the terms in γ. Evaluating a classical formula α (which is a special case of γ) takes $2^n \cdot |\alpha|$ steps as we need to traverse the set of all valuations in $|\psi\rangle$ to determine which ones satisfy α.

Algorithm EVAL(t):

1. **if** t is of the form x **then**
2. **return** $\sigma(x)$
3. **else if** t is of the form r **then**
4. **return** r
5. **else if** t is of the form $(\int \alpha)$ **then**
6. Search for all valuations satisfying α. Call this S.
7. **if** $S = \emptyset$ **then**
8. **return** 0
9. $total \leftarrow 0$
10. **for all** valuations $v \in S$ **do**
11. $c \leftarrow$ coefficient of v
12. $amp \leftarrow$ amplitude of c
13. $total \leftarrow total + amp^2$
14. Return $total$.
15. **else if** t is of the form $t_1 + t_2$ **then**
16. $m_1 \leftarrow$ EVAL(t_1)
17. $m_2 \leftarrow$ EVAL(t_2)
18. **return** $(m_1 + m_2)$
19. **else if** t is of the form $t_1\ t_2$ **then**
20. $m_1 \leftarrow$ EVAL(t_1)
21. $m_2 \leftarrow$ EVAL(t_2)
22. **return** $(m_1 \cdot m_2)$
23. **else if** t is of the form $\mathsf{Re}(u)$ **then**
24. $m \leftarrow$ EVALCOMPLEX(u)
25. **return** $\mathsf{Re}\{m\}$
26. **else if** t is of the form $\mathsf{Im}(u)$ **then**
27. $m \leftarrow$ EVALCOMPLEX(u)
28. **return** $\mathsf{Im}\{m\}$
29. **else if** t is of the form $\mathsf{arg}(u)$ **then**
30. $m \leftarrow$ EVALCOMPLEX(u)
31. **return** $\mathsf{arg}\,\{m\}$
32. **else if** t is of the form $|u|$ **then**
33. $m \leftarrow$ EVALCOMPLEX(u)
34. **return** $||m||$

FIGURE 5.7. EQPL Verification Algorithm: Real Terms.

Algorithm EVAL-COMPLEX(u):

1. **if** u is of the form z **then**
2. **return** $\sigma(z)$
3. **else if** u is of the form $|\alpha\rangle_{FA}$ **then**
4. **if** state is not F-factorisable **then**
5. **return** 0
6. Search for valuations which satisfy α. Call this set S.
7. **if** $S = \emptyset$ **then**
8. **return** 0
9. Search in S for a valuation which verifies (maps to $\top$) all $q \in A$. { *If such a valuation exists, it will be unique.* }
10. **if** no such valuation is found **then**
11. **return** 0
12. **return** the coefficient of the valuation found { *this will be one of* $\pm 1, \pm i, \pm\frac{1}{\sqrt{k}}, \pm i\frac{1}{\sqrt{k}}$ *where* k *is the number of basis vectors arising in the state vector expansion* }
13. **else if** u is of the form $t_1 + it_2$ **then**
14. $m_1 \leftarrow$ EVAL(t_1)
15. $m_2 \leftarrow$ EVAL(t_2)
16. **return** $(m_1 + i \cdot m_2)$
17. **else if** u is of the form $t_1 e^{it_2}$ **then**
18. $m_1 \leftarrow$ EVAL(t_1)
19. $m_2 \leftarrow$ EVAL(t_2)
20. **return** $(m_1 \cdot \exp(i \cdot m_2)$
21. **else if** u is of the form $\overline{u}$ **then**
22. $c \leftarrow$ EVAL-COMPLEX(u)
23. **return** complex conjugate of c
24. **else if** u is of the form $(u_1 + u_2)$ **then**
25. $c_1 \leftarrow$ EVAL-COMPLEX(u_1)
26. $c_2 \leftarrow$ EVAL-COMPLEX(u_2)
27. **return** $(c_1 + c_2)$
28. **else if** u is of the form $(u_1\ u_2)$ **then**
29. $c_1 \leftarrow$ EVAL-COMPLEX(u_1)
30. $c_2 \leftarrow$ EVAL-COMPLEX(u_2)
31. **return** $(c_1 \cdot c_2)$
32. **else if** u is of the form $(\alpha \triangleright u;\ u)$ **then**
33. **if** $b =$ true **then**
34. **return** EVAL-COMPLEX(u_1)
35. **else**
36. **return** EVAL-COMPLEX(u_2)

FIGURE 5.8. EQPL Verification Algorithm: Complex Terms.

However, in our setting there is the additional computational cost of computing the set of all valuations from the tableau representation of the quantum state, which has a worst–case complexity of 2^n (based on the algorithm used in CHP [1]). So evaluating a classical EQPL formula over a stabilizer quantum state represented using a tableau has a worst case complexity of $2^n \cdot 2^n \cdot |\alpha| = 2^{2n} \cdot |\alpha| = O(2^n \cdot |\alpha|)$.

Theorem 5.1 (based on [12])**.** *Assuming that all basic arithmetical operations take $O(1)$ time, there is an algorithm $O(2^n \cdot |\gamma|)$ to decide if an n–qubit stabilizer quantum state $|\psi\rangle$(represented using a tableau) satisfies a quantum formula γ.*

PROOF. Observe that the logical terms which take longer to evaluate are the forms α, $\int \alpha$, $|\top\rangle_{FA}$ (we ignore the form $[F]$ here because it can be evaluated in polynomial time for stabilizer states using the algorithms described in Section 5.5.2). The number of terms in the above forms appearing in a formula γ is bounded above by $|\gamma|$, and each of these requires travelling through the set of all valuations and checking a classical formula. For the forms $\int \alpha$ and $|\top\rangle_{FA}$ we accumulate intermediate results during evaluation (summing amplitudes or squares of amplitudes of those terms satisfying a classical formula), but these are just arithmetic operations. As we have seen the time complexity of evaluating these terms is exponential in the number of qubits n and linear in the length of the formula.

The remaining computations needed for arithmetic operations on real and complex values take at most $O(|\gamma|)$ time, so that the total time needed to decide if a quantum state $|\psi\rangle$satisfies a quantum formula γ is $O(2^n \cdot |\gamma| + |\gamma|) = O(2^n \cdot |\gamma|)$. △

5.5.2. Computation of Bipartite Entanglement Normal Form. The most interesting class of quantum state formulae for which an efficient implementation exists (thus avoiding the expensive state vector conversion) were those of the form $[F]$ where F is a list of qubit symbols, *i.e. entanglement partition formulae*. Such a formula is true in the quantum state $|\psi\rangle$ if all the qubits indexed within the square brackets may be separated from all the other qubits in the state (that is to say, if qubits $q_i, q_j, \ldots \in F$ are disentangled from the rest). We refer to a list of qubits $F = \{q_i, q_j, \ldots\}$ as a *partition* of the state $|\psi\rangle$ if the formula $[q_i, q_j, \ldots]$ is satisfied by $|\psi\rangle$.

We can evaluate such a formula on a stabilizer state by making use of the following theorem due to Audenaert and Plenio [8].

Theorem 5.2 (Bipartite Normal Form)**.** *Consider a system of n qubits, separated into two parties, A and B, containing N_A and N_B qubits, respectively. Consider a stabilizer state described by an array of k independent, commuting generators.*

- *By applying a suitable sequence of elementary row operations and local elementary column (qubit) operations, the stabilizer array can be brought into the following normal form:*

$$\left[\begin{array}{cccc:ccc:cccc:ccc}
X & I & \cdots & I & I & \cdots & I & X & I & \cdots & I & I & \cdots & I \\
Z & I & \cdots & I & I & \cdots & I & Z & I & \cdots & I & I & \cdots & I \\
I & X & \cdots & I & I & \cdots & I & I & X & \cdots & I & I & \cdots & I \\
I & Z & \cdots & I & I & \cdots & I & I & Z & \cdots & I & I & \cdots & I \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
I & I & \cdots & X & I & \cdots & I & I & I & \cdots & X & I & \cdots & I \\
I & I & \cdots & Z & I & \cdots & I & I & I & \cdots & Z & I & \cdots & I \\
\hdashline
I & I & \cdots & I & * & \cdots & * & I & I & \cdots & I & * & \cdots & * \\
\vdots & \vdots & \ddots & \vdots & * & \cdots & * & \vdots & \vdots & \ddots & \vdots & * & \cdots & * \\
I & I & \cdots & I & * & \cdots & * & I & I & \cdots & I & * & \cdots & *
\end{array}\right] \tag{5.3}$$

$$\underbrace{\qquad\qquad\qquad}_{\text{party } A} \qquad \underbrace{\qquad\qquad\qquad}_{\text{party } B}$$

  Here, the asterisk ($*$) stands for either I or X.

- *Every pair of rows containing the XZ combinations corresponds to two qubits (one from each party) being in a pure maximally entangled* EPR *state and completely disentangled from the other qubits. The rows in the lower blocks of the normal form, containing only I and X operators, correspond to the remaining qubits being in a (general, mixed) separable state.*
- *The stabilizer state described by the stabilizer array is locally equivalent to a tensor product of a certain number p of* EPR *pairs Ψ with a separable state. For any additive entanglement measure E, the entanglement of the stabilizer state is $pE(\Psi)$. An upper bound on p is given by $p \leqslant \min(\lfloor k/2 \rfloor, n_A, n_B)$. Equality is obtained if and only $k = 2n_a = 2n_b$.*

If we treat the qubits listed in the partition F of the formula $[F]$ of interest as party A in the bipartite normal form shown, then it follows from part (ii) of the theorem that the existence of

matching XZ pairs in parties A and B will indicate the presence of entanglement between the qubits represented by the corresponding columns.

We omit the presentation of the CNFP algorithm here, as it is thoroughly detailed, along with a proof of correctness of Theorem 5.2, in [8].

5.5.2.1. *Complexity.* The CNFP algorithm involves a sequence of row and column operations on a stabilizer array with k rows and n columns. For our purposes we always use $k = n$ (since n generators are required to represent a unique n–qubit stabilizer state), and the tableau representation has an additional $k = n$ rows representing destabilizer generators. (In the QMC implementation, the CNFP algorithm is modified to update not only the stabilizer generators, but also the destabilizers.)

The computational cost of a row operation is $O(n)$ since all elements of a row are traversed, and similarly for column operations. Since the entire stabilizer array is traversed (including the phase column) during a run of the CNFP algorithm, there is a time complexity of $O(n^2)$.

In the tableau representation described in Fig. 5.4, there are $2n$ rows and $2n + 2$ columns overall; the CNFP algorithm ignores the last (scratch) row. So in total the algorithm goes through all of the $4n^2 + 2n$ elements, and the time taken for this is $O(4n^2 + 2n) = O(n^2)$.

To summarise, it takes polynomial time in n, the number of qubits, to bring the tableau representation of a stabilizer state into a form which allows EQPL formulae of the form $[F]$ to be evaluated. This is in contrast to other formulae, which in general require time exponential in n.

5.5.3. QCTL Verification Algorithms and Complexity. The temporal connectives of QCTL are implemented as an extension to the evaluator for EQPL formulae, and involve the usual labelling algorithms for computation tree logic [24], which are applied to the tree structure generated by QMC. The propositional fragment of the logic is just EQPL, so the QCTL algorithm necessarily invokes Algorithm SAT-EQPL.

The top–level algorithm for checking QCTL formulae is Algorithm SAT-QCTL, shown at the top of Fig. 5.10. The crux of the algorithm is the function MARK($\theta, \mathscr{T}$), which will mark those states of the program tree which satisfy the formula θ. At any one time, this function will need

$$\begin{aligned}
\mathsf{Sat}_{\mathscr{T}}(\gamma) &= \{|\psi\rangle \in S : |\psi\rangle \Vdash \gamma\} && (5.7)\\
\mathsf{Sat}_{\mathscr{T}}(\theta_1 \sqsupset \theta_2) &= (S \backslash \mathsf{Sat}_{\mathscr{T}}(\theta_1)) \cup \mathsf{Sat}_{\mathscr{T}}(\theta_2)) && (5.8)\\
\mathsf{Sat}_{\mathscr{T}}(\mathsf{EX}\theta) &= \{|\psi\rangle \in S : R(|\psi\rangle) \cap \mathsf{Sat}_{\mathscr{T}}(\theta) \neq \emptyset\} && (5.9)\\
\mathsf{Sat}_{\mathscr{T}}(\mathsf{AF}\theta) &= \textbf{FixedPoint}[\lambda X.\{R^{-1}X\} \cap X, \mathsf{Sat}_{\mathscr{T}}(\theta)] && (5.10)\\
\mathsf{Sat}_{\mathscr{T}}(\mathsf{E}[\theta_1 \mathsf{U} \theta_2]) &= \textbf{FixedPoint}[\lambda X.\{R^{-1}X \cap \mathsf{Sat}_{\mathscr{T}}(\theta_1)\}, \mathsf{Sat}_{\mathscr{T}}(\theta_2)] && (5.11)
\end{aligned}$$

FIGURE 5.9. Fixed point model checking algorithm for QCTL.

to initialise and refer to several different labellings, until the final labelling for the formula is produced (the variable *currentlabelling* is a pointer to the current labelling). The final result of evaluating θ is the value of the labelling for the root of the program tree, which is denoted by *currentlabelling*[0] in SAT-QCTL. We assume here that the variable *currentlabelling* has initialised prior to running SAT-QCTL.

The presentation of the QCTL model checking algorithms in Figs. 5.9 and 5.10 is very close to the style in which they have been implemented in QMC. The presentation of model checking algorithms for CTL and its variants (of which QCTL is but one) is more usually given using fixed points. We give such a description of the QCTL model checking algorithm in Fig. 5.9.

Let S denote the set of all states in the program tree $\mathscr{T}$. We denote by $\mathsf{Sat}_{\mathscr{T}}(\theta)$ the set of states in $\mathscr{T}$ which satisfy θ:

$$\mathsf{Sat}_{\mathscr{T}}(\theta) := \{|\psi\rangle \in S : \mathscr{T}, |\psi\rangle \Vdash_{\text{QCTL}} \theta\} \tag{5.4}$$

Let R denote a relation over the states in S such that, for all $q, q' \in S$:

$$R(q, q') \text{ if and only if } q \Rightarrow q' \tag{5.5}$$

In the fixed point version of the algorithm, we use the relation R^{-1} defined thus:

$$R^{-1}X = \{q \in S \mid \exists q' \in X, R(q, q')\} \tag{5.6}$$

5.5.3.1. *Complexity.* The temporal connectives of QCTL are implemented in exactly the same fashion as for the original logic CTL. For a transition system $\mathscr{T}$ of size $S_{\mathscr{T}}$ the complexity of

Algorithm SAT-QCTL(θ, $\mathscr{T}$):
1. MARK(θ, $\mathscr{T}$)
2. **return** *currentlabelling*[0] { *The formula θ holds if the initial state of the execution tree $\mathscr{T}$ has been marked as* true *at the end of the algorithm.* }

Algorithm MARK(θ, $\mathscr{T}$):
1. **if** θ is of the form γ **then**
2. Clear *currentlabelling*
3. **for all** states q in the execution tree $\mathscr{T}$ **do**
4. **if** SAT-EQPL(γ) = true **then**
5. *currentlabelling*[q] $\leftarrow$ true { *this is the case when $q \Vdash \gamma$* }
6. **else**
7. *currentlabelling*[q] $\leftarrow$ false { *this is the case when $q \nVdash \gamma$* }
8. **return** *currentlabelling*
9. **else if** θ is of the form $\theta_1 \sqsupset \theta_2$ **then**
10. *currentlabelling* $\leftarrow$ **new** $labelling_1$
11. MARK(θ_1, $\mathscr{T}$)
12. *currentlabelling* $\leftarrow$ **new** $labelling_2$
13. MARK(θ_2, $\mathscr{T}$)
14. *currentlabelling* $\leftarrow$ **new** $labelling_3$
15. **for all** states q in the execution tree $\mathscr{T}$ **do**
16. $labelling_3[q] \leftarrow (labelling_1[q] \implies labelling_2[q])$
17. **return** $labelling_3$
18. **else if** θ is of the form (EXθ) **then**
19. *currentlabelling* $\leftarrow$ **new** $labelling_1$
20. MARK(θ, $\mathscr{T}$)
21. *currentlabelling* $\leftarrow$ **new** $labelling_2$
22. **for all** transitions (q, q') in $\mathscr{T}$ **do**
23. **if** $labelling_1(q')$ = true **then**
24. $labelling_2(q) \leftarrow$ true
25. **return** $labelling_2$

FIGURE 5.10. The QCTL model checking algorithm (continues in Fig. 5.11).

verifying a CTL formula ϕ has been shown to be $O(|\phi|^2 \cdot S_{\mathscr{T}}^2)$. This extends naturally to QCTL, with the essential distinction that the computational cost of evaluating a propositional (EQPL) formula is now $O(2^n \cdot |\phi|)$.

Theorem 5.3. *Assuming that basic arithmetic operations take $O(1)$ time, the algorithm* SAT-QCTL(θ, $\mathscr{T}$) *for model checking a* QCTL *formula θ over a program tree $\mathscr{T}$ takes $O(|\theta|^2 \cdot S_{\mathscr{T}}^2 \cdot 2^n)$ time.*

PROOF. The propositional CTL model checking algorithm takes $O(|\theta|^2 \cdot S_{\mathscr{T}}^2)$ time. If a quantum formula is treated as a propositional constant, then that would have been the complexity for

Algorithm MARK(θ, $\mathscr{T}$) continued:

```
26. if θ is of the form E[θ1 U θ2] then
27.     currentlabelling ← new labelling1
28.     MARK(θ1)
29.     currentlabelling ← new labelling2
30.     MARK(θ2)
31.     currentlabelling ← new labelling3
32.     for all states q in the execution tree 𝒯 do
33.         currentlabelling(q) ← false
34.         seenbefore(q) ← false
35.         if labelling2(q) = TRUE then
36.             Add q to the set L of states to process
37.     while L is nonempty do
38.         Draw a state q from L
39.         currentlabelling(q) ← true
40.         for all predecessors q′ of q in T do
41.             if seenbefore(q′) = false then
42.                 seenbefore(q′) ← true
43.                 if labelling1(q′) = true then
44.                     Add q′ to the set L of states to process
45.     return labelling3
46. else if θ is of the form (AFθ) then
47.     currentlabelling ← new labelling1
48.     MARK(θ)
49.     currentlabelling ← new labelling2
50.     for all states q in the execution tree 𝒯 do
51.         if labelling1(q) = true then
52.             Add q to the set L of states to process
53.     while L is nonempty do
54.         Draw a state q from L
55.         currentlabelling(q) ← true
56.         if seenbefore(q′) = false then
57.             seenbefore(q′) ← true { where q′ is the predecessor of q }
58.             if labelling1(q′) = true then
59.                 Add q′ to the set L of states to process
60.     return labelling2
```

FIGURE 5.11. The QCTL model checking algorithm (continued from Fig. 5.10).

QCTL. However, a quantum formula must be evaluated for each quantum state in the program tree, and we have found that checking a quantum formula on a quantum state $|\psi\rangle$ in our setting requires time $O(2^n \cdot |\gamma|)$ for a quantum formula γ. The theorem follows directly. △

5.6. Special Extensions

We have implemented two special extensions to the state logic in the implementation of QMC which have proved useful in the development of examples and case studies. Their importance is demonstrated in the case studies and discussed further in Chapter 6.

We use the keyword `history` prior to a variable name to indicate to QMC that we are referring to the history variable of that name, rather than the value of that variable in the current state. This is an extension to the original syntax of the logic. While EQPL was originally intended to enable reasoning about individual quantum states, this extension enables us to reason about the quantum states that arise at different stages during a particular execution (run) of the protocol.

We have introduced an *equality operator* for qubits. We use it to compare the state of a qubit at a particular point during a run of the protocol (after the overall quantum state $|\psi\rangle$ has been explicitly saved) with the state of a different qubit at the end of the k-th run of the protocol (when the overall quantum state is, say, $|\psi'_k\rangle$, for all values of k). The equality of qubits in two different quantum states is well–defined only if the qubits in question are disentangled from the other qubits in the respective states, otherwise these qubits do not have an individual state which can be used in a comparison. Thus a qubit equality formula implicitly incorporates a specification of entanglement properties. Note that this notion of equality is an extension to the EQPL logic which we have implemented in QMC, and its semantics is different from the notion of "qubit equivalence" (denoted by $\Leftrightarrow$ in standard EQPL. Finally, we distinguish the qubit equality operator "==" from the standard equality symbol "=" which we use in QMC for comparing classical variables (integers, booleans, reals).

5.7. Concluding Remarks

In this chapter we have detailed several key aspects of the implementation of the Quantum Model Checker (QMC). The emphasis has been on the algorithms used for model checking, in particular, the algorithms for evaluating EQPL formulae over individual quantum states, and the algorithms for evaluating temporal (QCTL) formulae over the execution tree corresponding to an input program. The complexity of these algorithms has been discussed, and we have noted that

it is possible to escape the need to extract the set of all valuations for a quantum state $|\psi\rangle$ when evaluating an entanglement partition formula by using the so–called Bipartite Normal Form of Audenaert and Plenio.

Some small extensions to EQPL have been described, but their significance will only become apparent in the following chapter, which demonstrates our techniques (and the QMC tool in particular) for analysing specific quantum protocols.

CHAPTER

6

APPLICATIONS

Few things are harder to put up with than a good example.

— Mark Twain

Applications of QMC to various quantum protocols of interest are described here, including quantum teleportation, quantum coin–flipping, a network involving quantum key distribution and quantum error correction, and quantum secret sharing.

6.1. Quantum Teleportation

The first case study is the quantum teleportation protocol [23] discussed in Section 2.7.2. In QMC's specification language, the protocol may be expressed as shown below. This model teleports an arbitrarily-chosen input state representable in the stabilizer formalism, i.e. one of $|0\rangle$, $|1\rangle$,

$\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$, $\frac{1}{\sqrt{2}}(|0\rangle \pm i|1\rangle)$. Because model–checking is exhaustive, the behaviour of the protocol for *all* of these input states is automatically checked.

```
program Teleportation;
var epr_to_A, epr_to_B: channel of qubit;
    A_to_B: channel of integer;
process EPRSource;
var q1,q2: qubit;
begin
    { q1 := newqubit; q2 := newqubit; }
    { had q1; cnot q1 q2; }
    epr_to_A ! q1;
    epr_to_B ! q2;
end;
process Alice;
var q, ancilla, epr1: qubit; a,b: integer;
begin
    q := newqubit;
    if
    □break;
    □X q;
    □had q;
    □{ X q; had q; }
    □{ had q; ph q; }
    □{ X q; had q; ph q; }
    fi
    savequbit q;
    epr_to_A ? epr1;
    { cnot q epr1; had q;
```

```
    a := meas q; b := meas epr1; }
28    A_to_B ! a;
    A_to_B ! b;
30  end;
  process Bob;
32  var epr2: qubit; c,d: integer;
  begin
34    epr_to_B ? epr2;
    A_to_B ? c;
36    A_to_B ? d;
    if
38    □((c=0) and (d=0)) -> break;
    □((c=0) and (d=1)) -> X epr2;
40    □((c=1) and (d=0)) -> Z epr2;
    □((c=1) and (d=1)) -> { Z epr2; X epr2; }
42    fi
  end;
44  endprogram.
```

In our setting, we allow for global variables (such as `epr1`, `epr2`), typed communication channels (such as `epr_to_A`) which are always global, and local (private) variables for each process (such as `a`, `b`, `c`, `d`, `q`). Communication is asynchronous, with executability rules restricting the way in which the interleaving of process is performed. For instance, the process `Bob` cannot start unless channel `epr_to_B` is filled with a value. Note that the curly braces ('{' and '}') are used to group together a sequence of statements that must be executed in one step.

The model shown above describes a system of three communicating processes `EPRSource`, `Alice`, and `Bob`. The process `EPRSource` represents a device which produces pairs of quantum bits in the entangled state $|\Psi\rangle$, and sends the first of these to `Alice` (through a channel `epr_to_A`) and the second to `Bob` (through a channel `epr_to_B`). Note that the process `Alice` may begin

independently of `EPRSource`; its first task is to create a qubit `q` and to place it in one of the six possible states (there are only six since we are restricting the possible operations to those within the stabilizer formalism; however, we can express the fact that *any* one of these possible states can be prepared by Alice, through a non–deterministic choice in lines 16–23. On line 24, the qubit state that has been prepared is saved for later reference; thus we obtain a history variable which will be used for the specification of the property of the protocol.

In line 25, Alice receives a qubit from the `EPRSource` process, and immediately after she performs the operations that transform this qubit so that teleportation can occur. By performing the operations on lines 26–27, `Alice` influences the entangled state (and hence the qubit that Bob receives on line 34). The idea is that measuring one qubit in an entangled state will affect the state of the other qubit. When Bob receives the other qubit in the entangled state (line 34), he can transform it into the original state $|\psi\rangle$ depending on the outcomes of Alice's measurements (the values of variables `c` and `d`).

6.1.1. The Teleportation Property. The requirement for the teleportation protocol described in section 6.1 is that, at the end of the protocol, whatever the measurement outcomes, the third qubit will be in the same state as the first qubit was to begin with.

To express this property, we need to compare the final state of the qubit `Bob.epr2` with the state of qubit `Alice.q`. What is important to note, however, is the state of `Alice.q` evolves during the protocol (so it will not remain in the prepared state $|\psi\rangle$). We actually need to refer to the state of this qubit just before the operations on lines 28—29 of the model are performed.

This justifies the need for the statement on line 24, which "freezes" the state of qubit `Alice.q` and the overall internal state at that particular step in the execution. We can then refer to the state of this variable at that step using the *history variable* `history Alice.q`.

The property we need to express must be stated thus:

```
history Alice.q == Bob.epr2
```
(6.1)

Refer to Section 5.6 for an explanation of history variables and the equality operator for qubits, which we are using here.

6.1.2. Verifying Teleportation. The model presented in section 6.1 and the property in section 6.1.1 may be supplied to QMC for verification directly.

The teleportation model produces a transition system with 369,667 states in a total of 141,108 runs. It takes 110 seconds for QMC to construct this structure from the description of the protocol on a machine with a 3.0GHz Intel Core Duo processor and 2GB of memory. The EQPL formula in (6.1) is shown to be true in all runs of the protocol.

6.2. Quantum Coin Flipping

We have built a QMC model for the quantum coin–flipping protocol due to Bennett and Brassard [20].

We model the quantum coin–flipping protocol in QMC as a system of two processes `Alice` and `Bob`. A qubit is prepared by Alice by making a random choice between one of the four states: $|0\rangle$, $|1\rangle$, $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The preparation of these states involves the application of the H and X operators.

The QMC model for this protocol is listed below.

```
program QuantumCoinFlipping;
var AtoB, BtoA: channel of qubit;
    A2B, B2A: channel of bool;

process Alice;
var x, b, g, result: bool; q: qubit;
begin
    q := newqubit;
    /*Choose one of the four BB84 states and prepare qubit q*/
    if
    □true -> x:=false; b:=false;
```

```
12      □true -> x:=false; b:=true; had q;
        □true -> x:=true; b:=false; X q;
14      □true -> x:=true; b:=true; X q; had q;
        fi
16      AtoB!q;
        B2A?g;
18      A2B!b;
        A2B!x;
20      result := ((not (b and g)) and (b or g));
    end;
22
    process Bob;
24  var g, x_hat, b_hat, x, b, result: bool; rq: qubit;
        abort: bool;      /*Bob aborts the protocol*/
26      dontknow: bool;   /*Bob cannot determine Alice's honesty*/
    begin
28      AtoB?rq;
        if
30      □true -> g:=false; B2A!g;
        □true -> g:=true; B2A!g;
32      fi
        /*Select random measurement basis and measure*/
34      if
        □true -> b_hat:=false;
36      □true -> b_hat:=true; had rq;
        fi
38      x_hat:=meas rq;
        /*Receive original b and x*/
```

```
40      A2B?b;
        A2B?x;
42      /*Compare bases and bits*/
        if
44      □((b=b_hat) and (not (x=x_hat))) -> abort:=true;
        □(not (b=b_hat)) -> dontknow:=true;
46      fi
        result := ((not (b and g)) and (b or g));
48  end;
    endprogram.
```

6.2.1. Verifying the Quantum Coin–Flipping Protocol. Upon successful completion, the quantum coin–flipping protocol must ensure that Alice's and Bob's bit values, x and $\hat{x}$, are equal. Thus a successful run of the protocol will satisfy the formula:

$$\texttt{Alice.result} = \texttt{Bob.result} \tag{6.2}$$

Note that the protocol doesn't always succeed (as there is a possibility that Alice and Bob make different basis choices, $b \neq \hat{b}$; in this case the protocol must be aborted and restarted later). However, if Alice and Bob make compatible basis choices, the protocol should not abort. This is expressed thus:

$$\text{AG}\ (((\texttt{b=b\_hat}) \wedge (\texttt{x=x\_hat})) \implies (\texttt{abort} = \bot)) \tag{6.3}$$

When the coin–flipping model is supplied to QMC, it builds a transition system of 4,742 states. The above properties are satisfied as expected.

Note that we have not considered cheating scenarios or attacks on the protocol as of this writing.

6.2.2. Breaking the Quantum Coin–Flipping Protocol. The properties discussed above are merely correctness properties; it is far more desirable to reason about the *security* of the protocol, and we are developing an extension of QMC that will be able to detect certain types of attack.

This particular protocol can be systematically subverted by using the properties of entanglement. In particular, if Alice uses an entangled state, she can influence Bob's measurement so that the final common bit is the one she chooses.

(1) Suppose that Alice sends a qubit in an entangled state in step 1, *e.g.* of $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$.

(2) After she receives the value of g, she measures her half of the state with basis $b = a \oplus g$. Let x be the outcome.

(3) She knows that Bobs measurement will produce a totally anticorrelated value $\hat{x} = 1 \oplus x$ if he uses basis b.

Thus Alice can influence the outcome of the protocol to have value a by sending Bob the values b and $x \oplus 1$.

6.3. Quantum Key Distribution with Error Correction

In this section we consider a model of a system combining part of the quantum key distribution protocol BB84 [20] with the quantum error correcting code described in [139] (the quantum bit flip code). We will discuss these protocols and proceed to explain the structure of a QMC model for this system. Then we will turn to the verification of this model using the tool.

Quantum key distribution enables two users, Alice and Bob, to establish a common cryptographic key which is secure in the presence of eavesdropping. The two users are assumed to be linked by a quantum channel (such as an optical fibre, through which qubits can be transmitted in the form of polarised light bursts), and they also can communicate over a public, authenticated, classical channel. The attacker has full access to the quantum channel, but can only monitor the classical channel passively. Quantum key distribution has been proven unconditionally secure against all attacks permitted by the laws of quantum mechanics [99] and implementations of this technology already exist, both in academic [31] and commercial settings (see *e.g.* `http://www.idquantique.com`). We consider a simplified version of the BB84 protocol for quantum key distribution here, omitting some classical post-processing steps (esp. secret-key reconciliation and privacy amplification, see [111] for details) that the users must perform after the quantum transmission is complete.

The BB84 protocol was described in some detail in Section 2.7.5. The protocol must be repeated several times in order for a key to be established; a common key bit is generated every time Bob makes a compatible basis choice for his measurement. We model one run of the protocol here.

A simple attack that can be made on this protocol is the following:

- The attacker, Eve, intercepts the qubit that Alice has sent (note that it is impossible to copy the qubit due to the no-cloning theorem of quantum mechanics [111]), chooses a measurement basis of her own, and measures the qubit to obtain a (possibly correct) bit value. She then sends the measured qubit to Bob.

It is not difficult to see that an incorrect choice of measurement basis by Eve would disturb the qubit which Alice originally sent, placing it in a different basis state. It is possible for Alice and Bob to detect this disturbance by exchanging a few of their bits.

We can model Alice's behaviour as the following QMC process (we assume that the qubit `q` is a global variable, and similarly for `ch`):

```
process Alice;
var bit,basis:integer;
begin
    q := newqubit;
    q1:= newqubit;
    q2:= newqubit; /*for redundant encoding of q*/
    q3:= newqubit;
    q4:= newqubit; /*will be used for error recovery*/
    if
    □basis:=0; bit:=0;
    □basis:=0; bit:=1; X q;
    □basis:=1; bit:=0; had q;
    □basis:=1; bit:=1; X q; had q;
    fi
```

```
    cnot q q1; cnot q q2;
    ch!q;
end;
```

Notice how Alice uses the X and H gates to prepare the different qubit states in each case. Applying the H gate essentially transforms a rectilinear basis state into a diagonal basis state.

The processes describing the actions of Eve and Bob are quite similar, since both involve a basis choice and a measurement. Here is the definition of Eve's process in QMC's input language.

```
process Eve;
var ebit,ebasis:integer;
begin
    ch?q;
    if
    □ebasis:=0;
    □ebasis:=1; had q;
    fi
    bbit := meas q;
    ch!q;
end;

process Bob;
var bbit,bbasis:integer;
begin
    ch?q;
    if
    □bbasis:=0;
    □bbasis:=1; had q;
    fi
```

```
    bbit := meas q;
88  end;
```

So far we have assumed that transmissions of qubits are noise-free, *i.e.* the communication channels are perfect. Now we will revise our model to describe the case where a quantum error-correcting code is used to recover from a single bit flip caused by a noisy channel. The code requires each individual qubit prepared by Alice to be redundantly encoded into a 3-qubit system, so that the state $|0\rangle$ is transmitted as $|000\rangle$, and $|1\rangle$ is transmitted as $|111\rangle$. We assume that the quantum channel may induce a bit flip error on any one of the three qubits that are used in this code; for instance, the channel might transform the state

$$\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \quad \text{into} \quad \frac{1}{\sqrt{2}}(|010\rangle + |101\rangle)$$

In this case, the second qubit has been disturbed by the channel. In order to detect such an error, two additional qubits are used; they are known as *ancillas*. By applying a sequence of operations and measurements to the ancillas, the so–called *error syndrome* is obtained, which determines the location of the error. Then, the X operator is applied to the erroneous qubit, thus restoring the initial quantum state of the 3–qubit system (*i.e.* $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ in the above example). The quantum circuit for the bit–flip code is given in Fig. 2.5 (Chapter 2).

In order to account for the error correction in the model of the protocol we have been describing, we need to introduce: (1) two qubits `q1` and `q2` which are used to encode the original state as a three-qubit state, (2) two qubits `q3` and `q4` corresponding to the ancillas which are used to detect the location of the error in the three-qubit state. The QMC process for Alice needs to include the additional commands `cnot q q1; cnot q q2;` just before the transmission `ch!q`, thus encoding the state of qubit `q` across the three qubits.

We model the channel that causes the disturbance as a separate process `Disturb`, defined below:

```
process Disturb;
90  begin
```

```
    ch?q;
92  if
    □X q;
94  □X q1;
    □X q2;
96  fi
    ch!q;
98 end;
```

The final part of the model consists of the process `Correct`, which is responsible for applying the error correction procedure implemented by the circuit in Fig. 2.5.

```
process Correct;
100 var a,b: integer;
begin
102   ch?q;
      cnot q q3; cnot q1 q3;
104   cnot q q4; cnot q2 q4;
      a:=meas q3;
106   b:=meas q4;
      if
108   □((a=1) and (b=1)); -> X q;
      □((a=1) and (b=0)); -> X q1;
110   □((a=0) and (b=1)); -> X q2;
      fi
112   ch!q;
end;
```

6.3.1. Properties for Verification. The model of a quantum key distribution system with an error correcting component described in the previous section is quite realistic, given that we are

taking into account the possibility of a direct attack on the protocol as well as the presence of a noisy quantum channel. A larger system for quantum key distribution might also involve quantum teleportation, so that a qubit is transferred not via a direct quantum channel but through the use of entangled quantum states. We discussed the quantum teleportation protocol in Section 2.7.2 separately.

There are number of combinatorial possibilities which arise during quantum key distribution. Depending on the choices of basis made by Alice, Bob and Eve, it may or may not be possible to detect Eve's presence. For instance, a compatible choice of basis by all three users implies that Eve's measurement of the transmitted qubit does not cause a disturbance to its state. There will be cases when the outcome of Bob's measurement matches the choice of bit originally made by Alice, cases in which Eve's measurement is correct, and so on. QMC explores all possibilities automatically and constructs the correct tree structure for the model described. We can verify different QCTL formulae expressing success or failure of the protocol. For instance, the BB84 protocol normally succeeds to produce a key bit if Alice and Bob use compatible basis choices, *i.e.* when the following state formula is satisfied (`bbasis` and `bbit` are the variables corresponding to Bob's chosen basis and bit value):

$$((\texttt{basis=bbasis}) \implies (\texttt{bit=bbit})) \tag{6.4}$$

This property applies to the entire protocol including the quantum error correction procedure.

6.4. A Quantum Error Correcting Network

It is our opinion that data networks which are implemented using quantum protocols are likely to be a practical setting for secure communications in the future. Such systems are easily expressible in QMCLANG, and these are ideal targets for verification with QMC.

In this section we consider an example of a system consisting of a number of users in a network; the network is modelled by a set of channels between the users, and we assume that these channels are binary symmetric, so that with specified probability p (for our purposes we assume $p = \frac{1}{2}$), the channels in the network will corrupt the bits, and with probability $1 - p$ they will transmit

the bits without error. The users perform quantum error correction to recover the correct bits in this system.

We begin by presenting a QMCLANG process which models the quantum error correction procedure described in Section 2.7.3. This protocol is able to correct a single bit flip error on any one of three qubits. The process `Correct` assumes the existence of a qubit channel `ch` upon which the original qubit `q` is sent. Qubits `q1` and `q2` are used to code the original qubit into a 3–qubit message, while the qubits `q3` and `q4` are ancillas which are used for error detection and recovery. Note that a syndrome of $a = 0, b = 0$ denotes the absence of an error, and in this case no correction is applied (the expression–statement `true` has no effect on the state).

```
process Correct;
var a,b: integer; q:qubit;
begin
    ch?q;
    cnot q q3;
    cnot q1 q3;
    cnot q q4;
    cnot q2 q4;
    a:=meas q3;
    b:=meas q4;
    if
    □((a=1) and (b=1)); -> X q;
    □((a=1) and (b=0)); -> X q1;
    □((a=0) and (b=1)); -> X q2;
    □((a=0) and (b=0)); -> true;
    fi
    ch!q;
end;
```

Next we consider how to model a (quantum) binary symmetric channel (with $p = \frac{1}{2}$, where we use a non-deterministic choice between two options to model equiprobable events).

```
process BSC;
begin
    ch?q;
    if
    □X q; /* disturb qubit q */
    □true; /* don't do anything */
    fi
    ch!q;
end;
```

These small examples demonstrate the key ideas that underlie the larger model discussed next, comprising two users, Alice and Bob, who are each independently linked to a quantum network or "cloud". A source prepares a qubit (in any one of the six possible states permitted in the stabilizer formalism) and passes it to Alice, who encodes it (thus producing a three–qubit state). The network is modelled by a process which makes choices as to whether to disturb each of the three qubits. Bob is the receiver of this data, and he is the one performing the error correction.

Remark 6.1. *This model use a special convention introduced in the latter stages of development of* QMC. *A sequence of statements enclosed in braces ('{' and '}') is treated by* QMC *as a single execution step. In other words, braces enrich the* QMCLANG *formalism with what is known as an atomic construct.*

```
program QECC_Parallel;
var ch_source, ch_alice_net, ch_bob_net : channel of qubit;

process Source;
var q : qubit;
begin
       q := newqubit;
```

```
8       if
        □ Z q;       // |0>
10      □ X q;              // |1>
        □ had q;            // 1/sqrt2 (|0> + |1>)
12      □ {X q; had q;}     // 1/sqrt2 (|0> - |1>)
        □ {had q; ph q;}    // 1/sqrt2 (|0> + i|1>)
14      □ {X q; had q; ph q;} // 1/sqrt2 (|0> - i|1>)
        fi
16      ch_source!q;
end;
18
process Alice;
20 var q, q1, q2, saved: qubit;
begin
22      ch_source?q;                // input qubit 0
        saved := savequbit q;
24      { q1:= newqubit;            // prepare qubit 1
          q2:= newqubit; }          // prepare qubit 2
26      { cnot q q1;
          cnot q q2; }
28      ch_alice_net!q;
        ch_alice_net!q1;
30      ch_alice_net!q2;
end;
32
process Network;
34 var q : qubit;
begin
```

```
36          ch_alice_net?q;
            if
38          □X q;
            □true;
40          fi
            ch_bob_net!q;
42          ch_alice_net?q;
            if
44          □X q;
            □true;
46          fi
            ch_bob_net!q;
48          ch_alice_net?q;
            if
50          □X q;
            □true;
52          fi
            ch_bob_net!q;
54  end;

56  process Bob;
    var q, q1, q2, q3, q4 : qubit;
58      a, b: integer;
    begin
60          ch_bob_net?q;
            ch_bob_net?q1;
62          ch_bob_net?q2;
            q3:= newqubit;       // ancilla 1
```

```
64          q4:= newqubit;       // ancilla 2
            cnot q q3;
66          cnot q1 q3;
            cnot q q4;
68          cnot q2 q4;
            a:=meas q3;
70          b:=meas q4;
            if
72          □((a=1) and (b=1)); X q;
            □((a=1) and (b=0)); X q1;
74          □((a=0) and (b=1)); X q2;
            □((a=0) and (b=0));
76          fi
            {cnot q q1;
78          cnot q q2;}
    end;
80

    endprogram.
```

6.4.1. Property for Verification. The correctness property for the quantum error–correction network is formulated as:

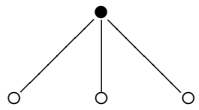
```
history Alice.saved == Bob.q
```
(6.5)

Note the use of a history variable; we are specifying here that the state of the original qubit (which Alice 'saves' using the `savequbit` keyword prior to encoding) must be the one Bob receives at the end of the exchange. QMC shows that the model of the quantum error correcting network does indeed satisfy this property.

6.5. Quantum Secret Sharing

The structure of quantum secret sharing protocols which use graph states was described in Section 2.7.7. Here we present QMC models for the protocol $(k = 3, n = 3)$.

The $(k = 3, n = 3)$ involves the preparation and distribution of the graph state:



where we are denoting by • the node/qubit upon which the secret is encoded. Other nodes are simply shown as ○.

```
program QuantumSecretSharing3_3;
var
       D_to_P1, D_to_P2, D_to_P3 : channel of qubit;
       P1_to_P2, P1_to_P3,
       P2_to_P1, P2_to_P3,
       P3_to_P1, P3_to_P2: channel of integer;

process Dealer;
var q1, q2, q3: qubit; bit: bool;
begin
       // ---- Prepare
       { q1 := newqubit; q2 := newqubit; q3 := newqubit; }
       { had q1; had q2; had q3;
       // CZ 1->2:
        had q2; cnot q1 q2; had q2;
       // CZ 1->3:
        had q3; cnot q1 q3; had q3; }
       // ----
```

```
        // Decide on a secret bit and encode onto vertex 1
        if
        □bit := 0;
        □{ bit := 1; Z q1; }
        fi
        // Send qubits to players
        D_to_P1 ! q1;
        D_to_P2 ! q2;
        D_to_P3 ! q3;
end;

process Player1;
var q: qubit; u, x, y, z, finalkey: integer;
begin
        D_to_P1 ? q;
        had q;
        u := meas q;
        P2_to_P1 ? x;
        P3_to_P1 ? y;
    // XOR operation expressed in terms of AND, OR, NOT
        { finalkey := ((1 - (u*x)) * (u+x));
        finalkey := ((1 - (finalkey*y)) * (finalkey+y));
        finalkey := ((1 - (finalkey*z)) * (finalkey+z));
        }
end;

process Player2;
var q: qubit; x: integer;
```

```
begin
48        D_to_P2 ? q;
          x := meas q;
50        P2_to_P1 ! x;
end;
52

process Player3;
54 var q: qubit; y: integer;
begin
56        D_to_P3 ? q;
          y := meas q;
58        P3_to_P1 ! y;
end;
60

endprogram.
```

The description of quantum secret sharing in QMCLANG is very natural; two special remarks are in order, however. Firstly, since QMC initialises with a computational–basis state, it is necessary for the `Dealer` process to perform some preparation steps to create the initial graph state (lines 13—17).

Secondly, the computation of the XOR of all players' outcomes is performed here in an elaborate way[1] In order to compute $r := u \oplus x \oplus y \oplus z$, we use the identity

$$a \oplus b = \neg(a \wedge b) \wedge (a \vee b) \tag{6.6}$$

and we compute r in three steps thus:

(1) $r := u \oplus x = \neg(u \wedge x) \wedge (u \vee x)$

(2) $r := r \oplus y = \neg(r \wedge y) \wedge (r \vee y)$

[1] This is because there is no built-in XOR operator in QMC for boolean variables, and also because *integer* variables `u`, `x`, `y`, `z`, have been used instead of booleans, so XOR needs to be expressed in terms of arithmetic operations).

(3) $r := r \oplus z = \neg(r \wedge z) \wedge (r \vee z)$

The correspondence between the above and lines 39—42 in the above code should be evident to the reader.

6.5.1. Properties for Verification. To verify the correctness of the protocol described by the model in the previous section, we ask QMC to check the properties:

$$\texttt{Player1.finalkey} \leqslant \texttt{Dealer.bit} \tag{6.7}$$

$$\texttt{Dealer.bit} \leqslant \texttt{Player1.finalkey} \tag{6.8}$$

It is necessary to use two properties to express the simple equality between the final key obtained and the original dealer bit, because the implementation of EQPL only has a less–than–or–equal ($\leqslant$) operator. Checking this formula takes just over 900 sec. on the hardware described in Section 6.1.2.

6.6. Concluding Remarks

In this chapter we have looked at five case studies, specific protocols for which we have built and run QMC models. We have demonstrated the capabilities of the modelling language QMCLANG and the property specification language. The progression of case studies has been in order of increasing complexity, starting with a simple protocol (quantum teleportation) and proceeding to larger systems (quantum key distribution, a quantum error–correcting network and quantum secret sharing). We note here that there is some overlap between the quantum key distribution system and the quantum error-correcting network, since the former incorporates a quantum error correction step as an intrinsic part of the protocol, but the nature of the two models is different; one describes a system combining two sub–protocols, while the other describes a network of different users performing small steps to obtain a common outcome.

CHAPTER

7

REVIEW AND CONCLUSION

Enough research will tend to support your conclusions.

— Arthur Bloch

We have described in this thesis a specification and verification framework for modelling and analysis of quantum information protocols, based on the method of model–checking. We have developed a formal modelling language, QMCLANG, defined its syntax, semantics and type system, and implemented it as part of a model–checking tool, QMC. We have studied, extended, and implemented Exogenous Quantum Propositional Logic (EQPL) and

Quantum Computation Tree Logic (QCTL). The QMC tool implements model–checking algorithms for EQPL/QCTL formulae over the semantic structures corresponding to QMCLANG programs. Here we summarise and discuss the work presented, and outline directions for future work.

7.1. Summary

Below is a detailed summary of the work presented in the thesis.

Chapter 1.: In the first chapter we discussed the emergence and significance of the quantum computation and quantum information discipline. We considered the characteristics of quantum systems which make them usable as information carriers and exposed their peculiarities (*e.g.* nondeterminism). Some key results of this field were surveyed and the motivations for protocol analysis presented. A short overview of formal methods (and particularly model–checking) was given, and relevant previous work by the author and colleagues was identified. A brief survey of related work included discussion of quantum programming languages, semantic techniques for quantum computation, quantum information logics, and quantum simulation.

Chapter 2: described all the theoretical background of relevance to this work, including the nuts and bolts of quantum computing and the stabilizer formalism. A detailed account of several quantum protocols was given, with a view to showing the variety that exists, while exposing common characteristics. Not all of the protocols presented in this section were included in later analyses, yet an understanding of them is considered fundamental to our subject matter.

Chapter 3: detailed an approach to the analysis of quantum protocols based on existing tools and techniques, especially probabilistic model–checking. The development of an experimental tool, PRISMGEN, was described. Our purpose in this chapter was to show why existing tools and techniques are not adequate for the modelling and analysis of quantum protocols, thus justifying the need for a more targeted approach such as the one presented in subsequent chapters.

Chapter 4: focussed on the modelling language QMCLANG, which we developed for use in conjunction with QMC. The full definition of the language includes syntax, operational semantics, an executability predicate (these last two implicitly define an abstract machine for the language), and a static type system. The logics EQPL and QCTL, originally due to Mateus and Sernadas [97], were explained and linked to the semantic structures produced when running QMCLANG programs.

Chapter 5: gave an account of all aspects of the implementation of the Quantum Model Checker (QMC), which implements QMCLANG as a modelling language and a significant fragment of EQPL and QCTL, for the case where quantum states in the stabilizer formalism only are considered. This restriction is significant, as it prevents modelling and analysis of arbitrary quantum computations. The implementation of QMC is based on an efficient (polynomial time) simulation algorithm, but requires time exponential in the size of the system being modelled to be produce verification results. QMC is the key practical contribution of this thesis.

Chapter 6: discussed the application of QMC to a number of real quantum protocols, and demonstrated some of its features in practical use. Where available, some performance figures were provided. The example protocols considered were quantum teleportation, quantum coin–flipping, quantum key distribution, quantum error correction, and quantum secret sharing.

7.2. Discussion

Next we will discuss the basic tradeoff between efficiency and generality that underlies the implementation of the Quantum Model Checker. We will consider the benefits of our approach and address some shortcomings and potential criticisms.

7.2.1. Efficiency *versus* Generality. At the outset of this work our stated objective was to develop a modelling and analysis framework that was sufficiently general for realistic applications, while bringing the benefits of model checking techniques (which had already been demonstrated

in the literature on classical communication and cryptographic protocols) to the novel setting of quantum computation and quantum information.

After initial experimentation with existing verification tools and techniques (as presented in Chapter 3), we became aware of the fact that the already restricted class of protocols that we were able to model could be modelled and simulated in this way as a direct consequence of the Gottesman–Knill Theorem. Since the stabilizer formalism afforded substantial flexibility and accounted well for entanglement (a key feature of most important quantum protocols), we did not see confinement to this formalism as a major limitation for practical applications.

The existence of an efficient simulation algorithm for stabilizer circuits was an important factor in our choice to restrict protocols of interest to those expressible within the stabilizer formalism. This, however, may be seen as a significant limitation to our approach.

While this design choice has led to an efficient simulation component, the verification of protocol properties is not as efficient. Unfortunately, the state formulae of the property specification logic require a computationally expensive conversion from the stabilizer array representation to the full state vector description of a quantum state. The one exception to the rule is the checking of entanglement partition formulae, which are checkable on an individual stabilizer state in polynomial time.

Practical experiments have shown that the efficiency issue is not a major one for current applications, since large input sizes (protocols manipulating many decades or hundreds of bits) are rarely encountered in the literature. We believe, therefore, that we have produced a useful verification tool that may help significantly in the analysis (and possibly design) of practical quantum protocols for various communication tasks.

The restriction to stabilizer states, which accounts for the efficiency of the simulation, is non–negligible. Designers of quantum protocols should ideally be unfettered in their choices of possible input/output states and gates (which need not be limited to those in the Clifford group). We believe that future work should address this limitation by extending QMC to allow for a universal set of quantum gates.

7.2.2. Benefits and Criticisms. The work we have presented includes a modelling language whose semantic definition is extensible, so that a universal set of quantum gates may easily be added. Taken in isolation, QMCLANG is an expressive, practical modelling language with static typing, that may be used in future simulation and analysis tools.

The language and logic used in QMC are both candidates for extensions and improvements, but combining the two as described in this thesis gives rise to a useful formal verification framework, as demonstrated by several case studies.

The QMC implementation is a useful testbed for protocol experimentation and development. Certainly it can be used to analyse other protocols, and models of quantum networks.

It is only too easy to direct criticisms at the complexity of the verification algorithms employed by QMC. Certainly the benefits of using the stabilizer representation are lost during most verifications. We feel that this concern has been addressed through the introduction of the final–states mode in the tool in which, when properties pertaining only to final protocol outcomes need to be checked, only the final states of all protocol runs are stored; the cost of storing and converting to state–vector form each intermediate state arising in a run is prevented in this way, thus improving overall efficiency of verification. Furthermore, the efficient checking of entanglement partition formulae (which would otherwise involve the exponential cost of converting between state representations, as with all other formulae) has been implemented.

Regarding the protocols chosen to test and demonstrate QMC, the following remarks are in order. Although the teleportation example is simple, it is an important building block for practical long-distance quantum communication and cryptographic systems. In contrast to pen-and-paper proofs of correctness of teleportation, our verification explicitly models the protocol as a distributed system, and could be extended to a larger system within the same framework. The other examples are more substantial.

For the protocols considered here, none has been demonstrated to have a flaw; we are aware of this, and we hope that in future work QMC will be used to explore attacks, and to highlight vulnerabilities and subtle flaws on protocols.

Our work on analysing quantum secret sharing protocols is promising in that it may be possible to use QMC to find new (as of yet, undiscovered) protocol variants. Why should particular values of (k, n) correspond to successful secret sharing protocols and not others? Verifying secret sharing protocols for different values of k and n with QMC might yield insights into the underlying structure and/or cause.

It should be added that our techniques may provide the basis for developing automated proofs of unconditional security (or the absence thereof) for some protocols. Mayers' proof of the unconditional security of quantum key distribution, for instance, would benefit from simplification, generalisation to similar protocol variants, and a more uniform presentation (one that does not require recourse to diverse mathematical disciplines and specialised notations). Maybe model checking techniques can provide a useful step in achieving this. The impossibility of unconditionally secure quantum bit commitment may also be demonstrable in an elegant fashion using QMC or QMC–inspired methods.

7.2.3. Future Work. We describe next a number of directions for future work based on QMC and the associated formal verification framework.

7.2.3.1. *Generalisation to Mixed States and a Limited Number of non–Clifford Gates.* According to the original paper of Aaronson and Gottesman [1], it is possible to generalise the tableau representation of stabilizer states, and the associated algorithm, to simulate circuits involving *mixed* states and a limited number of non–Clifford gates. This would be a useful extension to the existing QMC implementation.

7.2.3.2. *Generalised Pure States or Alternative Restrictions.* As we have discussed, it would be useful for QMC to be generalised so that non–stabilizer states and a universal set of quantum gates may be included in protocol models. This requires a substantial reworking of the internals of the tool so that a stabilizer representation can be used where possible, and a state vector representation everywhere else. We note here that, to form a universal set, only the $\frac{\pi}{8}$ gate needs to be added.

Another possibility of interest is the introduction of an alternative *restricted* set of quantum states and operators (as opposed to stabilizer states and Clifford operations). It may be useful to

implement the QUIDD representation of quantum states in QMC. QUIDD enables the representation of a different but still restricted class of quantum circuits (and hence protocols), and this may be interesting for some applications.

7.2.3.3. *Complexity Improvements and SAT Solvers.* Martin Plenio has intimated[1] that it should be possible to avoid the state vector conversion for several simple types of EQPL formulae. It seems that it is possible to extract more information about the state being represented directly from the corresponding tableau. Such efficiency improvements are clearly desirable.

Since the problem of checking an EQPL formula on a quantum state containing all possible basis vectors is an instance of SAT, there are likely to be certain kinds of formula for which the algorithms implemented in currently available SAT solvers offer improvements in efficiency. This is likely to be an important future experiment.

7.2.3.4. *A Process–Algebraic Modelling Language.* At the time the process algebra CQP was conceived, it had been envisaged as the definitive modelling language for quantum protocols. Providing a direct interface from CQP to QMC is a significant task as it would enable users of the tool to benefit from the expressiveness of the process algebra.

A partial translation from CQP to QMC has been developed already by Timothy Davidson (University of Warwick) and is continually in development.

7.2.3.5. *Language and Tool Improvements.* Of interest is the possible extension of QMCLANG to allow for *synchronous communication*; at present, only asynchronous channels are supported. Such an extension would lend more credence to the CQP–to–QMC translation described previously, since it is currently necessary to generate complex code fragments that simulate the synchrony.

Other analysis modes could be added to QMC, *e.g.* a means of computing probabilities of different runs. Links with probabilistic logics and probabilistic model checking tools are likely to emerge here.

7.2.3.6. *Alternative Approaches — Symbolic Model Checking, Bisimulation, Automated Theorem Proving.* Throughout the course of the work presented in this thesis, several alternative methods for quantum protocol analysis suggested themselves. It was proposed *e.g.* that a *symbolic* approach

[1] M.B. Plenio, Private communication, December 2008.

(as opposed to the explicit–state approach taken here) might be a good means of combatting the inevitable state space explosion. Such approaches have been used effectively in the verification of classical hardware, and also for protocols.

As mentioned in Section 1.3.2, a process–algebraic approach to the verification of protocols would involve the definition of suitable equivalence relations, such as *bisimulation* of processes. Developing algorithms to check bisimulations of CQP or QMC processes may well be an interesting technique to consider.

Finally, it has been clear from the beginning that a model checking approach is likely to work well only for manageable, finite protocols (due to the finiteness requirement for the state space, common in model checking). Infinite–state model checking techniques do exist, but they are not as mature as finite methods. It is practically impossible to produce proofs of correctness involving a high degree of generality (as needed *e.g.* for unconditional security proofs for cryptographic protocols) using only model checking, and this restricts the applicability of QMC. Automated theorem proving, on the other hand, does enable a human user to produce powerful and wide–ranging proofs, provided that suitable (meta–) theories have been developed. This alternative approach may be a fruitful avenue of investigation for proving properties of quantum protocols in a formal way.

7.3. Conclusions

As quantum technology becomes more commonplace, and as we have discussed the constant miniaturisation of computing devices makes this a historic inevitability, we are forced to consider ways of designing and reasoning about its different manifestations. The author personally believes that cryptographic applications (and increasing security and privacy concerns) will drive this market, and that relevant analysis tools and techniques are tantamount to its eventual success. It is hoped that the present work has provided a good indication of how such techniques may be implemented.

BIBLIOGRAPHY

[1] S. Aaronson and D. Gottesman. "Improved Simulation of Stabilizer Circuits". In: *Physical Review A* 70.52328 (2004), p. 52328. See pp. 17, 52, 88, 89, 91, 92, 98, 133.

[2] S. Aaronson. *The Complexity Zoo*. URL: `http://qwiki.caltech.edu/wiki/Complexity_Zoo`. See p. 91.

[3] S. Abramsky and B. Coecke. "A Categorical Semantics of Quantum Protocols". In: *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS)*. Also arXiv:quant-ph/0402130. IEEE Computer Society, 2004. See p. 15.

[4] S. Abramsky and B. Coecke. "Abstract Physical Traces". In: *Theory and Applications of Categories* 14.6 (2005), pp. 111–124. See p. 15.

[5] S. Abramsky and B. Coecke. "Physical Traces: Quantum vs. Classical Information Processing". In: *Proceedings of the 9th Conference on Category Theory and Computer Science (CTCS 2002)*. Vol. 69. Electronic Notes in Theoretical Computer Science. Also arXiv:cs.CG/0207057. Elsevier Science, 2003. See p. 15.

[6] R. Alur and T. A. Henzinger. "Reactive modules". In: (1996), pp. 207–218. See p. 42.

[7] S. Anders and H. Briegel. "Fast Simulation of Stabilizer Circuits Using A Graph State Representation". In: *Physical Review A* 73.22334 (2006), p. 022334. See pp. 17, 39.

[8] K. Audenaert and M. Plenio. "Entanglement on mixed stabiliser states: Normal Forms and Reduction Procedures". In: *New Journal of Physics* 7.170 (2005), p. 170. See pp. 94, 99, 100.

[9] F. Babich and L. Deotto. "Formal Methods for Specification and Analysis of Communication Protocols". In: *IEEE Communications Surveys & Tutorials* 4.1 (Dec. 2002). See p. 9.

[10] D. Bacon. *Stabilizer Quantum Error Correcting Codes*. Lecture Notes for course CSE 599d (Quantum Computing), University of Washington. 2006. See pp. 28, 30.

[11] J. Baggott. *Beyond Measure: Modern Physics, Philosophy and the Meaning of Quantum Theory*. Oxford University Press, 2004. See p. 3.

[12] P. Baltazar, R. Chadha, and P. Mateus. "Quantum computation tree logic – model checking and complete calculus". In: *International Journal of Quantum Information* 6.2 (2008), pp. 219–236. See pp. 16, 76, 80, 93, 95, 98.

[13] M. L. Bellac. *A Short Introduction to Quantum Information and Quantum Computation*. Cambridge University Press, 2006. See p. 3.

[14] M. Ben-Ari, Z. Manna, and A. Pnueli. "The temporal logic of branching time". In: *Acta Informatica* 20 (1983), pp. 207–226. See p. 13.

[15] G. Benenti, G. Casati, and G. Strini. *Principles of Quantum Computation and Information: Basic Concepts*. Vol. 1. World Scientific, 2004. See p. 3.

[16] G. Benenti, G. Casati, and G. Strini. *Principles of Quantum Computation and Information: Basic Tools and Special Topics*. Vol. 2. World Scientific, 2007. See p. 3.

[17] P. Benioff. "Quantum mechanical Hamiltonian models of Turing machines that dissipate no energy". In: *Physical Review Letters* 48 (1982), pp. 1581–1585. See p. 6.

[18] C. Bennett. "Quantum Cryptography Using Any Two Nonorthogonal States". In: *Physical Review Letters* 68.21 (1992), pp. 3121–3124. See p. 37.

[19] C. H. Bennett. "Logical reversibility of computation". In: *IBM Journal of Research and Development* 17.6 (1973), p. 525. See p. 6.

[20] C. H. Bennett and G. Brassard. "Quantum Cryptography: Public Key Distribution and Coin Tossing". In: *Proceedings of International Conference on Computers, Systems and Signal Processing*. 1984. See pp. 2, 14, 30, 34, 37, 44, 110, 113.

[21] C. H. Bennett, G. Brassard, and J.-M. Robert. "Privacy amplification by public discussion". In: *SIAM J. Comput.* 17.2 (1988), pp. 210–229. ISSN: 0097-5397. See p. 37.

[22] C. H. Bennett and S. J. Wiesner. "Communication Via One- and Two-Particle Operators on Einstein–Podolsky–Rosen States". In: *Physical Review Letters* 69.20 (1992), pp. 2881–2884. See pp. 30, 46.

[23] C. H. Bennett et al. "Teleporting an Unknown Quantum State Via Dual Classical and Einstein–Podolsky–Rosen Channels". In: *Physical Review Letters* 70 (1993), pp. 1895–1899. See pp. 30, 31, 48, 106.

[24] B. Bérard et al. *Systems and Software Verification: Model–Checking Techniques and Tools*. Springer–Verlag, 1999. See p. 100.

[25] E. Bernstein and U. Vazirani. "Quantum Complexity Theory". In: *Proceedings of the 25th ACM Symposium on Theory of Computation*. 1993, pp. 11–20. See p. 7.

[26] G. Birkhoff and J. von Neumann. "The Logic of Quantum Mechanics". In: *The Annals of Mathematics* 37.4 (Oct. 1936), pp. 823–843. See p. 15.

[27] D. Bjorner and C. B. Jones, eds. *The Vienna Development Method: The Meta-Language*. London, UK: Springer-Verlag, 1978. ISBN: 3-540-08766-4. See p. 11.

[28] P. E. Black, D. R. Kuhn, and C. J. Williams. "Quantum Computing and Communications". In: *Advances in Computers* 56 (2002), pp. 189–244. See p. 1.

[29] D. Bohm. *Quantum Theory*. Prentice Hall, 1951. See p. 5.

[30] T. Bolognesi and E. Brinksma. "Introduction to the ISO Specification Language LOTOS". In: *Computer Networks and ISDN Systems* 14.1 (Jan. 1987), pp. 25–59. See p. 12.

[31] D. Bouwmeester, A. Ekert, and A. Zeilinger, eds. *The Physics of Quantum Information*. Springer–Verlag, 2000. See pp. 3, 8, 113.

[32] J. P. Bowen. *Formal Methods Wiki*. URL: http://formalmethods.wikia.com/. See p. 10.

[33] G. Brassard. *Modern Cryptology: A Tutorial*. Vol. 325. Lecture Notes in Computer Science. Springer–Verlag, 1988. See p. 33.

[34] G. Brassard and C. Crépeau. "Quantum Bit Commitment and Coin Tossing Protocols". In: *Advances in Cryptology — CRYPTO '90*. Ed. by A. Menezes and S. Vanstone. Volume 537 of *Lecture Notes in Computer Science*. Springer–Verlag, 1991, pp. 49–61. See p. 37.

[35] G. Brassard and L. Salvail. "Secret-key reconciliation by public discussion". In: *Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT'93)*. Lofthus, Norway: Springer–Verlag, 1994, pp. 410–423. ISBN: 3-540-57600-2. See p. 36.

[36] L. Cardelli. "Type Systems". In: *CRC Handbook of Computer Science and Engineering*. 2nd ed. CRC Press, 2004. Chap. 97. See p. 71.

[37] F. Ciesinski and M. Größer. "On Probabilistic Computation Tree Logic." In: *Validation of Stochastic Systems*. 2004, pp. 147–188. See p. 43.

[38] E. M. Clarke and E. A. Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic". In: *Logic of Programs Workshop*. Vol. 131. Lecture Notes in Computer Science. Springer, 1981. See p. 10.

[39] R. Cleaveland, J. Parrow, and B. Steffen. *The Concurrency Workbench: A semantics-based tool for the verification of finite-state systems*. 1993. URL: http://citeseer.comp.nus.edu.sg/215937.html. See p. 12.

[40] C. Cohen-Tannoudji, B. Diu, and F. Laloë. *Quantum Mechanics*. Vol. 1. Wiley–Interscience, 1977. See pp. 4, 5.

[41] V. Danos and E. D'Hondt. "Quantum knowledge for cryptographic reasoning". In: *Proceedings of 3rd International Workshop on Development of Computational Models (DCM'07)*. 2007. See p. 16.

[42] V. Danos, E. D'Hondt, E. Kashefi, and P. Panangaden. "Distributed Measurement-based Quantum Computation". In: *Electron. Notes Theor. Comput. Sci.* 170 (2007), pp. 73–94.

ISSN: 1571-0661. DOI: http://dx.doi.org/10.1016/j.entcs.2006.12.012. See p. 15.

[43] V. Danos, E. Kashefi, and P. Panangaden. "The measurement calculus". In: *J. ACM* 54.2 (2007), p. 8. ISSN: 0004-5411. DOI: http://doi.acm.org/10.1145/1219092.1219096. See p. 15.

[44] D. Deutsch. "Quantum theory, the Church-Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London A* 400 (1985), pp. 96–117. See p. 6.

[45] D. Deutsch and R. Jozsa. "Rapid solution of problems by quantum computation". In: *Proc. Royal Soc. London A* 439 (1992), pp. 553–558. See p. 6.

[46] E. D'Hondt and P. Panangaden. "Reasoning About Quantum Knowledge". In: *FSTTCS*. Ed. by R. Ramanujam and S. Sen. Vol. 3821. Lecture Notes in Computer Science. Springer, 2005, pp. 553–564. ISBN: 3-540-30495-9. See p. 16.

[47] E. W. Dijkstra. *A Discipline of Programming*. Prentice–Hall, 1976. See p. 60.

[48] A. Ekert. "Quantum Cryptography Based on Bell's Theorem". In: *Physical Review Letters* 67.6 (1991), pp. 661–663. See p. 37.

[49] C. Elliot. "Quantum Cryptography". In: *IEEE Security & Privacy Magazine* 2.4 (July 2004), pp. 57–61. See p. 8.

[50] E. A. Emerson. "Temporal and modal logic". In: *Handbook of Theoretical Computer Science*. Vol. B: Formal Models and Semantics. MIT Press, 1990, pp. 995–1072. See p. 80.

[51] R. Fagin, J. Y. Halpern, and N. Megiddo. "A logic for reasoning about probabilities". In: *Information and Computation* 87 (1990), pp. 78–128. See p. 16.

[52] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. Cambridge: MIT Press, 1995. See p. 13.

[53] Y. Feng, R. Duan, Z. Ji, and M. Ying. "Probabilistic bisimulations for quantum processes". In: *Inf. Comput.* 205.11 (2007), pp. 1608–1639. ISSN: 0890-5401. DOI: http://dx.doi.org/10.1016/j.ic.2007.08.001. See p. 15.

[54] R. Feynman. "Simulating Physics with Computers". In: *International Journal of Theoretical Physics* 21.6&7 (1982), pp. 467–488. See p. 6.

[55] R. P. Feynman. "There's Plenty of Room at the Bottom". In: *Feynman and Computation: Exploring the Limits of Computers*. Ed. by A. Hey. Perseus Books, 1999. Chap. 7, pp. 63–76. See p. 4.

[56] Formal Systems Europe Ltd. *Failures Divergences Refinement (FDR)*. `http://www.fsel.com`. See p. 10.

[57] E. Fredkin and T. Toffoli. "Conservative logic". In: *International Journal of Theoretical Physics* 21.3—4 (1982), pp. 219–253. See p. 6.

[58] E. Gagnon. "SableCC, An Object–Oriented Compiler Framework". MA thesis. School of Computer Science, McGill University, Montréal, 1998. See p. 86.

[59] E. R. Gansner and S. C. North. "An open graph visualization system and its applications to software engineering". In: *Software Practice and Experience* 30.11 (2000), pp. 1203–1233. ISSN: 0038-0644. See p. 86.

[60] S. J. Gay. "Quantum Programming Languages: Survey and Bibliography". In: *Mathematical Structures in Computer Science* 16.4 (2006), pp. 581–600. See pp. 15, 57.

[61] S. J. Gay and R. Nagarajan. "Communicating Quantum Processes". In: *POPL '05: Proceedings of the 32nd* ACM *Symposium on Principles of Programming Languages, Long Beach, California*. 2005. See pp. 14, 83.

[62] S. J. Gay, R. Nagarajan, and N. Papanikolaou. "Probabilistic Model–Checking of Quantum Protocols". In: *Proceedings of DCM 2006: Proceedings of the 2nd International Workshop on Developments in Computational Models*. arXiv:quant-ph/0504007. 2006. See pp. 14, 42, 46.

[63] S. J. Gay, R. Nagarajan, and N. Papanikolaou. "QMC: A Model Checker for Quantum Systems". In: *Proceedings of 20th International Conference on Automated Verification (CAV 2008) (Princeton, NJ, USA, July 7–14, 2008)*. Vol. 5123. Lecture Notes in Computer Science. 2008, pp. 543–547. See p. 15.

[64] I. Glendinning. *Links on Simulation, Modelling, and Error Prevention for Quantum Computers*. `http://www.vcpc.univie.ac.at/~ian/hotlist/qc/simulation.shtml`. 2006. See p. 17.

[65] G. Gonthier. "A computer-checked proof of the Four Colour Theorem". Microsoft Research, Cambridge. 2005. See p. 11.

[66] M. Gordon. *HOL: A Machine Oriented Formulation of Higher Order Logic*. Tech. rep. 68. Computer Laboratory, University of Cambridge, 1985. See pp. 11, 13.

[67] M. J. C. Gordon. "HOL: A Proof Generating System for Higher-Order Logic". In: *Current Trends in Hardware Verification and Automated Theorem Proving*. Ed. by G. Birtwistle and P. A. Subrahmanyam. Springer-Verlag, 1989, pp. 73–128. See pp. 11, 13.

[68] D. Gottesman. "An Introduction to Quantum Error Correction". In: *Quantum Computation: A Grand Mathematical Challenge for the Twenty-First Century and the Millennium*. Ed. by S. J. Lomonaco, Jr. American Mathematical Society, 2002. See pp. 28, 30, 52.

[69] D. Gottesman. "The Heisenberg Representation of Quantum Computers". In: *Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics*. Ed. by S. Corney, R. Delbourgo, and P. Jarvis. International Press, 1999. See pp. 28, 30, 42, 52.

[70] L. K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proc. 28th Annual* ACM *Symposium on the Theory of Computing (STOC)*. 1996, pp. 212–219. See pp. 2, 7.

[71] J. Gruska. *Quantum Computing*. McGraw–Hill International, 1999. See pp. 2, 3, 7, 25, 30, 31, 53.

[72] Y. Hardy and W.-H. Steeb. *Classical and Quantum Computing with C++ and Java Simulations*. Birkhäuser Verlag, 2001. See p. 3.

[73] M. Hennessy and R. Milner. "On Observing Nondeterminism and Concurrency". In: *Proceedings of the 7th Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, 1980, pp. 299–309. ISBN: 3-540-10003-2. See p. 13.

[74] T. Hey and P. Walters. *The New Quantum Universe*. Cambridge University Press, 2003. See p. 3.

[75] M. Hirvensalo. *Quantum Computing*. Springer Verlag, 2001. See p. 3.

[76] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice–Hall, 1985. See pp. 11, 12, 58.

[77] G. Holzmann. *The Design and Validation of Computer Protocols*. Prentice–Hall, 1991. See pp. 8–10.

[78] G. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Pearson Education, 2003. See pp. 10, 11, 42.

[79] J. Humphreys and M. Prest. *Numbers, groups and codes*. Cambridge University Press, 1989. See p. 26.

[80] M. R. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. 1st ed. Cambridge University Press, 2000. See p. 10.

[81] B. Jonsson, W. Yi, and K. G. Larsen. "Probabilistic Extensions of Process Algebras". In: *Handbook of Process Algebra*. Elsevier, 2001. ISBN: 444828303. See p. 12.

[82] P. Jorrand and M. Lalire. "From Quantum Physics to Programming Languages: A Process Algebraic Approach". In: *Unconventional Programming Paradigms: Revised Selected and Invited Papers from the International Workshop UPP 2004*. Vol. 3566. Lecture Notes in Computer Science. Springer, 2005. See p. 15.

[83] P. Jorrand and M. Lalire. "Toward a Quantum Process Algebra". In: *Proceedings of the 1st ACM Conference on Computing Frontiers*. Also arXiv:quant-ph/0312067. ACM Press, 2004. See p. 15.

[84] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2007. See p. 3.

[85] R. W. Keyes. "Miniaturization of electronics and its limits". In: *IBM J. Res. Dev.* 32.1 (1988), pp. 24–28. ISSN: 0018-8646. See p. 2.

[86] M. Kwiatkowska, G. Norman, and D. Parker. "Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach". In: *International Journal on Software Tools for Technology Transfer (STTT)* 6.2 (2004), pp. 128–142. See p. 10.

[87] T. P. Laboratory". *YAHODA Verification Tools Database*. Faculty of Informatics, Masaryk University. URL: `http://anna.fi.muni.cz/yahoda/`. See p. 10.

[88] M. Lalire. "Relations among quantum processes: bisimilarity and congruence". In: *Mathematical Structures in Comp. Sci.* 16.3 (2006), pp. 407–428. ISSN: 0960-1295. DOI: http://dx.doi.org/10.1017/S096012950600524X. See p. 15.

[89] R. Landauer. "Information is Inevitably Physical". In: *Feynman and Computation: Exploring the Limits of Computers*. Ed. by A. Hey. Perseus Books, 1999. See p. 3.

[90] K. G. Larsen, P. Pettersson, and W. Yi. "UPPAAL in a nutshell". In: *Journal of Software Tools for Technology Transfer* 1.1—2 (1997), pp. 134–152. See p. 10.

[91] S. Lloyd. *Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos*. Jonathan Cape, 2006. See pp. 2, 3.

[92] S. Loepp and W. K. Wootters. *Protecting Information: From Classical Error Correction to Quantum Cryptography*. Cambridge University Press, 2006. See p. 3.

[93] G. Lowe. "Breaking and Fixing the Needham–Schroeder Public-Key Protocol using FDR". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 1055. Lecture Notes in Computer Science. Springer, 1996, pp. 147–166. See p. 10.

[94] D. C. Marinescu and G. M. Marinescu. *Approaching Quantum Computing*. Pearson Prentice Hall, 2005. See p. 3.

[95] D. Markham and B. C. Sanders. "Graph States for Quantum Secret Sharing". In: *Physical Review A* 78 (2008), p. 042309. See pp. 38, 39.

[96] P. Mateus and A. Sernadas. "Exogenous Quantum Logic". In: *Proceedings of CombLog'04 - Workshop on Combination of Logics: Theory and Applications*. Ed. by W. A. Carnielli, F. M. Dionísio, and P. Mateus. Lisbon: IST Press, 2004, pp. 141–150. See pp. 16, 76, 81.

[97] P. Mateus and A. Sernadas. "Reasoning about quantum systems". In: *Proceedings of Ninth European Conference on Logics in Artificial Intelligence*. Springer–Verlag, 2004, pp. 239–251. See pp. 16, 76, 78, 130.

[98] P. Mateus and A. Sernadas. "Weakly complete axiomatization of exogenous quantum propositional logic". In: *Information and Computation* 204.5 (2006), pp. 771–794. See pp. xiii, 16, 76, 77, 93.

[99] D. Mayers. "Unconditional Security in Quantum Cryptography". In: *Journal of the ACM* 48.3 (2001), pp. 351–406. See pp. 37, 113.

[100] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. See p. 10.

[101] R. McWeeny. *Quantum Mechanics: Principles and Formalism*. Dover Publications, 2003. See p. 5.

[102] R. Van der Meyden and M. Patra. "A Logic for Probability in Quantum Systems". In: *Computer Science Logic, 17th International Workshop, CSL 2003, 12th Annual Conference of the EACSL, and 8th Kurt Gödel Colloquium, KGC 2003, Vienna, Austria, August 25-30, 2003, Proceedings*. Ed. by M. Baaz and J. A. Makowsky. Vol. 2803. Lecture Notes in Computer Science. Springer, 2003, pp. 427–440. See p. 16.

[103] R. Van der Meyden and M. Patra. "Knowledge in Quantum Systems (Extended Abstract)". In: *Proceedings of Conference On Theoretical Aspects of Rationality and Knowledge*. University of Indiana, Indiana: ACM Press, 2003, pp. 104–117. See p. 16.

[104] J.-J. C. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Vol. 41. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1995. See p. 13.

[105] R. Milner. *Communicating and Mobile Systems: The π–Calculus*. Cambridge University Press, 1999. See pp. 11, 12.

[106] R. Milner. *Communication and Concurrency*. Prentice–Hall, 1989. See pp. 11, 12.

[107] G. Moore. "Cramming more components onto integrated circuits". In: *Electronics* 38.8 (1965), pp. 82–85. See p. 2.

[108] R. Nagarajan and S. J. Gay. "Formal Verification of Quantum Protocols". arXiv:quant-ph/0203086. 2002. See p. 14.

[109] V. Natarajan and G. Holzmann. "Outline for an Operational Semantics of PROMELA". In: *The SPIN Verification System*. Ed. by J.-C. Grégoire, G. J. Holzmann, and D. A. Peled. Vol. 32. DIMACS Series in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS, 1997. See p. 82.

[110] R. Needham and M. Schroeder. "Using Encryption for Authentication in Large Networks of Computers". In: *Communications of the ACM* 21.12 (Dec. 1978), pp. 993–999. See p. 10.

[111] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. See pp. 3, 7, 26–28, 30, 31, 53, 89, 113, 114.

[112] N. J. Nilsson. "Probabilistic logic". In: *Artificial Intelligence* 28.1 (1986), pp. 71–87. See p. 16.

[113] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Vol. 2283. LNCS. Springer, 2002. See p. 11.

[114] B. Ömer. "A Procedural Formalism for Quantum Computing". MA thesis. Department of Theoretical Physics, University of Vienna, 1998. See p. 15.

[115] B. Ömer. "Quantum Programming in QCL". MA thesis. Institute of Information Systems, Technical University of Vienna, 2000. See p. 15.

[116] I. S. Organisation. *LOTOS - A formal description technique based on the temporal ordering of observational behaviour*. ISO Standard 8807:1989. 1989. See p. 12.

[117] S. Owre, N. Shankar, and J. Rushby. "PVS: A Prototype Verification System". In: *CADE 11*. Saratoga Springs, NY 1992. See p. 11.

[118] N. Papanikolaou. "Reasoning Formally about Quantum Systems: An Overview". In: *ACM SIGACT News* 36.3 (Sept. 2005), pp. 51–66. See p. 16.

[119] N. Papanikolaou. "Techniques for Design and Validation of Quantum Protocols". Also available as Research Report CS-RT-413. MA thesis. Department of Computer Science, University of Warwick, 2005. See pp. 8, 14.

[120] D. Park. "Concurrency and automata on infinite sequences". In: *5th GI–Conference on Theoretical Computer Science*. Springer, 1981, pp. 167–183. See p. 12.

[121] D. Parker, G. Norman, and M. Kwiatkowska. PRISM *2.0 Users' Guide*. Feb. 2004. See pp. 10, 43.

[122] D. Parker. "Implementation of Symbolic Model Checking for Probabilistic Systems". PhD thesis. University of Birmingham, 2002. See pp. 10, 42, 43.

[123] M. Patra. "A Logic for Quantum Circuits and Protocols". Submitted for publication. 2005. See p. 16.

[124] S. Perdrix. "Quantum entanglement analysis based on abstract interpretation". In: *SAS'08: The 15th International Static Analysis Symposium*. 2008. See p. 15.

[125] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac. "Matrix Product State Representations". In: *Quantum Information and Computation* 7 (2007), p. 401. URL: `http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0608197`. See p. 53.

[126] E. Pietriga. "A Toolkit for Addressing HCI Issues in Visual Language Environments". In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* 0 (2005), pp. 145–152. See p. 86.

[127] A. Pnueli. "A Temporal Logic of Concurrent Programs". In: *Theoretical Computer Science* 13 (1981), pp. 45–60. See p. 13.

[128] A. Ponse, S. A. Smolka, and J. A. Bergstra. *Handbook of Process Algebra*. Elsevier Science Inc., 2001. ISBN: 444828303. See p. 11.

[129] F. Prost and C. Zerrari. *A logical analysis of entanglement and separability in quantum higher-order functions*. Quantum Physics Archive: arXiv:0801.0649v1 [cs.LO]. 2008. See p. 15.

[130] J.-P. Quielle and J. Sifakis. "Specification and Verification of Concurrent Systems in CESAR". In: *Proceedings of the Fifth International Symposium on Programming*. Vol. 137. Lecture Notes in Computer Science. Turin, Italy: Springer, 1982, pp. 337–351. See p. 10.

[131] S. Richards and M. Sadrzadeh. "Aximo: Automated Axiomatic Reasoning for Information Update". In: *Electron. Notes Theor. Comput. Sci.* 231 (2009), pp. 211–225. ISSN: 1571-0661. DOI: `http://dx.doi.org/10.1016/j.entcs.2009.02.037`. See pp. 13, 17.

[132] R. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. See p. 7.

[133] A. Roscoe. "Model-Checking CSP". In: *A Classical Mind, Essays in Honour of C.A.R. Hoare*. Prentice-Hall, 1994. See p. 10.

[134] M. Sadrzadeh. "Actions and Resources in Epistemic Logic". PhD thesis. Université du Québec, Montréal, Canada, 2006. See p. 17.

[135] J. Sanders and P. Zuliani. "Quantum Programming". In: *Mathematics of Program Construction*. Volume 1837 of *Lecture Notes in Computer Science*. Springer, 2000, pp. 80–99. See p. 15.

[136] P. Selinger. "Towards a Quantum Programming Language". In: *Mathematical Structures in Computer Science* 14.4 (2004), pp. 527–586. See p. 15.

[137] P. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings of 35th Annual Symposium on Foundations of Computer Science*. IEEE Press, 1994. See pp. 2, 7.

[138] J. Spivey. *Understanding Z: A specification language and its formal semantics*. Cambridge University Press, 1998. See p. 11.

[139] A. M. Steane. "Quantum Computing and Error Correction". In: *Proceedings of the NATO Advanced Research Workshop*. Ed. by A. Gonis and P. Turchi. Mykonos: IOS Press, 2000, pp. 284–298. See pp. 32, 50, 113.

[140] W.-H. Steeb and Y. Hardy. *Problems and Solutions in Quantum Computing and Quantum Information*. World Scientific, 2004. See p. 3.

[141] S. Stenholm and K.-A. Suominen. *Quantum Approach to Informatics*. Wiley Interscience, 2005. See p. 3.

[142] Sun Microsystems, Inc. *Java Platform Standard Edition 6 SDK*. `http://java.sun.com/javase/6/docs/`. 2006. See p. 93.

[143] The Coq development team. *The Coq proof assistant reference manual*. Version 8.1. LogiCal Project. 2006. URL: `http://coq.inria.fr`. See p. 11.

[144] T. Toffoli. *Reversible computing*. Technical Memo MIT/LCS/TM-151. MIT Laboratory for Computer Science, 1980. See p. 6.

[145] L. G. Valiant. "Quantum Circuits That Can Be Simulated Classically in Polynomial Time". In: *SIAM J. Comput.* 31.4 (2002), pp. 1229–1254. See p. 53.

[146] M. Van den Nest, J. Dehaene, and B. De Moor. "Graphical description of the action of local Clifford transformations on graph states". In: *Physical Review A* 69 (2004), p. 022316. See p. 39.

[147] A. Van Tonder. "A Lambda Calculus for Quantum Computation". In: *SIAM Journal on Computing* 33.5 (2004). Also arXiv:quant-ph/0307150, pp. 1109–1135. See p. 15.

[148] S. Wiesner. "Conjugate Coding". In: *ACM SIGACT News* 15 (1983 (original manuscript 1969)), pp. 78–88. See pp. 4, 33.

[149] C. P. Williams and S. H. Clearwater. *Ultimate Zero and One: Computing at the Quantum Frontier*. Springer, 2000. See p. 4.

[150] N. Wirth. "The Programming Language Pascal". In: *Acta Informatica* 1 (1971), pp. 35–63. See p. 57.

[151] W. K. Wootters and W. H. Zurek. "A Single Quantum Cannot Be Cloned". In: *Nature* 299 (1982), pp. 802–803. See p. 5.

[152] M. Ying, Y. Feng, R. Duan, and Z. Ji. "An algebra of quantum processes". In: *ACM Trans. Comput. Logic* 10.3 (2009), pp. 1–36. ISSN: 1529-3785. DOI: http://doi.acm.org/10.1145/1507244.1507249. See p. 15.

Index

π–calculus, 12

QMC
  interpreter, 88
  scheduler, 87

Aaronson–Gottesman simulation algorithm, tableau, 90
Abramsky, Samson, 15
automated theorem proving, 9, 11

BB84, 3, 34

check matrix, 89
Coecke, Bob, 15
CSP, 11

exogenous logic, 16

FDR, 10, 11

Gonthier, Georges, 11
Grover, Lov, 2
Gruska, Jozef, 2

Hennessy–Milner logic, 13

Landauer, Rolf, 3
Lloyd, Seth, 2

Mayers, Dominic, 37
measurement, 5
model checking, 9, 10
Moore, Gordon, 2

Needham–Schroeder PKCS protocol, 10
no–cloning theorem, 5

Perdrix, Simon, 15
privacy amplification, 37
process algebra
  bisimulation, 12
  CCS, 12
  CSP, 12
  equivalences, 12
  LOTOS, 12
  probabilistic, 12
PROMELA, 11
protocol, 8

QMC (Quantum Model–Checker)
  graphical user interface, 85
quantum
  computing, 6

entanglement, 5
logic, 16
parallelism, 5
superposition, 5

SableCC, 86
secret key reconciliation, 36
Shor, Peter, 2
SPIN, 10, 11
superposition, 5

Wootters, 5