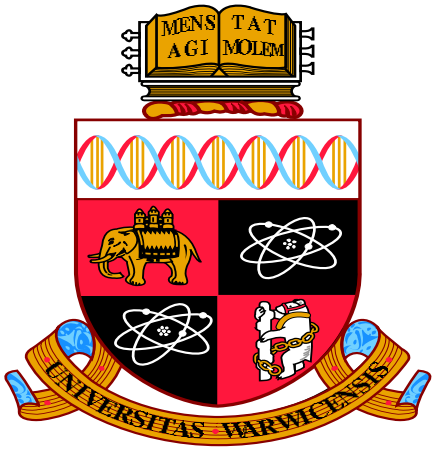
AUTOMATED EQUIVALENCE CHECKING OF QUANTUM INFORMATION SYSTEMS

Ebrahim Ardeshir-Larijani



A thesis submitted for the degree of

Doctor of Philosophy

Supervisors:

Prof. Rajagopal Nagarajan

Dr. Ranko Lazic

Department of Computer Science

The University of Warwick

February 2014

DECLARATION

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work presented was carried out by the author and appeared in the following publications:

1. Equivalence Checking of Quantum Protocols (With S. J. Gay and R. Nagarajan). In: 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Volume 7795 Lecture Notes in Computer Science, pages 466-480, Springer 2013 [9].

2. Verification of Concurrent Quantum Protocols by Equivalence Checking (With S. J. Gay and R. Nagarajan). In: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Volume 8413 Lecture Notes in Computer Science, Springer 2014 (in press) [11]

3. Automated Verification of Quantum Protocols (With S. J. Gay and R. Nagarajan). Poster presented in 17th Conference on Quantum Information Processing (QIP), 2014 [10].

Ebrahim Ardeshir-Larijani,

February 2014

Glasgow, Scotland

ABSTRACT

Quantum technologies have progressed beyond the laboratory setting and are beginning to make an impact on industrial development. The construction of practical, general purpose quantum computers has been challenging, to say the least. But quantum cryptographic and communication devices have been available in the commercial marketplace for a few years. Quantum networks have been built in various cities around the world, and plans are afoot to launch a dedicated satellite for quantum communication. Such new technologies demand rigorous analysis and verification before they can be trusted in safety and security-critical applications.

In this thesis we investigate the theory and practice of equivalence checking of quantum information systems. We present a tool, Quantum Equivalence Checker (QEC), which uses a concurrent language for describing quantum systems, and performs verification by checking equivalence between specification and implementation. For our process algebraic language CCS^q, we define an operational semantics and a superoperator semantics. While in general, simulation of quantum systems using current computing technology is infeasible, we restrict ourselves to the *stabilizer* formalism, in which there are efficient simulation algorithms and representation of quantum states.

By using the stabilizer representation of quantum states we introduce various algorithms for testing equality of stabilizer states.

In this thesis, we consider concurrent quantum protocols that behave *functionally* in the sense of computing a deterministic input-output relation for all interleavings of a concurrent system. Crucially, these input-output relations can be abstracted by superoperators, enabling us to take advantage of linearity. This allows us to analyse the behaviour of protocols with arbitrary input, by simulating their operation on a finite basis set consisting of stabilizer states. We present algorithms for the checking of functionality and equivalence of quantum protocols. Despite the limitations of the stabilizer formalism and also the range of protocols that can be analysed using equivalence checking, QEC is applied to specify and verify a variety of interesting and practical quantum protocols from quantum communication and quantum cryptography to quantum error correction and quantum fault tolerant computation, where for each protocol different sequential and concurrent model are defined in CCS^q.

We also explain the implementation details of the QEC tool and report on the experimental results produced by using it on the verification of a number of case studies.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [70]

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisors Prof. Rajagopal Nagarajan and Dr. Ranko Lazic for their support, encouragement and invaluable guidance throughout my studies. Raja introduced me to Quantum Information Processing and Quantum Computing and challenging problems in this new exciting field. Ranko, patiently and kindly helped me to overcome research hurdles during my PhD.

I am indebted to Dr. Simon Gay for sharing his insights and ideas regarding the main problems addressed in this thesis. As a scientific mentor, he showed me how to understand a research problem and how to move towards its solution. His insistence on clarity in developing ideas will remain an important lesson in my future career. He also hosted me at School of Computing of the University of Glasgow during last the six months, and provided a conducive atmosphere to write my thesis. It would be impossible to accomplish my research goals without him.

My PhD was generously funded by Centre for Discrete Mathematics and its Applications (DIMAP) and the Department of Computer Science of the University of Warwick. I also have received travel funds from EPSRC Network on Structures at the Interface of Physics and Computer Science, and ETH Zurich for participation in Qcrypt 2011.

I have enjoyed working along with academic staff in Warwick and Glasgow, in particular people who helped me in assessing and progressing my research: Dr. Marcin Jurdziński, Prof. Artur Czumaj and Dr. Jane Sinclair.

I have benefited from useful discussions with Dr. Nick Papanikolaou, Dr. Tim Davidson and Ittoop Puthoor on my research topic.

I am grateful to my examiners Dr. Ross Duncan and Dr. Jane Sinclair for giving invaluable comments that significantly improved my thesis.

My brother Dr. Hadi A Larijani warmly hosted me, in numerous occasions, in Glasgow. His unconditional support as well as sharing his academic experience, had a great impact on me and my research. I had very enjoyable times with him and his family. I cannot thank them enough.

My passion for pursuing science mainly comes from my father Prof. Mohammad Javad A Larijani. He always encourages me to be tireless in learning and exploring new areas. His influence on me and my life is beyond words. My mother Farah A Larijani, has supported me in difficult times and her prayers are always with me. I even do

not know how to thank her. This thesis is sincerely dedicated to my parents.

CONTENTS

LIST OF FIGURES

1

INTRODUCTION

Quantum Information Processing (QIP) is an emerging field at the boundary of quantum physics and computer science. The success of *quantum mechanics* in understanding the physical world on one hand, and expanding the domain of computer science on the other hand, has left *QIP* in the spotlight of many researchers across engineering, physics, mathematics and computer science.

The idea of building a computational device based on quantum mechanical laws dates back to 1980s [47] and [77]. In particular, Deutsch introduced the *Quantum Turing Machine* in [35].

The Classical Turing Machine [98] operates on digital bits, where they have binary values of 0 or 1. In contrast, the Quantum Turing Machine can operate on *quantum bits* (*qubits*), where their values are determined by *quantum states*, which can be not only 0 or 1, but also can be in a combination (superposition) of 0 and 1.

Quantum Computers are devices that work with qubits and they operate according to the laws of quantum mechanics, e. g. they are abstracted by a quantum Turing machine. It has been shown that quantum computers can outperform digital computers in *some* computational tasks: *Deutsch-Jozsa* [36], Bernstein-Vazirani [18] and Simon's al-

gorithm [96] are among early examples of *Quantum Algorithms* which are faster than their classical analogues. It was in 1995 that Shor introduced a polynomial time algorithm for *integer factorization* and *discrete logarithms* using a quantum computation model [95]. Shor's discovery revived research in QIP, because of the importance of these two problems in cryptography, and the consequences of solving them for current communication security technologies.

The prospect of *QIP* is not limited to a novel computational model, it also offers new ways of developing fast and secure communication systems. Quantum communication technologies use unique characteristics of quantum mechanics such as *Quantum Entanglement* and *No-Cloning* theorem to enhance the security of communicating systems. Bennett and Brassard introduced the first quantum cryptographic protocol in 1984 (BB84) for distributing classical secret keys (QKD) using qubits [15]. The importance of quantum cryptography is due to the fact that it is *unconditionally* secure [79] (unless quantum mechanics is fundamentally wrong). Later Bennett et al. [17] presented the *Quantum Teleportation* protocol for transferring quantum states using only classical communication, local quantum operations and quantum entanglement.

With a growing interest in security of communications, many industrial and governmental agencies are looking into opportunities that quantum technologies can present in the future [86]. Nevertheless, to realise quantum technologies, physicists and engineers have to be able to control quantum phenomena. This is a difficult task due

to the existence of *noise* at the quantum level, which affects quantum computational processes and creates *quantum decoherence*. *Quantum Information Theory* tries to understand quantum decoherence and transmission of information by qubits within a mathematical formalism. Remarkably, it has been discovered that with the help of *Quantum Error Correction* codes and keeping the level of noises within a certain *threshold*, quantum decoherence can be overcome [27].

Today, theoretical developments in QIP are followed by our advances in *implementing* QIP systems in the laboratory and industry. This is emphasised by a recent physics Nobel prize to Wineland and Haroche for their achievements in experimenting with fundamental aspects of QIP, such as entanglement, in laboratory [1]. Companies like ID Quantique and MagiQ Technologies are selling quantum cryptography products, based on QKD, to the private and public sectors. There is even a Canadian company called D-Wave, selling a product based on *Adiabatic Quantum Computation* [43] for solving specific optimization problems, and major companies and organisations such as Google and NASA purchased their systems despite divided opinions on whether they have *truly* quantum devices. Quantum communication networks based on QKD have been built around the world, such as DARPA Quantum Network in Boston [42], the SeCoQC network around Vienna [103] and the Tokyo QKD Network [93]. Recently, as a part of EU project, qubits have been teleported over 140 km of open space, between two Canary Islands [76]. Moreover, there are plans to

launch satellites capable of QKD based communications in the near future [104].

Naturally, one should distinguish between information-theoretic proofs of the security of quantum cryptography and the security of *implementation* of such systems. The growing complexity of QIP systems demands alternative and novel ways of analysis. In particular, in this thesis, we are interested in investigating the *formal verification* of QIP systems.

1.1 FORMAL VERIFICATION

Formal verification combines logical and deductive reasoning with algorithmic techniques to model and understand software, hardware and distributed systems. The scope of formal verification techniques in analysing complicated systems has grown dramatically in the past few decades. As a result, today there exists a wide spectrum of verification techniques from *Model Checking* and *Theorem Proving*, to *Process Calculus*, all of which have helped us grasp a better analytical understanding of sophisticated computer systems.

In model checking [28], [13], a system S is described by its *state space* (i. e. configuration of systems at any given time) and a model M_S, described by a *Labelled Transition System (LTS)* and consisting of states and transitions between them. Every state in M_S is *labelled* with atomic propositions that represent basic properties satisfied at the

given state. A *property* of S, denoted by ϕ_S is a logical formula e. g. in *Temporal Logic* which describes the intended specification. Model checking involves that $M_S \vDash \phi_S$, that is to show that ϕ_S is satisfied on all *paths* from *initial* states to *reachable* states of M_S. There are variations of model checking with respect to different kind of models (such as probabilistic) and properties (such as CTL, LTL, CTL*, etc.).

The main limitation of model checking is the *explosion of states*, as the number of states grows exponentially in the size of the input, especially when there are many processes and communications. *Symbolic Model Checking* [25] and *Bounded Model Checking* [29] have been developed to tackle the state explosion problem. In Symbolic Model Checking, a concise representation of a model is constructed by using *Binary Decision Diagrams (BDD)* [24], and the checking of properties on BDD can be done rapidly in many important cases [25]. Bounded Model Checking however, explores models incrementally. Properties on paths of length k are checked first, and if necessary, model checking then proceeds on paths of length $k+1$. Bounded Model Checking in some cases avoids explosion of states [29].

Since its introduction three decades ago, model checking has progressed from an ambitious theoretical technique to successful industrial practice. Many model checkers are used for the verification of various software, hardware and safety-critical systems. Some example of model checkers in use are Symbolic Model Checker SMV [25], PRISM [72] (for probabilistic model checking), SPIN [67], BLAST [20] (for checking C programs), JAVA Path Finder (JPF) [64] (for verification of

Java byte codes) and UPPAAL [74], (which verifies real-time systems using timed automata) among others.

An alternative way of reasoning about computer systems is to *formally prove* certain properties as theorems, in a well defined deduction system. The idea is to formulate theories in a deduction system by defining necessary rules and axioms and then implement systematic procedures for derivation of the theorems (constructing proofs) by applying rules to the axioms. The introduction of *Higher Order Logic (HOL)* to analyse programming languages in one hand and development of *Automated Theorem Provers* such as HOL [57],Isabelle [85] and later Coq [89] and Agda [21] on the other hand, has contributed significantly to the area of formal verification. Although *Automated Theorem Proving* provides more generality in terms of verification compared to Model Checking, it needs user intervention to guide proofs for specific problems, which is a drawback.

Process calculi are frameworks for analysing *observable behaviours* of distributed systems. This idea was studied by Milner [81] in the *Calculus of Communicating Systems (CCS)*, where the *algebraic* structure of concurrent systems and equational reasoning for processes was developed. At around same time, Hoare introduced *Communicating Sequential Processes (CSP)* [66] as a programming language for describing concurrent systems. Later, inspired by CCS, a semantics of CSP was defined in [23]. For concurrent systems where the configuration is changing, Milner introduced a more flexible calculus, called π*-calculus* [92]. In this calculus, processes in a network can also

communicate names of channels, thus possibly changing the configuration of the network. There have been numerous applications in developing π-calculus style formalisms, such as calculus for cryptographic protocols (Spi-Calculus) [5] and stochastic process calculus for analysing biological networks [88].

Combining process calculus and model checking resulted in *process oriented* model checking. In particular, showing *behavioural equivalence* (or bisimulation) of processes using model checkers is the most important application of process oriented model checking. Model checkers CWB-NC (*Concurrency Work Bench of the New Century*) for CCS [30] and FDR (Failures Divergences Refinement) for CSP [91] are among successful examples of applying process oriented model checking to industrial problems.

1.2 FORMAL VERIFICATION IN QIP

This thesis makes a contribution to the application of formal verification techniques to QIP, particularly by using *equivalence checking*. Over the last decade there have been efforts to bridge computer science areas such as programming languages and formal verification with QIP. The *Quantum Pseudo Code* of Knill [69], for describing quantum algorithms at a higher level than quantum circuits, is an early example of using formal languages in QIP. Meanwhile, different quantum programming languages for future quantum computers have been de-

signed (see the survey by Gay [50]), among which is Selinger's *Quantum Programming Language (QPL)* [94]. In another direction, Abramsky and Coecke [6] have defined *categorical semantics* for quantum protocols, which lead to the formulation of *categorical quantum mechanics*, giving more insights into the foundation of quantum mechanics using theoretical computer science methods.

Process calculus has been extended to the quantum setting. *Communicating Quantum Processes (CQP)* by Gay and Nagarajan [53] is an instance in which π-calculus is modified with QIP constructs for the analysis of quantum communication systems. Subsequently, behavioural equivalence (bisimulation) for CQP processes has been investigated in [33]. Another example of quantum process calculus is *qCCS*, introduced by Ying et al. [102]. In addition to an operational semantics, a denotational semantics of *qCCS* is defined using *super-operators*. These are *linear transformations* acting on *subspaces* of the *Hilbert Space* (the vector space that qubits live in) and of great significance in quantum mechanics. Superoperators will be defined in Chapter 2 and used throughout this thesis. Equivalence relations for qCCS and different variations of bisimulation (i.e open and symbolic bisimulation) are studied in [44], [34] and [46].

The aforementioned formal approaches to QIP are based on a particular model for quantum computation, which we will explain in Chapter 2 and use throughout the thesis. However, there are other quantum computational models such as *measurement based* [61], [83] and [90], and *linear optical* quantum computation [40]. *Measurement*

Calculus [32] and *Linear Optical Quantum Processes* [48] study formal aspects of these alternative models of computation.

Despite theoretical developments in formal analysis of quantum systems, still there is a demand for tool support. For this purpose, *Quantum Model Checker (QMC)* [56] has been implemented. This tool verifies quantum protocols using *property oriented* model checking. Properties in QMC are described in *Quantum Computation Tree Logic (QCTL)* [14], a logic for specifying quantum states. In general analysing quantum systems with classical computers is *infeasible*. QMC can only verify a restricted class of quantum protocols, which lie under *Stabilizer Formalism* (see Chapter 5 for definitions). This is a class of quantum computation which represents *Stabilizer States* in *polynomial space* and simulates certain operations on them in *polynomial time*. Furthermore, simulation tools for the stabilizer formalism have been developed in [4] and [7]. This formalism is a useful for many applications in QIP, and will be introduced in Chapter 5. Nevertheless, the stabilizer formalism does not cover all possible quantum states and operations, therefore model checking of quantum protocols using stabilizer formalism only *provides* a witness rather than a *proof* for the correctness of quantum protocols. Among other examples of tool supports for QIP is *Quantomatic* which is a tool for reasoning about categorical semantics of quantum protocols as mentioned earlier [38]. The input to this tool are graphs representing the underlying categorical structure of quantum systems and rules are translated as graphical

operations on the input. However, this tool is not fully automatic and user intervention can be complicated.

At the circuit level, the aim of tool development is to optimise *synthesis* of quantum circuits for different implementations of quantum algorithms. Despite scalability issue, there have been many implementations of such tools (see for example [22], [100]). Among very recent quantum circuit simulators with a programming interface are *Quipper* [62] and *Liquid* [80], developed at Dalhousie University and Microsoft, respectively.

1.3 OUTLINE AND CONTRIBUTION

In Chapter 2 we give necessary definitions from quantum mechanics and linear algebra. Then in Chapter 3 we review related work, some of which were mentioned in the introduction, in more detail.

In Chapter 4, we review the language QPL [94] and discuss its *superoperator* semantics. Then we introduce a concurrent language CCS^q for describing concurrent QIP protocols. This language is based on *CCS* and allows the mixing of classical and quantum data.

In terms of *concurrency*, we consider the *synchronised interleaving* model, and provide a justification for this choice. Furthermore, we present reduction rules for CCS^q. We define its semantics by reducing each interleaving to a sequential QPL program and interpreting it as a superoperator.

In Chapter 5, we explain the stabilizer formalism and the main simulation algorithm along with many *normal form* algorithms. We give details of the *Stabilizer Basis (SB)* [51] for the space of *density matrices*, i. e. the linear space which superoperators act upon. We also present three algorithms for *equality testing* of stabilizer states, based on algebraic and information theoretic aspects of stabilizer states. Subsequently, we show how to interpret CCS^q protocols with *arbitrary* inputs using the *linearity* of superoperator semantics. We will define *functional* quantum protocols, i. e. quantum protocols that behave *deterministically* with respect to the input/output relation and show that we are able to verify them by our equivalence checking technique. Furthermore, we we present our algorithm for the verification of *functional concurrent protocols* which uses stabilizer states equality testing, stabilizer basis and the linearity of superoperators. A discussion on computational complexity of this algorithm follows. We also consider how *error correction* protocols with *general errors* can be verified using the *linearity* argument. The extension of linearity arguments for verification of stabilizer quantum circuits, using *map state duality*, is also discussed.

In Chapter 6 we give implementation details of an equivalence checking tool *Quantum Equivalence Checker (QEC)*, followed the definition of models for quantum protocol used in case studies. The case studies, include the verification of a range of protocols from quantum communication, quantum fault tolerant protocols to quantum

cryptography. Finally, experimental results of equivalence checking will be discussed.

2

BACKGROUND

In this chapter we review fundamental concepts of quantum mechanics and QIP which are necessary for presentation of this thesis. We start from linear algebra and vector spaces, then we explain basic principles of quantum mechanics followed by main elements of quantum computation and quantum information theory. For more elaborate details the reader may consult [84]. It should be noted that although our approach to study QIP is based on *formal* rather than *quantitative* analysis, recalling main results of quantum information theory makes the context of this thesis more clear.

2.1 LINEAR ALGEBRA

Linear algebra studies vector spaces and linear transformations. An important vector space in mathematics and physics is *Hilbert Space*, where quantum mechanics is formulated. In the following, first we give basic definitions and notations, then Hilbert Space will be formally defined.

Vectors are represented by Dirac's *ket* notation [37] : $|v\rangle$. Let *complex conjugate* be denoted by $|v\rangle^*$ and $|v\rangle^\dagger$ stands for conjugate transpose, then *bra* notation represents *dual* vector :

$$\langle v| = |v\rangle^\dagger$$

Inner product for two vectors $|v\rangle$ and $|w\rangle$ in the vector space V is defined as a function:

Inner product

$$\langle v|w\rangle : V \times V \to \mathbb{C}$$

which satisfies the following properties:

1. $\langle v| \sum_i \alpha_i |w_i\rangle = \sum_i \alpha_i \langle v|w_i\rangle$
2. $\langle v|w\rangle = \langle w|v\rangle^*$
3. $\langle v|v\rangle \geq 0$

Outer product

Similarly the *outer product* is defined as a function which maps two vectors to another vector:

$$|w\rangle\langle v| : V \times V \to V$$

such that

$$|w\rangle\langle v|(\alpha) = |w\rangle\langle v|\alpha\rangle$$

Trace Function

Let A be a matrix representation of an operator ρ. Trace of ρ, denoted by $tr(\rho)$, is a matrix function which returns the sum of diagonal elements of A:

$$tr(\rho) = \sum_i A_{ii}$$

A *Hilbert Space* $\mathcal{H}$ is a complex vector space, equipped with inner product, zero 0 and unit 1 elements, satisfying following conditions for all $|v\rangle, |w\rangle, |u\rangle \in \mathcal{H}$: *Hilbert Space*

$$|v\rangle + |w\rangle = |w\rangle + |v\rangle$$

$$(|v\rangle + |u\rangle) + |w\rangle = |v\rangle + (|u\rangle + |w\rangle)$$

$$0 + |v\rangle = |v\rangle$$

$$\alpha(\beta|v\rangle) = (\alpha\beta)|v\rangle$$

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

$$\alpha(|v\rangle + |w\rangle) = \alpha|v\rangle + \alpha|w\rangle$$

$$1|v\rangle = |v\rangle$$

$$\langle v|w\rangle = \langle w|v\rangle^*$$

$$\langle v|v\rangle \geq 0$$

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

A useful operator on Hilbert Space is called *Tensor Product*. This linear operator describes how Hilbert Spaces can be composed to construct a larger Hilbert Space. *Tensor Product*

Formally, for Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$ with dimensions n_1 and n_2 respectively, the *Tensor Product* maps these two spaces to a $n_1 \cdot n_2$ Hilbert space $\mathcal{H}_1 \otimes \mathcal{H}_2$, which is constructed in the following way:

Let $\langle|\rangle_1$, $\langle|\rangle_2$ and $\langle|\rangle$ be the inner products defining $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_1 \otimes \mathcal{H}_2$, respectively. Suppose $v_1, w_1 \in \mathcal{H}_1$ and $v_2, w_2 \in \mathcal{H}_2$, we have:

$$\langle v_1 \otimes v_2 | w_1 \otimes w_2 \rangle = \langle v_1 | w_1 \rangle_1 \langle v_2 | w_2 \rangle_2$$

The space $\mathcal{H}_1 \otimes \mathcal{H}_2$ is then spanned by linear combinations of tensor products, that is also satisfies the following properties:

1. For a complex number c and states $|v\rangle \in \mathcal{H}_1$ and $|w\rangle \in \mathcal{H}_2$:

$$c(|v\rangle \otimes |w\rangle) = (c|v\rangle) \otimes |w\rangle = |v\rangle \otimes (c|w\rangle)$$

2. For states $|v_1\rangle, |v_2\rangle \in \mathcal{H}_1$ and $w \in \mathcal{H}_2$:

$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$$

3. For states $v \in \mathcal{H}_1$ and $|w_1\rangle, |w_2\rangle \in \mathcal{H}_2$:

$$|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$$

2.2 QUANTUM MECHANICS

Quantum Mechanics is the mathematical theory which formalises quantum systems (e.g. sub-atomic particles) and their behaviours. Each quantum system is abstracted by Hilbert Spaces and their components are described by *quantum states*, as vectors of Hilbert Spaces. In this way, many phenomena of physics at quantum level such as *non-locality*, *contextuality* and *no-cloning* can be explained mathematically. Some cases such as non-locality and contextuality, are *impossible* to be described merely by a classical probabilistic model (*Hidden Vari-*

able model [41]). The impossibility results of this kind are referred as *no-go theorems*, and play an important role in foundations of physics.

Nevertheless, results in quantum mechanics are based on a few principles, known as *postulates* of quantum mechanics. In the following we introduce these principles, then we generalise *pure quantum states* to *density matrices* and finally, *superoperators* will be introduced.

2.2.1 *Postulates of Quantum Mechanics*

As a mathematical theory, quantum mechanics provides a model for understanding quantum phenomena. The *definitive* rules in which quantum systems *must* follow are not given in quantum mechanics, instead a *conceptual framework* where such rule can be inferred, is provided. In the following we list this framework, known as *postulates* of quantum mechanics:

1. **Postulate 1**: Any physical system (which is assumed to be isolated) is abstracted by a *Hilbert Space* which is called *state space*. The system is completely specified by unit vectors in it state space, known as *state vectors*.

2. **Postulate 2**: Evolution of a *closed* quantum system is described by a *unitary* operator U, that is a linear transformation operating on the *state vector* and has the property $UU^\dagger = I$. It should be noted that timed evolution of quantum states is described by a differential equation, called *Schrödinger equation* [84, p 82]

however, this formulation is not necessary for this thesis arguments.

3. **Postulate 3**: Quantum measurement is described by a set of operators $\{M_m\}$ where m corresponds to the measurement outcomes. For a state $|\phi\rangle$, the probability of measuring with outcome m is:

$$p(m) = \langle\phi|M_m^\dagger M_m|\phi\rangle \tag{2.1}$$

and after performing measurement, $|\phi\rangle$ permanently changes to

$$\frac{M_m|\phi\rangle}{\sqrt{\langle\phi|M_m^\dagger M_m|\phi\rangle}} \tag{2.2}$$

Measurements operators are subject to the *completeness equation*, which ensures the probabilities adds up to 1:

$$\sum_m M_m^\dagger M_m = I \tag{2.3}$$

2.2.2 *Density Matrices and Superoperators*

An alternative way of describing a quantum state to vector state is by using *density operator*. The benefit of using density operators is two folds: Firstly, often in QIP we need to deal with uncertainties about quantum states. Density operators can describe *uncertain* state vectors

(*mixed state*) in a compact and convenient way, although these two representations of quantum states are mathematically equivalent. In the following we give two formulations of density operators, one uses *probabilities* and another is in terms of *trace preserving* matrices. Then we present two significant applications of density operators namely *reduced density operator* and *Superoperators*.

Definition 2.1 *Let $\{(|\phi_i\rangle, p_i)\}$ be an ensemble of a quantum state , where p_is are probabilities (i. e. $\sum_i p_i = 1$). The density operator corresponding to the above ensemble is defined as:*

Density Operator (defined by ensemble)

$$\rho := \sum_i p_i |\phi_i\rangle\langle\phi_i|$$

where $|\phi_i\rangle\langle\phi_i|$ denote outer product as in Section 2.1.

Another characterisation of density operators uses structural algebraic properties:

Theorem 2.1 *The operator ρ is a density operator associated with the ensemble $\{(|\phi_i\rangle, p_i)\}$ if and only if it satisfies following conditions:*

Density Operator (Defined by positive matrices)

1. (Trace condition): $tr(\rho) = 1$.
2. (Positivity): $\langle\phi|\rho|\phi\rangle \geq 0$.
3. (Hermitian) : $\rho^\dagger = \rho$.

As a corollary of Theorem 2.1, a criterion for deciding if a quantum state is mixed or pure, are obtained.

Corollary 2.1 *For every density operator ρ, we have:*

1. $tr(\rho^2) \leq 1$.

2. $tr(\rho^2) = 1$ *if and only if* ρ *is pure state otherwise,* ρ *is in mixed state.*

An important application of density operators is *reduced density operators* for describing composite quantum systems. Using reduced operator, one can obtain *partial trace* of a larger quantum state. Formally, suppose ρ_{AB} is a density operator corresponding to two different quantum systems A and B. In order to obtain a density operator corresponding to the system A, we need to trace out system B from the joint state. This can be done by using partial trace function, assuming tr_B applies trace function to B:

$$\rho_A = tr_B(\rho_{AB}) \tag{2.4}$$

For example, suppose we have given the following entangled quantum state, known as *Bell pair* in density operator form:

$$\rho_{AB} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\frac{1}{\sqrt{2}}(\langle 00| + \langle 11|)$$

tracing out the second half of this state gives:

$$\begin{aligned}
\rho_A &= tr_B(\rho_{AB}) \\
&= \frac{tr_B(|00\rangle\langle 00|) + tr_B(|00\rangle\langle 10|) + tr_B(|00\rangle\langle 01|) + tr_B(|00\rangle\langle 11|)}{2} \\
&= \frac{|00\rangle\langle 00| + |11\rangle\langle 11|}{2} \\
&= \frac{I}{2}
\end{aligned}$$

Recalling Corollary 2.1, the final reduced density operator in the above example is mixed state, indicating that in quantum mechanics it is possible to know a joint quantum state with certainty but be uncertain about its individual subsystems.

Another application of density operators is the notion of *Superoperators*. These *linear* operators acting on density operators and describe how quantum state in the form of density operators evolve. The following theorem characterise superoperators:

Theorem 2.2 *(Superoperator Characterisation) A map between density operators ρ and ρ', $S : \rho \mapsto \rho'$ is a* ***Superoperator*** *if and only satisfy the following conditions:* Superoperators

1. *preserves* Hermitian*: ρ' is Hermitian $\Leftrightarrow$ ρ is Hermitian.*
2. *preserves trace : $tr(\rho') = 1 \Leftrightarrow tr(\rho) = 1$.*
3. *preserves positivity: ρ' is positive $\Leftrightarrow$ ρ is positive.*
4. *Linearity: $S(\rho_1 + \rho_2) = S(\rho_1) + S(\rho_2)$.*

Remark 2.1 *The above characterisation of superoperators is based on* operator-sum *representation, which we shall use in this thesis. However, there is another representation of superoperators known as* Krause representation *(see [71]).*

2.3 QUANTUM COMPUTATION

In this section we explain fundamental concepts of quantum computation. We present *circuit model* as a widely used formalism to describe quantum computation and algorithms.

The basic element of quantum information is *quantum bit* or *qubit*. The state of a qubit q is defined by a state vector $|\phi\rangle_q$ in the Hilbert space $\mathcal{H}^{\otimes 2}$.

Definition 2.2 ***(Qubit)*** *The state of a quantum bit q is defined to be the following state vector in $\mathcal{H}^{\otimes 2}$:* *Quantum bit:qubit*

$$|\phi\rangle_q := \alpha|0\rangle + \beta|1\rangle$$

Where $|\alpha|^2 + |\beta|^2 = 1$, $\alpha, \beta \in \mathbb{C}$. For n qubits, the tensor product *2.1 of individual qubits defines the quantum state i. e. :*

$$|\Psi\rangle = \alpha_1|00\ldots0\rangle + \ldots + \alpha_n|11\ldots1\rangle$$

Such that $\sum_i |\alpha_i|^2 = 1$ for $\alpha_i \in \mathbb{C}$.

A primitive model of quantum computation is *Quantum Circuit Model*. In this model each computational process consists of a number of *Quantum Gates*, operating on qubits. Quantum gates can have arity one or more e. g. Pauli operators (see Figure 2.1) have arity one and controlled *CNot*, *Toffoli* have arity two and three respectively. The Pauli gate X operates as the quantum not gate. Z and Y gates change the phase of a qubit state. *CNot* gate consists of a control qubit and a target qubit, depending on the value of control qubit, it applies X *Quantum Circuits*

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},\ Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},\ I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Figure 2.1: Matrix representation of arity-1 quantum operations

$$CNot = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},\ Toffoli = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 2.2: Matrix representation of arity-2,3 quantum operations

gate on the target qubit. Finally, Toffoli gate has two controlled qubits and depending on both of them applies X gate on the third (target) qubit. Remarkably, *any classical* circuit can be replaced by an equivalent circuit with only Toffoli gates.

In the quantum circuit model, measurement is done using the *Measurement gate*. This gate performs *general measurement* i. e. in the standard basis. The outcome of measurement gate is a *classical* piece of information and *permanently* changed quantum state. It should be noted that often we need to apply a special case general measurement or *projective measurement* (i. e. in other basis than standard basis (see [84, p 87]). In quantum circuit model, this kind of measurement can be achieved using series of unitary gates prior the measurement gates.

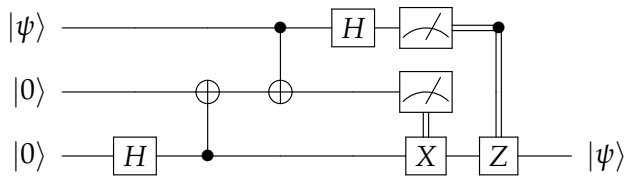


Figure 2.3: Teleportation Circuit

Using a *discrete* set of quantum gates, any quantum circuit can be approximated to an arbitrary precision. In other words, similar to the set of classical gates $\{AND, OR, NOT\}$, there is a set of quantum gates capable of approximating *universal* quantum computation. For example, *Solovay-Kitaev* theorem [84, p 617] states that a circuit with m arbitrary unitaries, can be approximated for any ϵ using only $O(m\, log^c(m/\epsilon))$ gates from the universal set:

$$\{Hadamard, Phase, CNOT, \pi/8\}$$

Quantum circuits usually are depicted with wires as for qubits and boxes for quantum gates (e. g.—$\boxed{X}$—). Double wire appears after quantum measurement, denoting classical outcome of measurement gate (depicted by —$\boxed{\text{⌒}}$═). Controlled gates are shown with circle for control qubit and dots for target qubits. For example the circuit in the Figure 2.3 shows how to perform quantum *Teleportation* [17], using Pauli and measurement gates. We shall present Teleportation protocol in the Chapter 6

Main quantum algorithms such as Shor's algorithms [95] for factoring integer and discrete logarithm are presented using quantum circuits. From complexity point of view, these algorithms belong to the class *BQP*, i. e. the problems which can be decided by a *uniform*

family of *polynomial sized* quantum circuits with *bounded* error probability. The aforementioned Solovay-Kitaev theorem gives a *uniform* construction of quantum circuits. A remarkable result in quantum complexity theory connects quantum computing with classical computing and states that $BQP \subseteq PSPACE$ [19]. More details on *quantum* computational complexity can be found in [19].

2.4 QUANTUM INFORMATION

In the previous section we have presented *closed* QIP system. In reality, we need to deal with *noises*, and in fact this is a challenging part in the implementation of quantum systems. *Quantum Information Theory* investigates the *dynamics* of *open quantum systems* (i. e. systems which are affected by noises). The success of quantum information theory follows from the progresses which have been made during development of *Quantum Error Correction* and *Quantum Fault Tolerant* computation. In this section, the main results in quantum information theory are briefly presented. We start from the model of quantum error corrections, then we look into the quantum fault tolerant computation. The case studies related to these areas are presented in 6. Finally, we introduce the notion of *overlap* of quantum states, arises in many applications of quantum information theory. In particular, by computing *fidelity* between two quantum states, we can measure how close two quantum states are. This of course gives us an alterna-

tive way of testing equality of quantum states and therefore suggests another method for implementing *equality test* in Chapter 5.

Quantum Error Model

In quantum error correction theory, errors or noises are considered to be superoperators $\mathcal{E}$, acting on density operators. This model can describe dynamics of open quantum systems that are weakly or even strongly coupled with environment, assuming that the effect of errors/noises are quantum operations themselves. A useful feature of this model is that it describes changes to quantum states, *discretely*, making it more convenient for formal analysis.

Let ρ be a density operator corresponding to the states which we want to protect with error correcting code and let $\mathcal{E}$ be as above. Let C denote a quantum error correcting code i. e. a subspace of a larger Hilbert state than the ρ's initial Hilbert space (some examples of such codes are given in Chapter 6). Then an error correction protocol is successful if there is a superoperator $\mathcal{R}$, such that it can correct $\mathcal{E}$ to retrieve ρ:

$$(\mathcal{R} \circ \mathcal{E})(\rho) \propto \rho \tag{2.5}$$

The Equation 2.4 is in its most general form, however in our case studies we have performed perfect recovery, that is to say we substitute $\propto$ with equality. The following theorem [84, p 436] formalises the conditions needed for error correction protocols:

Theorem 2.3 *For a quantum code C, and a error correction protocol P which can be thought as a projector onto C and errors $\mathcal{E} = \{E_i\}$, a nec-*

essary and sufficient condition for the existence of correction operator $\mathcal{R}$ is that there is a Hermitian operator α such that

$$PE_i^\dagger E_j P = \alpha_{ij} P$$

Moreover, a remarkable discovery by Calderbank and Shor [27] shows that in fact such an error correcting scheme can always be constructed.

A major application of quantum error correction is in realising fault tolerant quantum computation. The main idea here is to replace qubits in quantum protocols with *encoded blocks* of qubits using quantum error correction codes. Quantum gates has to be modified to operate on encoded blocks of qubits. In this way quantum computation processes can be protected from noisy environment and avoid *de-coherence*. Interestingly, such a fault tolerant schemes exist if the *faults* probabilities are kept low during the computation.

Fault Tolerant Quantum Computation

Theorem 2.4 *(*Threshold theorem*) [84, p 481] Suppose a circuit has $p(n)$ gates, depending on the circuit's input size . Assume the probability that quantum gates fail is at most p and that is always bounded by a constant threshold $p \leq P_{th}$. Then with probability of error at most ϵ, there is a quantum fault tolerant scheme which uses*

$$O(poly(log p(n)/\epsilon)p(n))$$

gates to simulate the circuit on a faulty hardware.

In Chapter 6 we will analyse two protocols which implement fault tolerant $CNOT$ gate.

Comparison of quantum data in terms of information theoretic measures, is a central task in quantum information theory. Several metrics for quantum states have been proposed in the literature, among them is the notion of *fidelity* between quantum states.

Fidelity between quantum states

Definition 2.3 *Let σ and ρ be quantum states, the (Uhlmann) fidelity between two states is defined by:*

$$F(\rho, \sigma) = tr\sqrt{\rho^{1/2}\sigma\rho^{1/2}} \tag{2.6}$$

The following proposition, which is a direct consequence of Uhlmann's theorem [84, p 410], characterizes the properties of fidelity, in particular a criteria for equality of two states. In Chapter 5, an algorithm for testing equality of stabilizer states based on fidelity will be presented.

Proposition 2.1 *Suppose $F(\rho, \sigma)$ denotes the fidelity, according to the Definition 2.3. Then F has the following properties:*

1. $F(\rho, \sigma) = F(\sigma, \rho)$.
2. $0 \leq F(\rho, \sigma) \leq 1$.
3. $F(\rho, \sigma) = 1$ *if and only if* $\rho = \sigma$, $F(\rho, \sigma) < 1$ *if and only if* $\rho \neq \sigma$.

3

FORMAL METHODS IN QUANTUM INFORMATION PROCESSING

Quantum computation and information are mostly studied in the quantum circuit model, i. e. in the level of hardware of quantum devices. With growing complexity of quantum technologies, it has become imperative to develop high level *interfaces*, like programming languages, specifically designed for QIP. In this chapter we review formal methods techniques which have been applied to QIP. We start by introducing Quantum Programming Language (QPL) [94], its syntax and semantics. This language has been used in the early version QEC tool [9]. We review QMC [56], its main features and its limitations. Then we introduce process algebraic languages, such as, CQP [53] and qCCS [102] and the main results surrounding them. We will move on to present the tool Quantomatic [38], based on graphical language for QIP.

3.1 QUANTUM PROGRAMMING LANGUAGE (QPL)

This language is designed by Selinger [94], primarily for describing quantum algorithms. QPL is structured for a quantum device with

```
P,Q ::= input x|output x|newbit b|newqbit q|discard x
skip|P;Q|q*= S|
if b then P else Q end| measure q then P else Q end |
while b do P | proc X:{P} in Q | call X
```

Figure 3.1: Compact Syntax of QPL

classical control, thus it handles both classical and quantum data in order to express both quantum *data* and *control* flow. The main features of QPL are as following:

1. *Mixing classical data and quantum data*.
2. Expressing both quantum data flow and classical control flow, so it can express *recursion* and *loops* as well as classical *conditions* and quantum *measurement*.
3. Having denotational semantics in terms of *superoperators* (see Chapter 2).
4. Designed as a functional language, so it is convenient for describing complex quantum algorithms with many subroutines.
5. QPL is statically typed, thus can be checked against runtime errors.

The syntax of QPL can be described by a textual language and also by quantum flow charts [94]. Figure 3.1 shows the textual syntax of QPL. Quantum programs in QPL are initialised with input variables by `input x`. Quantum variables correspond to qubits can be declared by `newqbit x`, as well as classical variables by `newbit x`. Unitary operators on qubits are represented by `q *= U`. Quantum measurement

Syntax of QPL

```
program Teleportation
input q0:qbit
//Preparing EPR pair.
newqbit q1;
newqbit q2;
q1*=H;
q1q2*=CNot;
//Entangling Alice's qubit.
q0q1*=CNot;
q0*=H;
//Alice's Measurement and Bob's corrections.
measure q0 then q2*=Z else q2*=I end;
measure q1 then q2*=X else q2*=I end
output q0:qbit
```

Figure 3.2: Teleportation in QPL

in QPL is expressed as `measure q then P else Q end`, where each branch corresponds to a classical outcome of quantum measurement. Qbits in QPL are discarded by `discard x`, where it is assumed that there is an operating system which gives or reset access to a finite number of qubits and thus, qubits are never created or dumped in QPL. As an example, Teleportation protocol (see Chapter 6) in QPL is illustrated in Figure 3.2. In [94], the semantics of QPL has been thoroughly investigated. The operational semantics for this sequential language is straightforward and therefore we focus our attention to the denotational semantics of QPL. Each fragment of a program in QPL is described by a *superoperator*, and thus the two primitives of defining denotational semantics, namely *abstraction* and *composition*, are obtained by using properties of superoperator.

Semantics of QPL

In the following we define the domain in which detonations of fragments of QPL are defined. This will be followed by the formal definition of QPL denotational semantics.

Definition 3.1 *(Set of density operators) Set of density operators of dimension n is defined by:*

$$U_n = \{A \in \mathbb{C}^{n\times n} \mid A \text{ positve Hermitian and } tr(A) \leq 1\}$$

Definition 3.2 *(Löner partial order) For matrices $A, B \in \mathbb{C}^{n\times n}$, $A \sqsubseteq B$ if and only if $B - A \geq 0$.*

Lemma 3.1 *The partial order set $(U_n, \sqsubseteq)$ has least upper bounds for monotone sequences.*

Proof *See Proposition 3.6 in [94].*

In particular, Lemma 3.1 implies that the set of density operators with Löner partial order is *well defined*, makes it usable for defining denotational semantics.

Definition 3.3 *(Denotational Semantics of QPL) The semantics of a QPL program $\mathcal{P}$ is defined by a **superoperator** of the form:*

$$[\![\mathcal{P}]\!] : (U_n, \sqsubseteq) \longmapsto (U_n, \sqsubseteq)$$

A key aspect for developing a useful domain specific language of quantum systems is the ability to handle both classical and quantum data at the same time. This is particularity important in QPL, where

we assume there is a classical control in place. In order to mix classical and quantum data, in [94] concepts of *trace*, *adjoints* and *Unitaries* of density operators are extended to *tuples* of density operators, followed by formal procedure of combining classical and quantum data. These steps are shown in the Definition 3.4 and Definition 3.5.

Definition 3.4 *Trace, adjoint and unitaries of the tuples of density operators are defined by:*

- $tr(A_0, \ldots, A_n) := \sum_i tr(A_i)$.
- $(A_0, \ldots, A_n)^* := (A_0^*, \ldots, A_n^*)$.
- $U(A_0, \ldots, A_n)U^* := (UA_0U^*, \ldots, UA_nU^*)$.

Definition 3.5 *Suppose a fragment of a QPL program consist of n bits and m qubits. Then we can represent the semantic of this fragment with the tuple $A = (A_0, \ldots, A_{2^n-1})$ where each A_i is a density operator of the dimension $2^m \times 2^m$. All operations on density operators can be extended as defined in the Definition 3.4. Likewise, superoperators corresponding to the denotational semantics in the Defintion 3.3, with classical data, can be defined on the space of tuples such as A:*

$$[\![\mathcal{P}]\!] : (U_m^n, \sqsubseteq_n) \longmapsto (U_m^n, \sqsubseteq_n)$$

Where $\sqsubseteq_n$ is lexicographic generalisation of $\sqsubseteq$.

The superoperator semantics that which we consider in this work (and is also implemented in [9]) is slightly different than the one presented in [94]. For example, introducing new qubits in this work is

Superoperator	Action
$[\![newqubit(\mathrm{q})]\!]$	$\rho \mapsto \rho \otimes \lvert 0\rangle\langle 0\rvert$
$[\![newbit(\mathrm{b})]\!]$	$\rho \mapsto (0, \rho)$
$[\![\mathrm{b}{:=}1]\!]$	$(0, \rho) \mapsto (\rho, 0)$
$[\![\mathrm{b}{:=}0]\!]$	$(\rho, 0) \mapsto (0, \rho)$
$[\![\mathrm{U(q)}]\!]$	$\rho \mapsto U\rho U^*$
$[\![Output(\mathrm{q})]\!]$	$\rho \mapsto Tr_q(\rho)$
$[\![Measure(\mathrm{q})]\!]$	$\rho \mapsto \rho(p_0 M_0(q) + p_1 M_1(q))$

Figure 3.3: Actions of Superoperator

done by increasing the dimension of the underlying quantum state, whereas in [94], it is assumed that there is a a collection of qubits and a operating system which assigns them to quantum variables. Similarly, deallocation of qubits in [94] is done using the mentioned operating system resetting the access to the storage of qubits, whereas we deallocate quantum variables by *tracing out* qubits, using partial trace function. Our approach however, presents a more realistic model of the current quantum technology devices. Figure 3.3 summarises the actions of superoperators in QPL.

However, the main limitation of QPL for specifying QIP system is its *sequential* structure. Concurrency, especially in the presence of quantum entanglement, needs a more expressive language with concurrent constructs. Languages *CQP* [53], *QMCLang* [56], *qCCS* [102] and CCS^q, which will be introduced in Chapter 4, are examples of concurrent languages.

3.2 QUANTUM MODEL CHECKING (QMC)

A model checking tool for quantum system is developed in [56], [52] and [87]. The modelling language is called *QMCLang*, has an imperative style language where different process are defined separately and can communicate to each other either by sending classical or quantum variables. QMC

Properties in QMC are expressed by formulas of *Quantum Computation Tree Logic (QCTL)* [14], a logic for expressing properties of quantum states. The models in QMC are restricted to those in stabilizer formalism, i. e. only stabilizer states and Clifford operators and measurements are allowed (see Section 5.1). On the other hand, often evaluation of QCTL properties in QMC needs to convert quantum states stabilizer representation to the state vector representation, adding to the complexity of model checking. A model expressing quantum Teleportation in QMC can be found in [52, p 457]. In this model there are three processes *EPR*, *Alice* and *Bob*. A formula *Alice.q* refers to the qubit *q* defined as a variable inside the process *Alice*. History variables are defined as "history *Alice.q*", and stores the quantum state (in the stabilizer representation). Thus, the property for checking Teleportation in QMC is as following:

$$\text{history } Alice.q == Bob.epr2 \tag{3.1}$$

QMC checks the property in Equation 3.1 on all possible interleavings of Teleportation's processes and on all possible stabilizer states, as the input. Models in QMC are constructed using a *process scheduler* and a *model interpreter* which is based on stabilizer formalism. The main features of QMC is summarised as follows:

1. A flat concurrent language, QMCLang, to model quantum processes and communication between them (classical or quantum).
2. QMCLang supports both classical and quantum variables.
3. QMCLang has formal operational semantics [87, p 61].
4. QMCLang has a type system that facilitate type checking.
5. Properties in QMC are expressed with history variables.
6. QMC tool comprises a process scheduler to extracts interleavings, an Interpreter which uses stabilizer simulation and a property interpreter.
7. Properties are checked on all possible interleaving arising from protocols models on all possible *stabilizer states*. It provides a *witness* for the correction of protocols, since checking a continuum of quantum states, as the input, is not feasible.
8. A number of quantum protocols have been modelled and analysed in QMC [87].

For more details on QMC, reader may consult [87] and [52]. It should be noted that checking models in QMC on all possible stabilizer states

can not *guarantee* the correctness of quantum protocols. Also the QMCLang has a flat concurrency structures i. e. a number of processes are defined without *explicit* parallel compositions. Finally, not all desired properties of quantum protocols are easily expressible in QCTL and checkable in QMC, and that is why properties in QMC's case studies share a simple form, rather than QCTL formulas.

3.3 PROBABILISTIC REASONING OF QUANTUM SYSTEMS

Reasoning about probabilistic systems using model checking has been studied extensively in the past, and applied to a range of areas using well developed automated tools such as PRISM [72]. So it seems natural, to use probabilistic model checking in analysing and verification of QIP protocols.

Gay et al. [54] have investigated the application of PRISM tool in verification of quantum protocols. For example in Teleportation, the follow property is specified for PRISM:

$$P \geq 1 \left[\text{true}\, \mathcal{U}\, ((\text{telep-end}) \wedge ((st = s_1) \vee \ldots \vee (st = s_n)))\right]$$

Where *telep-end* is a predicate which by the end of protocols it is **true** and $s_1, \ldots, s_n$ represent finite set of quantum states (a continuum of states can not be verified in this method).

A major challenge in this method is scalability of verifying larger protocols with a higher number of qubits, due to lack of efficient

representation of arbitrary quantum states. Also without a dedicated language, specification of more complicated protocols and their properties becomes a difficult task.

Probabilistic model checking abstracts systems in terms of *Markov Chains*, where they are defined by a (infinite) set of states and probabilistic transition function on the states set. In quantum system, *quantum Markov Chain* is defined on a (infinite) set of *quantum* states (possibly mixed) and a set of superoperators that represent the transition between states. In particular Feng et al. [45], introduced a model checking technique and algorithm for quantum Markov Chains. In their work, normal probability distribution is replaced with superoperator valued distribution [45], for example the specification for QKD protocols in this formalism is as following:

$$s \models P_{\lesssim 0_{\mathcal{H}}}\left[\diamond \,\mathrm{fail}\right] \wedge P_{\gtrsim \frac{1}{2}\mathcal{I}}\left[\diamond^{\leq 4}\,\mathrm{succ}\right]$$

which means QKD never fails and within 4 steps and the probability at least one half, QKD terminates successfully. (here $\lesssim$ and $\gtrsim$ are defined according to superoperator valued distribution [45])

Although, quantum Markov Chains are convenient for specifying quantum cryptography protocols, there is no available tool that implements this technique yet.

$$
\begin{aligned}
&Alice(x : Qbit, c : \hat{}[0..3], z : Qbit) = \\
&\{z, x* = CNot\}.\{z* = H\}.c![measurez, x].0 \\
&Bob(y : Qbit, c : \hat{}[0..3]) = \\
&c?[r : 0..3].\{y* = (\text{case } r \text{ of} 0 \Rightarrow I, 1 \Rightarrow X, 2 \Rightarrow Z, 3 \Rightarrow Y)\}.Use(y) \\
&System(x : Qbit, y : Qbit, z : Qbit) = \\
&(new\, c : \hat{}[0..3])(Alice(x, c, z)|Bob(y, c))
\end{aligned}
$$

Figure 3.4: Teleportation in CQP

3.4 QUANTUM PROCESS CALCULI

In this section we look into two process algebraic formalisms for analysing quantum systems. We will introduce *CQP (Communicating Quantum Processes)* [53] and qCCS (Quantum Communicating Concurrent Processes) [102]. Main features and results around these languages will be reviewed.

Gay and Nagarajan [53] introduced CQP for analysing quantum systems. CQP is designed based on π-calculus, extended with quantum operations and communications. The syntax of CQP has primitive data types: *Int*, *Unit* and *Qubit* for integers, unitary operators and qubits. Channel types are constructed by $\hat{}[T]$ and $Op(n)$ is for n-qubit quantum operations. The full syntax can be found in [53]. For example, Teleportation in CQP is defined in the Figure 3.4, where 0 denote empty process and $Use(y)$ means using variable y in the continuation of the protocol.

CQP

The operational semantics of CQP is defined by reduction rules of the following form

$$t \longrightarrow \boxplus_i p_i \cdot t_i$$

where t and t_i denote configurations of the form (σ, ϕ, e). Here configurations contain process expressions along with quantum state. Also $\boxplus_i$ denotes probability distribution of configurations with probabilities p_i. The full operational semantics of CQP can be found in [53].

Davidson in [33], has studied theory of equivalence between CQP processes, by defining a *bisimulation* relation for quantum processes that is preserved in all contexts, in other words it is a *congruence* relation. This definition is based on *mixed configurations*, i. e. quantum states are in density operator form. The conditions for this definitions consists of *action matchings* and *probability distribution matchings*. Let S be the set of configurations and R be the bisimulation relation as above, and suppose $(s, t) \in R$ for $s, t \in S$. Let $\mu_D(x)$ denote the probability distribution on D. Then the condition regarding probability distributions on configuration in the aforementioned bisimulation definition says that for s and t we must have:

CQP Equivalence

$$\mu_D(s) = \mu_D(t)\,, \forall D \in S \setminus R$$

Thus if actions are matched according to the definition in [33, p 95], then we also need to ensures that the probabilities are paired with configurations consistently. The full proof that this bisimulation relation is also a *congruence* relation, is detailed in [33, Section 4.3].

Main features of CQP and results in the theory CQP, are summarised as following:

1. A process calculi for specifying quantum communication and cryptography protocols.

2. It is designed based on π-calculus.

3. It handles both classical and quantum data.

4. It has a full type system.

5. Operational semantics is defined by reduction rules, similar to the π-calculus.

6. Bisimulation relation, which is also proved to be congruence, is defined for CQP in [33].

7. There is no available tool support for CQP.

Now we review the process calculi *qCCS* developed by Ying et al. [102]. In the early version of *qCCS* only quantum variables were involved, in contrast to CQP with both classical and quantum variables, however, classical features are added to the new version [102], [44]. he quantum interactions in *qCCS* has a more general form in terms of *superoperators* rather than *unitary* operators. This makes *qCCS* more expressive when it comes to describe open quantum systems, e.g. systems with noises. The syntax of *qCCS* is based on *CCS* [81] and is outlined in [102]. In particular, for a set of quantum variables x and a *qCCS* process P, prefixes of the form $\mathcal{E}(x).P$, where $\mathcal{E}(x)$ denotes a superoperator $\mathcal{E}$ acting on x, are defined. As an example, Teleportation in *qCCS* is shown in the Figure 3.5, where $\mathcal{CN}$, M and σ^i denote Cnot, measurement and Pauli operators. Operational semantics of *qCCS* is

qCCS

$$Alice = c?q.\mathcal{CN}[q, q_1].\mathcal{H}[q].M[q, q_1 : x].e!x.nil$$
$$Bob = e?x. \sum_{0 \leq \leq 3} (\text{if } x = i \text{ then } \sigma^i[q2].d!q2.nil)$$
$$Tel = (Alice \parallel Bob) \setminus \{e\},$$

Figure 3.5: Teleportation in qCCS

defined using configurations of the form $\langle P, \sigma \rangle$, where P is a $qCCS$ process expression and σ is a density operator.

The theory of equivalence of $qCCS$ is studied extensively in a series of papers [44], [46]. Different bisimulation relations for $qCCS$ are defined, and proved to be congruence. These relations are defined on two primitives of *action matching* and *probability distribution matching*. The difference with CQP bisimulation is that in case of quantum input/output action, the bisimulation relation should be preserved for all superopeator acting on bisimilar configurations, and yet superoperators form a continuum, make it impossible to construct bisimulation relation computationally. However, in [46] a *symbolic bisimulation* relation for $qCCS$ is introduced, along with a variant of bisimulation relation, called *ground symbolic bisimulation*. For the latter bisimulation, an algorithm proposed that only works on *quantum input free* process [46]. For example in the Figure 3.5, if we remove $c?q$ from Alice and $d!q_2$ from Bob, the results is quantum input free Teleportation. We summarise $qCCS$ main features and results in the following:

qCCS Equivalence

1. A quantum process algebra for describing quantum systems.

2. Designed based on CCS, extended to include superoperators.

3. Bisimulation is defined and proved to be congruence, using an infinite set of superoperators.

4. Symbolic bisimulation, and an algorithm for ground symbolic bisimulation is introduced.

5. No tool support is available for *qCCS*.

3.5 QUANTOMATIC

In their seminal paper [6], Abramsky and Coecke have developed a novel approach for reasoning about quantum systems, based on a category-theoretic formulation of quantum mechanics. This elegant formulation of quantum mechanics, as it is called *categorical quantum mechanics*, provides a high level understanding of QIP systems.

Inspired by their work, diagrammatic reasoning techniques, based on category theory, have been developed by Coecke and Duncan in [31]. Furthermore this idea has been implemented in the tool *Quantomatic* [38], that uses graph rewriting in order to automate diagrammatic reasoning about underlying categorical structure.

The input of Quantomatic tool is graphical, in contrast to our tool which has programming language interface. Also, our tool verifies quantum protocols in a fully automatic way, whereas Quantomatic is a semi-automatic tool that needs a considerable amount of user intervention. This point is recently highlighted by Duncan in a case

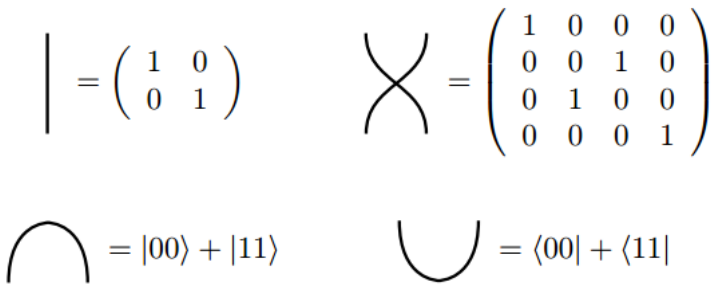


Figure 3.6: ZX Wires and their interpretation

study, that verifies Steane Quantum Error Correction code [97] with Quantomatic [39].

The graphical interface of Quantomatic uses ZX-language [31], a network of components joined by wires, similar to circuit diagrams. The wires can be bended or braided, each have a particular meaning and components may contain dots or boxes. In this graphical representation, dots with red colour indicate Z-basis whereas dots in green denote X-basis.

A diagram $\mathfrak{D}$ in ZX-language, can be interpreted by an operator $D : Q^n \longrightarrow Q^m$, with n qubits as input and m output qubits. For example, Figure 3.6 which shows the interpretation of ZX-language wires in Hilbert space.

As an example, a proof of the correctness of Teleportation protocol [31], specified in ZX-language can be seen in the Figure 3.7. The boxes and dots corresponds to Alice and Bob's operations. The goal is to show equivalence of Teleportation to identity using diagrammatic reasoning, where Quantomatic automates parts of this process. Comparing to our QEC tool, one can see that Quantomatic does not perform simulation of quantum states and operations and treat them

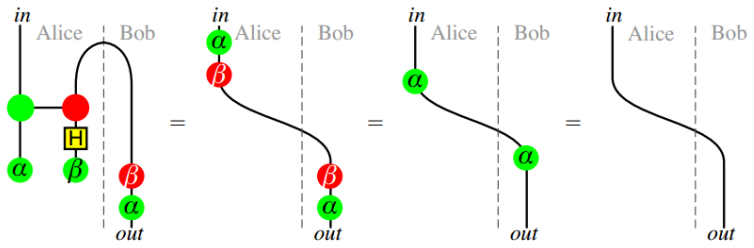


Figure 3.7: Diagrammatic proof of Teleportation

in a very high level manner. On the other hand, QEC adopts model checking rather theorem proving, so potentially can provide counter example, in case a protocol could get wrong. We summarise main feature of Quantomatic as follows:

1. High level graphical interface based upon categorical quantum mechanics, for specification of QIP systems.

2. Provides proof of correctness for protocols such as Teleportation, hence can be thought of as a quantum proof assistant.

3. Semi-automatic verification of QIP protocols, using graph rewriting techniques.

Duo to technicality of categorical quantum mechanics, that is beyond the scope of this thesis, we refer the reader to [6], [31] and [38], for more details.

4

SPECIFICATION OF CONCURRENT QUANTUM PROTOCOLS: SYNTAX AND SEMANTICS

In this chapter we discuss how to specify concurrent quantum protocols. Our goal is to design a concurrent language suitable for applying equivalence checking for the analysis of quantum protocols. To this end, we introduce a process algebraic language, CCS^q, for the specification of concurrent quantum protocols. It shares many features of the languages described in Chapter 3, such as inclusion of both classical and quantum data and also considers concurrency. However, it differs from QMCLang [55], by having a more general model of concurrency, i. e. processes in CCS^q are defined with *explicit* parallel compositions. Alos, in contrast to QMCLang, the semantics of CCS^q is defined by superoperators. This makes verification of QIP systems with arbitrary input, not just stabilizer states, feasible.

The rest of chapter is organised as follows: first we introduce CCS^q by presenting its concrete syntax and discuss the main features of the language. Secondly, we define a full operational semantics in the style of *reduction semantics*. In the third section, we explain how superoperators can be used to define a new semantics for our language, in addition to the operational semantics.

4.1 SYNTAX

In this section we present the syntax of CCS^q. Our focus in this thesis is to understand the role of concurrency in quantum system, therefore we made the syntax compact and simple. One reason to use this language is to illustrate how designing concurrent quantum protocols can be difficult and non intuitive compared to the design of classical protocols. For example, quantum measurement is a destructive action, so if a qubit is measured wrongly between parallel processes, this can destroy the effect of the whole system.

The protocols that we analyse in CCS^q are *functional*, i. e. they receive input at the beginning and produce output at the end of their execution. This is the reason we have input and output declaration of qubits in our language .

Although the current syntax is expressive enough to analyse our case studies in Chapter 6, it can be extended to include *loops* and *recursion*, while maintaining superoperator semantics in a similar way to QPL [94]. Now we explain the main constructs of our language: *CCS^q Syntax*

- There are two types of variables: classical *bits*, represented by V and *qubits* are shown by Q in the syntax.
- Here L denotes lists of matching variables, which are used in match conditionals.
- S is a lists of quantum variables.

- Expressions (E) consist of unitary operators, measurement, input/output and conditionals.

- Input and output variables are represented by a prefix of the form **input/output** S, where they indicate initialisation or finalisation of the protocols execution. Similar to functional programs, we specify the input and output of protocols explicitly. The difference is that we are computing a quantum function and therefore the input action creates a quantum state. The output action halts the execution and returns a quantum state. often by applying partial trace to a larger state.

- Unitary operator U on a qubit r is denoted by the prefix $\mathbf{U}(r)$. Similar to QMC, in QEC we only deal with Clifford operators and thus it is assumed that unitary U belongs to the set: $\{CNOT, H, P, X, Y, Z\}$ (see Section 2.3). Nonetheless, the semantic analysis in this chapter applies even if we have arbitrary unitaries.

- Quantum measurement is carried out by an expression of the form `x:= measure p`, where the possible outcomes of measuring a qubit p are assigned to a classical variable x.

- There are two types of conditionals, one with a single condition on a classical variable such as x, where the qubit q is the target qubit and is denoted by **if** x **then** $\mathbf{U}(q)$. The other one, imposes multiple conditions by matching a list of variables and values of the form **match** $\{x_i : B\}$ **then** $\mathbf{U}(r)$, where r is a qubit variable.

- Process terms (P) are defined via prefixes in a similar way to the original *CCS* [81]. The simplest process with no action is the *nil* process.

- Sending and receiving a bit x over a classical channel c is done using prefixes $c!x$ and $c?x$. Similarly for a qubit q over a quantum channel d, we use $d!q$ and $d?q$. Note that this *explicitness* of communication is not realised in the circuit diagrams, making it difficult to specify concurrent protocols in terms of quantum circuits.

- Finally, parallel composition is described by expressions of the form "$(P \mid Q)$", where P and Q are process expressions. We use brackets for multiple parallel compositions e. g. $P \mid (Q \mid R)$. This is another useful feature of our language that represents parallel compositions explicitly, enabling us to add concurrency to the specification of quantum protocols and analyse their behaviour.

The syntax of CCS^q in Backus-Naur form is illustrated in Figure 4.1. As an example of usage of our language, Figure 4.2 shows implementation of Teleportation in CCS^q (see Chapter 6 for more details).

Note that although CCS^q has a simple structure, it is much more expressive than quantum circuits which is widely used throughout QIP literature. In this thesis we mainly concern with quantum communication and concurrency rather than quantum computation. However,

$$
\begin{aligned}
B &::= 0 \mid 1 \\
V &::= x \mid y \mid \ldots \\
Q &::= p \mid r \mid \ldots \\
L &::= V : B \mid L, V : B \\
S &::= Q \mid S, Q \\
E &::= V \textbf{:= measure } Q \mid \mathbf{U}(Q) \mid \textbf{if } V \textbf{ then } \mathbf{U}(Q) \mid \\
&\qquad \textbf{match } L \textbf{ then } \mathbf{U}(Q) \mid \textbf{newqubit } Q \mid \textbf{input } S \mid \textbf{output } S \\
P &::= nil \mid (P \mid P) \mid V\textbf{!}V.P \mid V\textbf{?}V.P \mid Q\textbf{!}Q.P \mid Q\textbf{?}Q.P \mid E.P
\end{aligned}
$$

Figure 4.1: Syntax of CCS^q

```
//Preparing EPR pair and sending to Alice and Bob:

newqubit y . newqubit z . H(y) . CNOT(y,z) . c!y . d!z .
nil

|

//Alice's process:

(input x . c?y . CNOT(x,y) . H(x) . m := measure x .

n := measure y . b!m . b!n . nil

|

//Bob's process :

d?w . b?m . b?n . if n then X(w) . if m then Z(w) .
output w . nil)
```

Figure 4.2: Quantum Teleportation

it is desirable to add useful constructs such as loops, functions and recursion to CCS^q in the future.

4.2 OPERATIONAL SEMANTICS

In this section, the operational semantics of CCS^q is studied in the framework of *reduction rules*, defined by small step evaluation of expressions, inter-process communication and non-deterministic transitions. This approach has been adapted when defining semantics of *CQP* [53], with the difference that in this work we do not consider probabilities explicitly, and only consider non-deterministic behaviour arising from quantum measurements and parallel composition. In the following, we give necessary definitions and then present reduction rules for the operational semantics.

Definition 4.1 (Configuration) *Configurations of a concurrent quantum system are defined by tuples of the form* (σ, ρ, R) *where* σ *denotes the assignment of variables to classical values (boolean for classical bits and integers for storing the index of qubits within a quantum state) and* ρ *represents a density operator, whose dimension is determined by the number of qubit variables. Finally,* R *represents a process term according to the grammar of* CCS^q.

Reduction relations are defined on *configurations* and have the following general form:

$$\mathcal{S} \longrightarrow \mathcal{T}$$

$$P \mid nil \equiv_{sc} P \qquad \text{(SC-NIL)}$$

$$P_1 \mid (P_2 \mid P_3) \equiv_{sc} (P_1 \mid P_2) \mid P_3 \qquad \text{(SC-ASSOC)}$$

$$P_1 \mid P_2 \equiv_{sc} P_2 \mid P_1 \qquad \text{(SC-COM)}$$

Figure 4.3: The axioms of structural congruence

where $\mathcal{S}$ and $\mathcal{T}$ are configurations, and $\longrightarrow$ denotes a reduction relation.

We use *structural congruence* in Definition 4.2, in order to simplify process expressions at the top level.

Definition 4.2 *(Structural congruence) The smallest relation that satisfies axioms in Figure 4.3, is called* structural congruence *and denoted by:* $\equiv_{SC}$.

Let τ represent silent action and $[v/x]$ be the substitution of variable v with variable x, so with this notation, $Q[v/x]$ means in the process Q, occurrences of variable v is substituted by variable x. We denote a list of variables by the notation $\widetilde{x}$.

Figure 4.4 presents reduction rules for configurations. Here $i(q)$ denotes the index of qubit q to which an operation or measurement is applied on. Therefore, by $\rho^{i(q)}$ we mean the sub-matrix of the global state ρ, corresponding to the qubit q. In the measurement rule, the "$m's$" stands for a boolean type outcome and $\boxplus$ denote a probability distribution over m. Note that in this work we do not deal with probabilities explicitly. Immediately after the measurement rule, (R-NONDET) is applicable and it reduces the configuration to nondeterministic choices i. The rule R-COND applies a unitary conditional

to the boolean condition b. Note that these rules are defined on configurations as in Definition 4.1.

The rule (R-CONG) is defined according to the axioms of Figure 4.3. Communication and parallel composition follow (R-COM) and (R-PAR), respectively.

Synchronisation

Remark 4.1 *In this work communication between quantum processes are done using synchronisation, which is reflected in the reduction rule R-COM. One of the reasons for considering this model is that the current quantum technologies do not have durable quantum memory, necessary to implement buffers for asynchronous communications.*

(R-QUBIT), specifies how a new qubit is declared in a protocol. The reduction rule (R-INPUT) specifies how a protocol is initialised with a basis density operator ρ_B at the beginning of the execution of the protocol. For the output, however, (R-OUTPUT) reduces the system to terminal configurations by applying a partial trace to the current quantum state. Therefore, after applying this reduction, process configurations become irreducible. These configurations may have *mixed* quantum states, which are denoted by ρ^*.

4.3 SUPEROPERATOR SEMANTICS

In this section we define a *Superoperator Semantics* of our language CCS^q. This is an important step in the equivalence checking since it provides a useful abstraction of the behaviour of concurrent quan-

$$(\sigma, \rho, U(q).P) \longrightarrow (\sigma, U\rho^{i(q)}U^\dagger, P) \quad \text{(R-UNIT)}$$

(where $i(q)$ is the index of qubit q in ρ)

$$(\sigma, \rho, \text{if } b \text{ then } U(q).P) \longrightarrow (\sigma, U^b\rho^{i(q)}(U^\dagger)^b, P) \quad \text{(R-COND)}$$

(where $i(q)$ is the index of qubit q in ρ)

$$(\sigma, \rho, a := \text{measure}\, q.P) \longrightarrow \boxplus_{0\leq m\leq 1}(\sigma[a/m], M_m\rho^{i(q)}M_m^\dagger, P) \quad \text{(R-MEASURE)}$$

(where $i(q)$ is the index of qubit q in ρ)

$$\boxplus_i\,(\sigma_i, \rho_i, P_i) \longrightarrow (\sigma_j, \rho_j, P_j) \text{ (where } 0 \leq i, j \leq 1\text{)} \quad \text{(R-NONDET)}$$

$$\frac{P \equiv_{sc} P' \;\; (\sigma, \rho, P') \longrightarrow (\sigma', \rho', Q') \;\; Q \equiv_{sc} Q'}{(\sigma, \rho, P) \longrightarrow (\sigma', \rho', Q)} \quad \text{(R-CONG)}$$

$$(\sigma, \rho, c!v.P \mid c?x.Q) \longrightarrow (\sigma', \rho, P \mid Q[v/x]) \quad \text{(R-COM)}$$

(where σ' is the updated classical store)

$$\frac{(\sigma, \rho, P) \longrightarrow (\sigma', \rho', P')}{(\sigma, \rho, P \mid Q) \longrightarrow (\sigma', \rho', P' \mid Q)} \quad \text{(R-PAR)}$$

$$(\sigma, \rho, (\texttt{newqubit}\; x).P) \longrightarrow (\sigma', \rho \otimes |0\rangle\langle 0|, P[q/x]) \quad \text{(R-QUBIT)}$$

(where q is a fresh variable and σ' is the updated classical store)

$$(\sigma, \rho, (\texttt{input}\; \widetilde{x}).P) \longrightarrow (\sigma', \rho \otimes \rho_B, P[\widetilde{q}/\widetilde{x}]) \quad \text{(R-INPUT)}$$

(where $\widetilde{q}$ are fresh variables, σ' is the updated classical store and ρ_B is a basis state)

$$(\sigma, \rho, (\texttt{output}\; \widetilde{x}).P) \longrightarrow (\sigma', \rho^*, nil) \quad \text{(R-OUTPUT)}$$

(where $\rho^* = tr_{\widetilde{x}}(\rho)$)

Figure 4.4: The reduction rules for process configurations

tum protocols. We will define *functional* quantum protocols, which are special cases of quantum protocols with a simpler structure. Although *functionality* of quantum protocols seems like a restriction, we will show in the Chapter 6 that many important QIP protocols are indeed functional.

Executing a concurrent quantum protocol, as formalised by the operational semantics in the previous section, will inevitably produce many *interleavings* (they are defined in the following). Due to quantum measurement, probabilities may be associated to interleavings, thus in general our knowledge about the configuration of a quantum system at a given point could be *probabilistic*. To address this issue *mixed configurations* were defined in [33], which are probability distributions over configurations of *CQP*. On the other hand Ying et al. [102] introduced *Superoperator Valued Distributions (SVD)* that formalises concurrent quantum systems defined by *qCCS*.

In this work, we use superoperators to describe the behaviour of interleavings arising from execution of concurrent protocols. In the following, we first define equivalence between superoperators, and then discuss how the semantics of concurrent protocols can be captured by superoperators. Finally, we define functional protocols, whose we are able to establish their equivalence computationally through the equivalence of corresponding superoperators.

The following definition explains what we mean by equivalence between superoperators:

Definition 4.3 *Let S be a set of density operators as defined in Section 2.2.2, then two superoperators $\mathcal{E} : S \longrightarrow \mathcal{E}(S)$ and $\mathcal{F} : S \longrightarrow \mathcal{F}(S)$, are said to be equivalent (denoted by $\approx$) if:*

Superoperators Equivalence

$$\mathcal{E} \approx_S \mathcal{F} \;\Leftrightarrow\; \forall \rho \in S,\, \mathcal{E}(\rho) = \mathcal{F}(\rho)$$

Definition 4.4 *(Configuration Tree) For a given concurrent quantum protocol in CCS^q, let $\mathcal{R}$ be the set of all reduction rules in the Section 4.2. Then, the Configuration Tree $\mathcal{CT}$ is defined recursively such that the root is the initial configuration and the children are defined using the rules in $\mathcal{R}$. The leaves consists of terminal configurations, i.e. no rules from $\mathcal{R}$ is applicable to them.*

Configuration Tree

Definition 4.5 *(Interleaving) Each path from the root of a configuration tree to the leaves, represents an interleaving of a concurrent protocol, consisting a sequence of configurations, obtained by applying a sequence of reduction rules. Every interleaving has one application of (R-INPUT), parametrised with the input quantum state. Final configurations are the result of applying (R-OUTPUT) and brings the execution of the protocol to the halt. A schematic representation of interleaving is given in Figure 4.5.*

Definition 4.6 *Let V be the space of density operators, associated to a set of qubits of a concurrent protocol $\mathcal{P}$ (i.e. the tensor product of each qubit Hilbert space), and let $\mathcal{I}$ denote the set of their interleavings (according to Definition 4.5). We define a function $\Delta_{\mathcal{P}} : V \longrightarrow V$, which for all $i \in \mathcal{I}$ of $\mathcal{P}$, takes initial quantum states ρ_{input} and returns final states ρ_{output}:*

$$\Delta_{\mathcal{P}}(\rho_{input}) = \rho_{output}$$

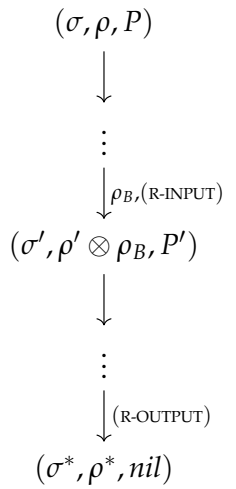


Figure 4.5: Interleaving

Then we define $\mathcal{D}_V(\mathcal{P})$ to be the set of all $\Delta_{\mathcal{P}}$s:

$$\mathcal{D}_V(\mathcal{P}) = \{\Delta_{\mathcal{P}}(\rho) \mid \rho \in V\}$$

Suppose we have a CCS^q protocol $\mathcal{P}$ such that its set of variables corresponds to the space of density operators V and its set of interleavings $\mathcal{I}$ is defined based on Definitions 4.4 and 4.5. As a consequence of executing $\mathcal{P}$, we obtain $\Delta_{\mathcal{P}} \in \mathcal{D}_V(\mathcal{P})$, However, this does not immediately associate $\Delta_{\mathcal{P}}$ to the superoperators since we need to check that whether it has the properties of superoperators according to Theorem 2.2. Indeed this involves working explicitly with complex matrices and real numbers. In the following we show how to avoid this problem by restricting protocols to those who have *I/O* structure, i. e. for each protocol there is an input declaration of variables (quantum/classical) and there is an output declaration that shows the end of the protocol and also have deterministic behaviour, which we call it functional and will be defined in Definition 4.10.

Definition 4.7 *A concurrent quantum protocol, containing a number of processes, is called I/O if there is a unique input declaration of quantum (classical) bits at the beginning of one of its processes, and only one output declaration of qubits (bits) appears at the end of one of its processes.*

The benefit of defining interleavings for the I/O protocols of Definition 4.7 is that we can translate them into sequential QPL programs. This will enable us to associate to each interleaving a superoperator as in [94], without directly verify the properties of superoperators in Theorem 2.2.

Translating Interleavings to QPL

In the following section we will explain how this translation from interleavings to QPL programs is defined.

Remark 4.2 *We have separated classical and quantum data in Definition 4.1 for a simpler formulation. Nonetheless, classical and quantum data can be mixed in a similar way as [94], using tuples of density matrices.*

4.3.1 *Translating Interleavings to QPL programs*

The translation to QPL is done using a function from the set of interleavings (where we will revisit the definition of interleaving in Definition 4.9)to a subset of QPL programs. In order to define this function, first we define the *Transition System* for CCS^q protocols. The reason for having this definition is to obtain a *syntactical* construction of interleavings which is independent from quantum states (in contrast

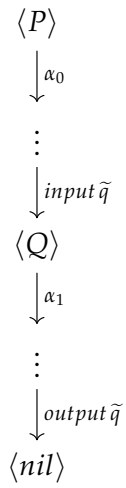


Figure 4.6: Interleaving in Transition Tree

to Definition 4.5), and is suitable for translation to QPL. Figure 4.6 shows the general form of such interleavings.

Definition 4.8 *(Transition System) A transition system for a CCS^q program is defined by a tuple of the form $(S_p, Act, \longrightarrow)$, in which:*

Transition System for CCS^q

1. *S_p is a set of CCS^q process terms as states. We denote states by $\langle s \rangle$, where s is a process term.*

2. *Act is a set of actions such as unitary operations, measurements, etc., including τ (silent) action.*

3. *$\longrightarrow \subseteq S_p \times Act \times S_P$. We denote $(\langle s \rangle, \alpha, \langle s' \rangle) \in \longrightarrow$ by $\langle s \rangle \xrightarrow{\alpha} \langle s' \rangle$. Here transition relations are defined by the rules in Figure 4.7.*

Definition 4.9 *(Transition Tree and Interleavings) Executing a CCS^q protocol results in a* transition tree*, where nodes are states (process terms) and edges are labelled with actions, complying with the rules in Figure 4.7. Moreover each path from the root of the transition tree to a leaf (e. g. an instance of*

Transition Tree & Interleavings

$$\frac{}{\langle input\ \widetilde{x}.P\rangle \xrightarrow{input\ \widetilde{x}} \langle P\rangle} \quad \text{(r-input)}$$

$$\frac{}{\langle U(q).P\rangle \xrightarrow{U(q)} \langle P\rangle} \quad \text{(r-unit)}$$

$$\frac{}{\langle a := measure\ q.P\rangle \xrightarrow{meas(q,a)} \langle P\rangle} \quad \text{(r-measure)}$$

$$\frac{}{\langle \text{if } b \text{ then } U(q).P\rangle \xrightarrow{cond(b,U(q))} \langle P\rangle} \quad \text{(r-cond)}$$

$$\frac{}{\langle c!v.P \mid c?x.Q\rangle \xrightarrow{\tau} \langle P \mid Q[v/x]\rangle} \quad \text{(r-com)}$$

$$\frac{\langle P\rangle \xrightarrow{\alpha} \langle P'\rangle,\ \alpha \in Act}{\langle P \mid Q\rangle \xrightarrow{\alpha} \langle P' \mid Q\rangle} \quad \text{(r-par)}$$

$$\frac{}{\langle output\ \widetilde{x}.P\rangle \xrightarrow{output\ \widetilde{x}} \langle nil\rangle} \quad \text{(r-output)}$$

Figure 4.7: Transition rules for CCS^q transition system

the state $\langle nil \rangle$) corresponds to an interleaving *as defined in Definition 4.5. For the translation we use this construction (paths in transition tree) as the definition of interleaving, which has the general form as in Figure 4.6 (compare it to Figure 4.5).*

Every path i in the transition tree is labelled by the sequence of its edges labels (i.e. actions t_i, including τ). We use the following notation to represent paths: $i := t_1 \,.\, t_2 . \,\ldots\, . t_n$

Translation Function

Let $\mathcal{I}^*$ be a set of interleavings (in the transition tree) and $\mathcal{P}_{QPL}$ be a set of QPL program, the translation function:

$$\mathcal{T}_{QPL} : \mathcal{I}^* \longrightarrow \mathcal{P}_{QPL}$$

is defined as the following, where Γ and Γ' are environments (as defined in [94]), p represents a QPL program and $i \in \mathcal{I}^*$ is path in the transition tree corresponding to an interleaving:

$$\mathcal{T}_{QPL}(i) = (\Gamma, p, \Gamma')$$

we also need an auxiliary function T' that returns p and the outbound environment:

$$T'(i) = (p, \Gamma')$$

For each path, input/output actions *fix* inbound and outbound environments of the final translated QPL program. So we have the following equation, where T' is defined as in Figure 4.8:

$$\mathcal{T}_{QPL}(input\, \widetilde{x}.t) = (\widetilde{x} : qbit\,, p,\, \Gamma') \tag{4.1}$$

Where $T'(t) = (p, \Gamma')$.

1.$T'(\tau.t) = (skip; p', \Gamma')$ where $T'(t) = (p', \Gamma')$
2.$T'(output\ \widetilde{x}.t) = (skip\ ,\ \widetilde{x} : qbit)$
3.$T'(U(q).t) = (q\text{*=}U; p', \Gamma')$ where $T'(t) = (p', \Gamma')$
4.$T'(cond(b, U(q)).t) = (if\ b\ then\ q\text{*=}U\ else\ skip; p', \Gamma')$ where $T'(t) = (p', \Gamma')$
5.$T'(meas(q, a).t) = (newbit\ a; measure\ q\ then\ a := 1\ else\ a := 0; p', \Gamma')$ where $T'(t) = (p', \Gamma')$

Figure 4.8: Auxiliary translation function T'

Remark 4.3 *For I/O protocols (see Definition 4.7), we are assured that each path in the transition tree will eventually leads to an output action, and that guarantees the soundness of our translation.*

Example: Suppose we have the following concurrent protocol that we want to translate to QPL:

$$input\ x.H(x).c!x.nil\ \mid\ c?y.H(y).output\ y.nil$$

Then Figure 4.9 shows the transition system for this protocol, where its only interleaving is described by the path i:

$$i := input\ x.H(x).\tau.H(x).output\ x$$

By applying translation function $\mathcal{T}_{QPL}$ to i, and using the Equation 4.1, we obtain the following QPL program:

$$T(i) = x : qbit\ ,\ x* = H; skip; x* = H; skip\ ,\ x : qbit$$

$$\langle input\, x.H(x).c!x.nil \mid c?y.H(y).output\, y.nil\rangle$$
$$\downarrow input\, x$$
$$\langle H(x).c!x.nil \mid c?y.H(y).output\, y.nil\rangle$$
$$\downarrow H(x)$$
$$\langle c!x.nil \mid c?y.H(y).output\, y.nil\rangle$$
$$\downarrow \tau$$
$$\langle nil \mid H(x).output\, x.nil\rangle$$
$$\downarrow H(x)$$
$$\langle nil \mid output\, x.nil\rangle$$
$$\downarrow output\, x$$
$$\langle nil \mid nil\rangle$$

Figure 4.9: Example of Transition System

Remark 4.4 *Note that in QPL [94], conditionals of the form "match then" are not defined, however, one can translate them into a series of nested QPL "if then else" conditions.*

Functionality

Having defined the translation function $\mathcal{T}_{QPL}$, now we define *functional* protocols, those we are able to associate them to the superoperators. In these protocols, all different interleavings exhibit *deterministic* with respect to input/output relation . Interestingly, it turns out many useful QIP protocols are in fact functional. In Chapter 5 we will present an algorithm for checking *functionality* and then in Chapter 6 , examples of functional protocols will be verified.

Definition 4.10 *An I/O quantum protocol is called* ***functional*** *if for its input density operators space V and the set of interleavings $\mathcal{I}$, we have:*

$$\forall i, j \in \mathcal{I} \;\; [\![\mathcal{T}_{QPL}(i^*)]\!] \approx_V \;\; [\![\mathcal{T}_{QPL}(j^*)]\!]$$

Where i^* and j^* are corresponding characterization of i and j according to Definition 4.9.

Definition 4.10 says that by executing a functional protocol, all interleaving will have the same effect on a given input density operator and thus this effect can be described by a unique superoperator $[\![\mathcal{T}_{QPL}(i^*)]\!]$, which is independent from the choice of i^*.

Superoperator Semantics

Finally, in the following a superoperator semantics is defined for concurrent functional I/O protocols.

Proposition 4.1 *Assume for a functional I/O protocol $\mathcal{P}$, we are given sets of interleavings $\mathcal{I}$ according to Definition 4.5 and $\mathcal{I}^*$ based on Definition 4.9. Let $i^* \in \mathcal{I}^*$ and $\Delta_{\mathcal{P}}$ be as Definistion 4.6. Suppose V is the space of density operators (corresponding to the protocol's input variables) and $[\![\,]\!]$ denote the superoperator semantics of a QPL program in [94]. Then we have the following:*

$$\forall \rho \in V \;\; \Delta_{\mathcal{P}}(\rho) = \;\; [\![\mathcal{T}_{QPL}(i^*)]\!](\rho)$$

Remark 4.5 *In Proposition 4.1, we assumed there is a correspondence between the two definition of interleavins presented in this thesis. However, the structure of interleavings in Definition 4.5 is more complicated than the one in Definition 4.9, due to dealing with quantum states. For example, in the case of measurement, an application of (R-MEASURE) together with (R-NONDET) in the interleaving of Definition 4.5, corresponds to a single application of "(r-measure)" in the interleaving of Definition 4.9.*

Correspondence of two definitions of interleavings

Corollary 4.1 *Any concurrent quantum protocol in CCS^q, which is **I/O** and **functional** can be described by a unique superoperator.*

4.4 CONCLUDING REMARKS

The design of domain-specific languages for QIP systems is a challenge since there exist more complex layers from low level physical (sub-atomic particles) structures to high level classical interfaces. As a consequence, many languages have been developed such as those mentioned in the Chapter 3, for different applications. In this work we have used a simple language for the verification of concurrent system by automated equivalence checking, used in the QEC tool. The ability of specifying communications and parallel compositions, makes CCS^q much more expressive than commonly used circuit diagrams.

There are some aspects of our language that relates to the underpinning architectural structure, for example, in this thesis allocation and de-allocation of qubits are interpreted as extending the global quantum state of a system and calculating a reduced quantum state whereas in QPL [94], it is assumed that qubits are not created nor traced out but there is a special operating system which can give or reset access to qubits.

We have studied the behaviour of concurrent systems, specified in CCS^q, using operational and superoperator semantics where we have considered interleaving concurrency, as the underlying model. This involves synchronisation between quantum processes when communications occur. Now, it would be interesting to investigate other models of concurrency, e. g. those with casual structure or true con-

currency [82] and see whether they can be defined in the framework of superoperators.

5

VERIFICATION OF CONCURRENT QUANTUM PROTOCOLS BY EQUIVALENCE CHECKING

In this chapter we describe the core verification technique used in QEC tool. Our approach is based on *process-oriented* model checking, that is to verify the implementation of a QIP protocol behaves *equivalently* to its specification. This is in contrast with QMC [55], where *property-based* model checking is adopted. Of course the main challenge of applying model checking is *explosion of states*, however another challenge in model checking of QIP systems is *explosion of space*, i. e. the space needed to store quantum states representation grows exponentially in the number of qubits.

In order to tackle explosion of space we confine ourself to the stabilizer formalism, the part of quantum mechanics in which our verification algorithms and system take advantage of compact representation and efficient simulation algorithms for certain kind of quantum states, known as *stabilizer states*. We also develop new algorithms based on stabilizer formalism that help us in the developing equivalence checking technique and QEC tool.

The chapter is organised as follows: we start from an introduction of *stabilizer formalism* in Section 5.1, where we review stabilizer simu-

lation and normal forms algorithms. Then in Section 5.2, we present algorithms to test equality of stabilizer states. Section 5.3 describes the construction of stabilizer basis for the space of density matrices of n-qubit states. The Equivalence checking algorithm for sequential protocols is presented in Section 5.4.

The algorithm that checks equivalence of specification and implementation of*concurrent* protocols, specified in CCS^q, is explained in Section 5.5. These algorithms make use of the *stabilizer basis* that spans the space of density operators, and is outlined in Section 5.3. Finally, we discuss the approximation of our equivalence notion, using the algorithms in Section 5.1.2.

5.1 STABILIZER FORMALISM

In this section we explain the main concepts and algorithms about *Stabilizer Formalism*, that is a simple, yet powerful model for describing the underlying physical structure of many important QIP protocols which are studied in this thesis.

Stabilizer Formalism was first introduced by Gottesman in [58], as a formalism to describe *Stabilizer Codes* [58], i. e. the quantum version of *classical linear codes* and has found numerous applications in quantum error correction, quantum complexity and other areas of QIP. The main idea of stabilizer formalism is to represent certain quantum states by their *Stabilizer Group*, instead of an exponential number of

complex amplitudes. Formally, for a stabilizer state $|\phi\rangle$, the stabilizer group is defined as follows:

Definition 5.1 $Stab(|\phi\rangle) = \{S : S|\phi\rangle = +1|\phi\rangle\}$

For $|\phi\rangle$ consists of n-qubits, the stabilizer group $Stab(|\phi\rangle)$ can be elegantly represented by its n generators, where each generator is a member of Pauli group, $\mathcal{P}_n$, defined in the Definition 5.2.

Definition 5.2

$$\mathcal{P}_n = \{P : P = s.P_i \otimes \ldots \otimes P_n, P_i \in \mathcal{P}_1, s \in \{\pm 1, \pm i\}\}$$

where $\mathcal{P}_1 = \{P : P = s.P', P' \in \{X, Y, Z, I\}, s \in \{\pm 1, \pm i\}\}$

An example of stabilizer state is two qubits entangled pair, known as **Bell** state, specified by its amplitudes in the Equation 5.1, and with its stabilizer group's generators in Equation 5.2.

$$|\phi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{5.1}$$

$$\begin{aligned} X \otimes X\,|\phi\rangle &= |\phi\rangle \\ Z \otimes Z\,|\phi\rangle &= |\phi\rangle \end{aligned} \tag{5.2}$$

The efficient representation of stabilizer states on one hand, and the structure of stabilizer group on the other hand, enables us to capture the evolution of stabilizer states under a limited number of quantum

operations, efficiently. More specifically, we have the following important theorem that shows the scope of stabilizer formalism.

Gottesman-Knill Theorem

Theorem 5.1 *(Gottesman-Knill, [84, p. 464]) Any quantum computation which consists of only the following components:*

1. *State preparation, Hadamard gates, Phase gates, Controlled-Not gates and Pauli gates.*
2. *Measurement gates.*
3. *Classical control conditions on the outcomes of measurements.*

can be efficiently simulated in polynomial time with polynomial space in the size of input (number of qubits), using a classical computer.

For equivalence checking, the ability of performing measurement and imposing classical conditions on their outcomes, play a crucial role in interpreting QIP protocols, specified in a high level language. In the Section 5.1.1, we explain how stabilizer states are specified and the effect of quantum operations on them are simulated. A number of useful algorithms which use of stabilizer formalism are presented in Section 5.1.2.

5.1.1 *Simulation Algorithm*

As a consequence of Theorem 5.1, several algorithms for simulation of stabilizer formalism have been proposed [4], [84, p. 463], [7]. In this work we adopt the algorithm in [4], where each stabilizer state

is associated with two subgroups of Pauli group namely, a stabilizer and a destabilizer group, where the latter is used for a more efficient measurement simulation. Applying quantum operations and measurements is thus manipulating this representation according to the simulation algorithm. In the following we explain the representation of the stabilizer states and their simulation algorithm.

Definition 5.3 *([4]) Let $S = \langle s_1, \ldots, s_n \rangle$ be a stabilizer group, as in Definition 5.1, with generators s_i, then the destabilizer group $D = \langle d_1, \ldots, d_n \rangle$ is a subgroup of $\mathcal{P}_n$ in which each generator d_i anti-commutes with s_j for $i = j$ and commutes with s_j when $i \neq j$.*

Each stabilizer state is represented by an array that keeps track of stabilizer generators, destabilizer generators and the overall phase of the state. A full stabilizer array, representing a stabilizer state, consists of stabilizer rows, destabilizer rows, phase column and a scratch row, i. e. an additional row for finding the outcomes of deterministic measurements. Note the stabilizer array representation is not unique, i. e. different stabilizer array may refer to the same stabilizer state.

Stabilizer Array

Definition 5.4 *([4]) Full stabilizer array consists of a pair $\mathcal{A} = (\mathcal{S}, \mathcal{D})$, where $\mathcal{S}$ is a stabilizer group and $\mathcal{D}$ is its destabilizer group, a phase column and a scratch row. $\mathcal{A}$ is represented in Figure 5.2. Each entry $\mathcal{A}_{i,j}$ in the stabilizer or destabilizer rows is thus a Pauli operator. Destabilizer rows are shown with entries $d_{i,j}$ and stabilizer rows with $s_{i,j}$. The column with entries of the form r_i^p, represents the phase, following the encoding in Figure 5.1. The last row in the array is the scratch row, with entries sc_i.*

$$+ \mapsto 0 \qquad i \mapsto 1$$
$$- \mapsto 2 \qquad -i \mapsto 3$$

Figure 5.1: Phase Encoding

$$\left[\begin{array}{ccc|c} d_{11} & \cdots & d_{1n} & r_1^d \\ \vdots & \ddots & \vdots & \vdots \\ d_{n1} & \cdots & d_{nn} & r_n^d \\ \hline s_{11} & \cdots & s_{1n} & r_1^s \\ \vdots & \ddots & \vdots & \vdots \\ s_{n1} & \cdots & s_{nn} & r_n^s \\ \hline sc_1 & \cdots & sc_n & sc_{(n+1)} \end{array}\right]$$

Figure 5.2: Stabilizer Array

The columns in the stabilizer array correspond to qubits of the stabilizer state. Simulation algorithm thus change the rows in the stabilizer array by applying three kinds of operations: single qubit unitary operation, CNOT operation between two columns and measurement (deterministic or random).

Remark 5.1 *In the previous version of QEC [9], we have used a different representation of stabilizer array in the binary matrix form, which had been used in [55] and [4].*

Operations within Pauli group follow the multiplication table of Figure 5.3, where the entries of this table are *phased Paulis*. In general, the set of operations on n-qubits that are allowed in stabilizer formalism are those in the *Clifford Group*, i. e. the group generated by the following set:

$$\{CNOT, P, H\} \tag{5.3}$$

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

Figure 5.3: Pauli Operator Multiplication Table

Let $\mathcal{C}_n$ denote Clifford Group of n-qubits state. Then $\mathcal{C}_n$ is the normalizer of Pauli group $\mathcal{P}_n$, meaning that for any $C \in \mathcal{C}_n$ and any Pauli operator P, we have:

$$C\,P\,C^{\dagger} = P' \tag{5.4}$$

Where $P' \in \mathcal{P}_n$. Now by applying a Clifford operation to the stabilizer array we compute instances of Equation 5.4, and additionally take care of phase of the target row.

Simulation of CNOT operation is similar, namely we look into the control column entries and update target column and the phase column. Finally, measurements in the *standard basis*, are simulated by considering two cases:

CASE 1: the outcome is deterministic and can be calculated using *row multiplication*, (according to Figure 5.3). After necessary row multiplications, the measurement outcome will be in the last entry of stabilizer array $sc_{(n+1)}$.

CASE 2: the outcome is random, so based on the chosen random outcome we apply necessary row multiplications.

Algorithm *STABILIZER-SIMULATION*($\mathcal{A}$, *Operation*, *Arguments*)

if *applying Pauli* σ_k *to qubit* j **then**
  for all *rows* $i \in \mathcal{A}$ **do**
    $\mathcal{A}_{i,j} = \sigma_k \, \mathcal{A}_{i,j} \, \sigma_k^\dagger$
    $r_i = (r_i + (1 - \alpha_{\text{phase}}))\%4$
  end for
end if
if *applying Hadamard to qubit* j **then**
  for all *rows* $i \in \mathcal{A}$ **do**
    $\mathcal{A}_{i,j} = HRotate(\mathcal{A}_{i,j})$
    $r_i = (r_i + (1 - \alpha_{\text{phase}}))\%4$
  end for
end if
if *applying Phase to qubit* j **then**
  for all *rows* $i \in \mathcal{A}$ **do**
    $\mathcal{A}_{i,j} = PRotate(\mathcal{A}_{i,j})$
    $r_i = (r_i + (1 - \alpha_{\text{phase}}))\%4$
  end for
end if
if *applying CNOT to qubit* j, r **then**
  for all *rows* $i \in \mathcal{A}$ **do**
    $\mathcal{A}_{i,j} = T_{\text{CNOT}}[\mathcal{A}_{i,j}, \mathcal{A}_{i,r}]$
    $r_i = (r_i + (1 - \alpha_{\text{phase}}))\%4$
  end for
end if

Figure 5.4: Stabilizer Simulation Algorithm

The complete algorithms for simulation of Clifford operators and measurements in the standard basis, is given in Figures 5.4 and 5.6.

In Simulation algorithm, after applying a Pauli σ_k on qubit j, we update each rows i and $\mathcal{A}_{i,j}$ according to Figure 5.3. This may cause appearance of the phase α_{phase} and that is used to update the overall phase column r_i. Here % denotes modulo function.

Hadamard and Phase operations on qubit j are done using *HRotate* and *PRotate*, based on Figure 5.5. These are called rotation because geometrically they act as rotations of quantum state's vectors.

Uniary rotations	X	Y	Z
I	X	Y	Z
PH	Z	X	Y
$HP^\dagger$	Y	Z	X
$PHP^\dagger$	$-X$	Z	Y
$HPPHP^\dagger$	Y	X	$-Z$
H	Z	$-Y$	X

Figure 5.5: Single qubit rotations

The two qubits CNOT operation acts on two qubits, represented as control and target columns , i. e. is done using CNOT table T_{CNOT} (see [12]). By applying a CNOT, both columns are updated, e. g. if we have X and Z in control and target columns, then after applying CNOT we will have $-Y$ and Y, respectively.

Figure 5.5 shows an interesting fact that all Clifford operators can be implemented as a sequence of Hadamard and Phase operations together with CNOT.

For Measurement, first we check whether it is random or not. Then if we encounter random measurement, we choose an outcome and apply a series of row multiplications and finally update the scratch row. In case we have deterministic measurement, after applying row multiplications we look into the scratch row's phase column and determine the outcome as follows:

$$Outcome = \begin{cases} 1 & if r_{sc} = 2 \\ 0 & otherwise \end{cases}$$

Algorithm *STABILIZER-MEASUREMENT*$(\mathcal{A}, q)$

boolean IsRandom;
for all *Stabilizer rows* $i \in \mathcal{A}$ **do**
  if $\mathcal{A}_{i,j} = \sigma_x$ *or* $\mathcal{A}_{i,j} = \sigma_y$ **then**
    return $IsRandom = true$
  else
    return $IsRandom = false$
  end if
end for
if $IsRandom = true$ **then**
  $r = K$
  while $\mathcal{A}_{r,q} = I$ *or* $\mathcal{A}_{r,q} = \sigma_z$, **do**
    for all *destabilizer rows* $d \in \mathcal{A}$ **do**
      multiply row d with row r
    end for
    for all *stabilizer rows* m *s.t.* $m \neq r$, $\mathcal{A}_{m,q} = \sigma_x$ *or* $\mathcal{A}_{m,q} = \sigma_y$ **do**
      multiply row m and r
    end for
    $\mathcal{A}_{sc,q} = \alpha_{\text{Outcome}}\sigma_z$
    $r = r + 1$;
  end while
end if
if $IsRandom = false$ **then**
  for all *destabilizer rows* $d \in \mathcal{A}$ **do**
    if $\mathcal{A}_{d,q} = \sigma_x$ *or* $\mathcal{A}_{d,q} = \sigma_y$ **then**
      multiply scratch row (*sc*) with d^{th} *stabilizer* row.
    end if
  end for
end if

Figure 5.6: Stabilizer Measurement

5.1.2 *Normal Forms Algorithms*

In the previous section we have seen that stabilizer states can be elegantly represented by Pauli generators. Unfortunately, this representation is not unique, e. g. row operations such as multiplication do not change the state, but change the representation. For various applications of stabilizer formalism, one needs to obtain unique normal forms of the stabilizer representations. Luckily, there are efficient algorithms to obtain such normal forms and they are thoroughly investigated by Audenaert and Plenio in [12]. Their motivation for obtaining normal forms was mainly concerned with expressing entanglement in stabilizer formalism, however, other useful algorithms are also introduced in [12], that are essential for the implementation of our tool, QEC.

In this section we briefly review algorithms related to the normal forms that have been used in the equivalence checking tool QEC. More details as well as proofs of correctness of these algorithms can be found in [12]. Row Reduced Echelon Form (RREF) of a stabilizer array can be obtained by applying a sequence of row operations on the stabilizer array. A stabilizer array is in *RREF*, if its stabilizer generators part can be recursively defined by the cases depicted in Figure 5.7.

RREF

Here *RREF* denotes a subarray that is in RREF. In the second case, P represents a Pauli operator and P_i's can be either identity or Pauli

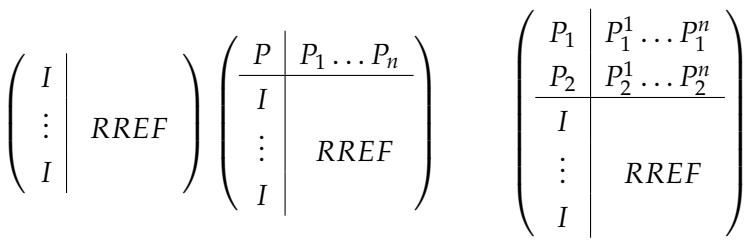


Figure 5.7: RREF Cases

operators. Likewise in the third case, P_1 and P_2 are Pauli operators, but P_i^j can be identity as well. Also P_1 and P_2 anti-commute.

The algorithm goes through these cases and in each case, applies a series of row multiplications and transposes until the entire stabilizer array has been put into RREF. For details of a the algorithm, please refer to [12].

Independence Checking

An important application of RREF algorithm is *independence checking* of a set of Pauli generators [12]. After applying RREF algorithm to a stabilizer array, the dependencies between stabilizer generators appear as rows containing only identity. The remaining stabilizer generators form an independent set of generators.

Partial Trace Algorithm

Another useful application of RREF algorithm is to obtain a partial trace of stabilizer states, using *PTRACE* algorithm [12]. This is crucial for QEC tool since we mostly deal with I/O protocols (see Definition 4.7), and therefore for computing the output of protocols we need to call PTRACE many times, depending on how many qubits have to be traced out. Partial trace algorithm changes quantum states and often results in *mixed* stabilizer states, so this fact has to be considered when it comes to further stabilizer simulation. However, in

Algorithm $PTRACE(\mathcal{A}, q)$
1: *RREF* ($\mathcal{A}_q$).
2: *In the column q, if there is no Pauli operator, do nothing.*
3: *In the column q, if there is only one kind of Pauli operator* σ*, remove the first row i where* $\mathcal{A}_{i,q} = \sigma$.
4: *In the column q, if there are two kinds of Pauli operators* σ_1 *and* σ_2 *such that they anti-commute, remove the first rows i and j, where* $\mathcal{A}_{i,q} = \sigma_1$ *and* $\mathcal{A}_{j,q} = \sigma_2$.
5: *Remove column q.*

Figure 5.8: PTRACE Algorithm

principle it is possible to defer partial trace to the end of execution of protocols, avoiding operations on mixed states. In particular QEC adopted the latter approach. The description of PTRACE algorithm is given in Figure 5.8, where $\mathcal{A}_q$ denotes column q as a subarray. Note that it is possible to apply PTRACE to a set of columns (corresponding to qubits). CNF

Another important normal form, discussed in [12] is *Column Normal Form (CNF)*, that in addition to row operations uses rotations in Figure 5.5. An stabilizer array after performing CNF would have the general form depicted in Figure 5.9. The CNF algorithm in each iteration counts the number of different Pauli operators in every column of the stabilizer array, then it applies a series of row operations and column operations as in Figure 5.9. For each column, there are three cases to consider: no Pauli operator in the column, there is only one kind of Pauli operators and finally, there are at least two different Pauli operators in the column. The details of each case of the algorithms can be found in [12]. Note that in contrast to RREF algorithm, CNF changes the stabilizer state, permanently.

$$\left(\begin{array}{cccc|ccc} X & I & \cdots & I & I & \cdots & I \\ I & X & \cdots & I & I & \cdots & I \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ I & I & \cdots & X & I & \cdots & I \\ \hline I & I & \cdots & I & I & \cdots & I \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ I & I & \cdots & I & I & \cdots & I \end{array}\right)$$

Figure 5.9: Column Normal Form (CNF)

$$\left[\begin{array}{c} \mathcal{A}_1 \\ \hline \mathcal{A}_2 \end{array}\right]$$

Figure 5.10: Combined Stabilizer Array

The most important application of CNF, as far as this work concerns, is in computing fidelity between two stabilizer states based on their stabilizer array representations. The concept of fidelity is discussed in Section 2.4 and is an information theoretic measure of similarity between two quantum states. The input of the algorithm for calculating fidelity [12] consists of two stabilizer arrays $\mathcal{A}_1$ and $\mathcal{A}_2$. The algorithm first brings the combined array in Figure 5.10 into CNF, by only considering the $\mathcal{A}_1$ because column operations will be automatically extended to $\mathcal{A}_2$. Then, for each column we will have three cases, similar to the CNF algorithm.

Fidelity Algorithm

For each case, the algorithm applies a series of column and row operations as well as maintaining values of certain variables for computing fidelity. After finishing iterations, based on these values, fidelity will be computed and the algorithm terminates. See [12] for more details.

5.2 EQUALITY TEST ALGORITHMS FOR STABILIZER STATES

This section introduces algorithms for testing equality of two stabilizer states. Our ability to test equality of stabilizer states stems from having classical efficient description in terms of stabilizer arrays. However, testing equality of two arbitrary quantum states (perfectly) by relying only on measurement, is not possible (see [84, p. 86]).

Considering non unique representation of stabilizer states by stabilizer arrays, equality tests require to incorporate normal form algorithms. In this regard, *Independence Checking (IC)* and *Inner Product (IP)* equality tests both rely on using RREF normal form algorithm.

IC Equality Test

The main idea of IC equality test is two prove equality of stabilizer states based on equality of their corresponding stabilizer groups. One can do the latter by looking into the dependence of their generators in the stabilizer array (we also take phases into account).

Let $|\phi_1\rangle$ and $|\phi_2\rangle$ be stabilizer states and $Stab(|\phi_1\rangle)$ and $Stab(|\phi_2\rangle)$ their stabilizer groups respectively. We have the following Lemma:

Lemma 5.1 *([9])* $|\phi_1\rangle = |\phi_2\rangle \Leftrightarrow Stab(|\phi_1\rangle) = Stab(|\phi_2\rangle)$

Proof *(of Lemma 5.1) To show* $|\phi_1\rangle = |\phi_2\rangle \Rightarrow Stab(|\phi_1\rangle) = Stab(|\phi_2\rangle)$ *suppose* $x \in Stab(|\phi_1\rangle)$ *then from* $x|\phi_1\rangle = x|\phi_2\rangle$, $|\phi_1\rangle = |\phi_2\rangle$ *we have* $x|\phi_2\rangle = |\phi_2\rangle$, *therefore* $x \in Stab(|\phi_2\rangle)$ *and* $Stab(|\phi_1\rangle) \subseteq Stab(|\phi_2\rangle)$. *Similarly we have* $Stab(|\phi_2\rangle) \subseteq Stab(|\phi_1\rangle)$.

For the other direction we need to show $Stab(|\phi_1\rangle) = Stab(|\phi_2\rangle) \Rightarrow |\phi_1\rangle = |\phi_2\rangle$.

Suppose in contrary, $|\phi_1\rangle \neq |\phi_2\rangle$*, then* $\exists v \in Stab(|\phi_1\rangle)$ *s.t.* $v|\phi_1\rangle \neq v|\phi_2\rangle$ *and that leads to* $Stab(|\phi_1\rangle) \neq Stab(|\phi_2\rangle)$*.*

The algorithm for IC equality test receives two input stabilizer arrays $\mathcal{A}_1$ and $\mathcal{A}_2$. Here the size of stabilizer arrays (number of qubits) is denoted by $N_{\mathcal{A}_i}$ and the number of identity rows, i. e. rows of the form $+I\ I\ \ldots\ I$, is shown by $Num_I^{\mathcal{A}_i}$. Sum of sets of rows is denoted by $\cup$. The algorithm first puts the input arrays into RREF, then computes the rank of stabilizer arrays ,r_i, by numbering the identity rows. At this point if they have distinct rank, then stabilizer states are not equal and the algorithm terminates, otherwise it continues by applying RREF to the sum of the stabilizer array rows and checking whether the rank, r, is matched with r_i's. If that is the case, the states are equal, otherwise, we conclude they are non-equal. Figure 5.11 gives the algorithm for checking IC equality.

Knowing that *RREF* algorithm has the complexity of $O(n^3)$, we conclude that the IC equality test algorithm terminates in polynomial time:

Proposition 5.1 *([9]) There is a polynomial time algorithm which decides for any stabilizer states* $|\phi\rangle$ *and* $|\psi\rangle$*, whether or not* $|\phi\rangle = |\psi\rangle$*.*

IP Equality Test

Another way of testing equality of two stabilizer states is computing their inner product, based on their stabilizer representation. This has been introduced in [4], and is given in the following definition.

Definition 5.5 *(Inner product of two stabilizer states [4]) Let* $|\phi\rangle$ *and* $|\psi\rangle$ *be two stabilizer states. For each set of generators* $G = \{G_1, \ldots, G_n\}$*, repre-*

Algorithm *IC-EQUALITY*$(\mathcal{A}_1, \mathcal{A}_2)$
1: $\mathcal{A}_1 = RREF(\mathcal{A}_1)$
2: $r_1 = N_{\mathcal{A}_1} - Num_I^{\mathcal{A}_1}$
3: $\mathcal{A}_2 = RREF(\mathcal{A}_2)$
4: $r_2 = N_{\mathcal{A}_2} - Num_I^{\mathcal{A}_2}$
5: **if** $(r_1 \neq r_2)$ **then**
6: **return** *False*
7: **end if**
8: $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$
9: $\mathcal{A} = RREF(\mathcal{A})$
10: $r_3 = N_{\mathcal{A}_1} + N_{\mathcal{A}_2} - Num_I^{\mathcal{A}}$
11: **if** $(r_3 = r_1 = r_2)$ **then**
12: **return** *True*
13: **else**
14: **return** *False*
15: **end if**

Figure 5.11: IC Equality Test Algorithm

senting $Stab(|\phi\rangle)$ *and* $H = \{H_1, \ldots, H_n\}$*, representing* $Stab(|\psi\rangle)$*, define* $D(G, H)$ *to be the number of generators such that* $G_i \neq H_i$*. Then, there are two cases:*

1. *If in all sets of generators such as G and H, there exists generators* G_i *and* H_i *in which they have equal Pauli operators with opposite sign, then* $\langle\phi|\psi\rangle = 0$*.*

2. *Otherwise for all such G and H we have* $\langle\phi|\psi\rangle = 2^{(-\min_{G,H} D(G,H))/2}$*.*

The algorithm for testing equality first has to bring the input states to normal forms in order to avoid iteration on all possible sets of generators. In this work we choose RREF, but in [4] a different normal form is used. The subroutine $ISORTHG(\mathcal{A}, \mathcal{B})$ checks whether the case 1 of Definition 5.5 applies, by looking into all stabilizer rows and

Algorithm *IP-EQUALITY*$(\mathcal{A}_1, \mathcal{A}_2)$
1: $\mathcal{A}_1 = RREF(\mathcal{A}_1)$
2: $\mathcal{A}_2 = RREF(\mathcal{A}_2)$
3: **if** $ISORTHG(\mathcal{A}_1, \mathcal{A}_2) = True$ **then**
4: **return** *False*
5: **else**
6: **if** $2^{(-\min_{G,H} D(G,H))/2} = 1$ *for G's in* $\mathcal{A}_1$ *and H's in* $\mathcal{A}_2$ **then**
7: **return** *True*
8: **else**
9: **return** *False*
10: **end if**
11: **end if**

Figure 5.12: IP Equality Test Algorithm

comparing their phases. The complete algorithm is presented in the Figure 5.12.

Remark 5.2 *In quantum information theory, equality of quantum states are usually considered in terms of* ***fidelity*** *between two states. Fidelity is defined in Definition 2.3 for general states and the algorithm for computing it, in the case of stabilizer states, is discussed in Section 5.1.2 and [12].*

From Proposition 2.1, we know that if two (mixed) stabilizer state are equal, then their fidelity is equal to 1 , otherwise they are not equal. Note that by using stabilizer states fidelity algorithm (Figure 5.2, [12]), we invoke CNF algorithm (Figure 5.9, [12]) that changes the states permanently, in contrast with the previous equality tests. Since computing fidelity of two stabilizer states uses CNF algorithm, the complexity at worst case is $O(n^3)$*, where n is the number of columns (qubits) in the stabilizer arrays.*

5.3 STABILIZER BASIS

In Section 5.4 and Section 5.5 we will describe decision procedures for testing *functionality* of quantum protocols (see Definition 4.10). However, for integrating these tests in QEC we need to have a workable basis set, that is consisting of only stabilizer states. In [51], Gay has introduced a stabilizer basis for the space of density matrices. We review this result in the following:

Theorem 5.2 *([51]) The space of density matrices for n-qubit states, considered as a $(2^n)^2$-dimensional real vector space, has a basis consisting of density matrices of n-qubit stabilizer states.*

Write the standard basis for n-qubit states as $\{|x\rangle \mid 0 \leqslant x < 2^n\}$, considering numbers to stand for their n-bit binary representations. We omit normalization factors when writing quantum states. With this notation, for $n \geqslant 1$ let $\mathsf{GHZ}_n = |0\rangle + |2^n - 1\rangle$ and $\mathsf{iGHZ}_n = |0\rangle + i|2^n - 1\rangle$, as n-qubit states.

Lemma 5.2 *([51]) For all $n \geqslant 1$, GHZ_n and iGHZ_n are stabilizer states.*

Proof *By induction on n. For the base case ($n = 1$), we have that $|0\rangle + |1\rangle$ and $|0\rangle + i|1\rangle$ are stabilizer states, by applying H and then P to $|0\rangle$.*

For the inductive case, GHZ_n and iGHZ_n are obtained from $\mathsf{GHZ}_{n-1} \otimes |0\rangle$ and $\mathsf{iGHZ}_{n-1} \otimes |0\rangle$, respectively, by applying CNot to the two rightmost qubits.

Lemma 5.3 *([51]) If $n \geqslant 1$ and $0 \leqslant x, y < 2^n$ with $x \neq y$ then $|x\rangle + |y\rangle$ and $|x\rangle + i|y\rangle$ are stabilizer states.*

Proof *([51]) By induction on n. For the base case ($n = 1$), the closure properties imply that $|0\rangle + |1\rangle$, $|0\rangle + i|1\rangle$ and $|1\rangle + i|0\rangle$ (equivalent to $|0\rangle - i|1\rangle$ by scalar multiplication) are stabilizer states.*

For the inductive case, consider the binary representations of x and y. If there is a bit position in which x and y have the same value b, then $|x\rangle + |y\rangle$ is the tensor product of $|b\rangle$ with an $(n-1)$-qubit state of the form $|x'\rangle + |y'\rangle$, where $x' \neq y'$. By the induction hypothesis, $|x'\rangle + |y'\rangle$ is a stabilizer state, and the conclusion follows from the closure properties. Similarly for $|x\rangle + i|y\rangle$.

Otherwise, the binary representations of x and y are complementary bit patterns. In this case, $|x\rangle + |y\rangle$ can be obtained from GHZ_n by applying X to certain qubits. The conclusion follows from Lemma 5.2 and the closure properties. The same argument applies to $|x\rangle + i|y\rangle$, using iGHZ_n.

Proof (of Theorem 5.2 [51]) *This is the space of Hermitian matrices and its obvious basis is the union of*

$$\{|x\rangle\langle x| \mid 0 \leqslant x < 2^n\} \tag{5.5}$$

$$\{|x\rangle\langle y| + |y\rangle\langle x| \mid 0 \leqslant x < y < 2^n\} \tag{5.6}$$

$$\{-i|x\rangle\langle y| + i|y\rangle\langle x| \mid 0 \leqslant x < y < 2^n\}. \tag{5.7}$$

Algorithm *BASIS(n)*

1: $\mathcal{B} = \emptyset$
2: *Add standard basis by applying* σ_X *to the qubits of the state* $|0_1 0_2 \dots 0_n\rangle$ *to* $\mathcal{B}$.
3: *Add GHZ and iGHZ states to* $\mathcal{B}$*: by applying a H or HP and then a sequence of CNOTs with the initial state* $|00\dots0\rangle$.
4: *Consider logical qubits:* $\left|\hat{b_1}\hat{b_2}\dots\hat{b_n}\right\rangle$. *Construct GHZ and iGHZ states of step (3) on the syndrome qubits, i. e. positions* $\hat{b_i}$ *with different binary value, and add them to* $\mathcal{B}$. *Also add the states such that* σ_X *is applied to the* $\hat{b_i}$*s with the same value.*
5: **return** $\mathcal{B}$

Figure 5.13: Algorithm for generating Stabilizer Basis

Now consider the union of

$$\{|x\rangle\langle x| \mid 0 \leqslant x < 2^n\} \tag{5.8}$$

$$\{(|x\rangle + |y\rangle)(\langle x| + \langle y|) \mid 0 \leqslant x < y < 2^n\} \tag{5.9}$$

$$\{(|x\rangle + i|y\rangle)(\langle x| - i\langle y|) \mid 0 \leqslant x < y < 2^n\}. \tag{5.10}$$

This is also a set of $(2^n)^2$ *states, and it spans the space because we can obtain states of forms (5.6) and (5.7) by subtracting states of form (5.8) from those of forms (5.9) and (5.10). Therefore it is a basis, and by Lemma 5.3 it consists of stabilizer states.*

We used the construction in the proof of Theorem 5.2 and present an algorithm in the Figure 5.13, for the generation of stabilizer basis.

5.4 EQUIVALENCE CHECKING OF SEQUENTIAL PROTOCOLS

In this section we explain the core idea of verifying sequential functional protocols using equivalence checking, as we previously investigated in [9]. We use the superoperator semantics, in [94] (see Section 3.1), to show that the implementation of a functional quantum protocol in QPL is equivalent to its specification. In order to automate this procedure, we take advantage of linearity of supeoperators (see Theorem 2.2) and iterate equivalence checking, for each *basis* vectors of the input space in Section 5.3, avoiding continuum of input density matrices.

Our first step is to check the *functionality* of given implementation and specification protocols. Assume we have given a sequential protocol, P, e. g. a QPL program, with the input space V. Suppose P has M branching points, e. g. there are M random measurements in the program, then we assign to each branch i, a superoperator $\mathcal{E}_i$ as in [9]. Also From [94], we know that the semantics of P is described by a unique superoperator, Δ. Thus to show that P is functional it suffices to prove that:

$$\forall\, 1 \leq i \leq 2^M,\ \mathcal{E}_i \backsimeq \Delta \tag{5.11}$$

We can write Equation 5.11 with density matrices as following:

$$\forall\, 1 \leq i \leq 2^M,\ \forall\, \rho \in V,\ \mathcal{E}_i(\rho) = \Delta(\rho) \tag{5.12}$$

However, the input space V in the above equation can be infinite, making automating the functionality test infeasible. To solve this problem, we use the fact that V is a linear space and also superoperators are linear operators. So it suffices to check validity of Equation 5.12 for each vectors in a basis of the space V. Let $\mathcal{B}_V$ be such basis, then Equation 5.12 can be rewritten as:

$$\forall\, 1 \leq i \leq 2^M,\ \forall\, \rho \in \mathcal{B}_V,\ \mathcal{E}_i(\rho) = \Delta(\rho) \tag{5.13}$$

Proposition 5.2 *([9]) Checking functionality of any sequential protocol, such as aforementioned P (specified in QPL or CCS^q), is decidable.*

Proof *Let the protocol P receive n qubits as input, then the dimension of V is 2^{2n} and therefore we have $|\mathcal{B}_V| = 2^{2n}$. Following the approach of [9], starting from a pure basis state, $|b\rangle \in \mathcal{B}_V$, and for each branch i, $\mathcal{E}_i(|b\rangle) = \left|\phi_i^b\right\rangle$ with probability 2^{-M}, where M is the number of branching points (e. g. random measurements). Equation 5.13 is then proved by checking that for every basis state, as input, all branches end up in an equal state:*

$$\forall |b\rangle \in \mathcal{B}_V\, \forall 1 \leq i, j \leq 2^M \left|\phi_i^b\right\rangle =_{EqT} \left|\phi_j^b\right\rangle \tag{5.14}$$

Here $=_{EqT}$ denotes an equality test between pure quantum states such as those explained in Section 5.2.

Now for given QPL programs P_i, $i \in \{1, 2\}$, representing the specification and implementation of a sequential protocol, we want to check

their equivalence $P_1 \cong P_2$. Let $S_i(\rho)^{(j)}$ denote the output state for the branch j of program P_i, for input state ρ. The equivalence checker first use the above procedure for functionality test then examines the equivalence of two programs, *for all states in the stabilizer basis constructed by the algorithm in Figure 5.13, which is denoted by* $\mathcal{B}_V$. In other words it computes the following (informal) expression:

$$\begin{aligned} & \forall \rho \in \mathcal{B}_V.\ \forall j,k.\ S_1(\rho)^{(j)} = S_1(\rho)^{(k)} \\ \wedge \quad & \forall \rho \in \mathcal{B}_V.\ \forall j,k.\ S_2(\rho)^{(j)} = S_2(\rho)^{(k)} \\ \wedge \quad & \forall \rho \in \mathcal{B}_V.\ S_1(\rho) = S_2(\rho) \end{aligned}$$

Let $paths(P, s)$ denote the set of possible paths, indexed by integers from 1 upwards, when executing program P on input state s. Let $StabSim(P, s, j)$ denote the final state produced by the stabilizer simulation algorithm in Section 5.1, starting with input state s and executing path j of program P. Let $EQ_S(v, w)$ be the equality tests algorithm in Section 5.2. Then the above procedure corresponds to the algorithm in Figure 5.14.

Remark 5.3 *The overall complexity of the above algorithm is* $O(2^{2n} poly(m+n))$*, where* n *is the number of input qubits and* m *is the number of qubits inside the programs (i.e those created by* ***newqbit****).*

5.5 EQUIVALENCE CHECKING OF CONCURRENT PROTOCOLS

In the classical theory of computation several equivalence checking methods have been introduced for concurrent systems namely: bisim-

1: **for all** $v \in \mathcal{B}_V$ **do**
2: **for all** $i \in \{1,2\}$ **do**
3: $\left|\phi_i^v\right\rangle = StabSim(P_i, v, 1)$
4: **for all** $j \in paths(P_i, v) - \{1\}$ **do**
5: **if** $\neg EQ_S(StabSim(P_i, v, j), \left|\phi_i^v\right\rangle)$ **then**
6: **return** P_i non-functional
7: **end if**
8: **end for**
9: **end for**
10: **if** $\neg EQ_S(|\phi_1^v\rangle, |\phi_2^v\rangle)$ **then**
11: **return** $P_1 \ncong P_2$
12: **end if**
13: **end for**
14: **return** $P_1 \cong P_2$

Figure 5.14: Algorithm for checking equivalence of QPL programs.

ulation based, automata based, game semantics and trace semantics. All of these methods are coupled with a notion of *state*, that is classical. However in QIP, systems are defined with quantum states and therefore the above techniques are not applicable directly.

In this work we use superoperator semantics, defined in the previous chapters, as a suitable abstraction of concurrent functional quantum protocols. Similar to the case of sequential protocols, we restrict ourself to the *functional concurrent protocols* (see Definition 4.10) that are specified in CCS^q. In the following we show that in principle, functionality test of concurrent I/O protocols is decidable.

To show functionality of concurrent I/O protocols, all possible interleavings have to be considered. Then we use Proposition 4.1 to obtain an associated superoperator to each interleaving. Let $\mathcal{I}$ be the set of all interleavings of a given concurrent I/O protocol and assume

$\mathcal{F}_i$ is the associated superoperator to the interleaving i. To check functionality of such a protocol, we need to show that:

$$\forall\, 1 \leq i \leq |\mathcal{I}|,\ \mathcal{F}_i \approx \Delta \tag{5.15}$$

where Δ is a unique superoperator. Subsequently, we get the following equation:

$$\forall\, 1 \leq i \leq |\mathcal{I}|,\ \forall\, \rho \in V,\ \mathcal{F}_i(\rho) = \Delta(\rho) \tag{5.16}$$

where V is the input space. Similar to the sequential case, we use the basis set $\mathcal{B}_V$ to check the functionality, feasibly:

$$\forall\, 1 \leq i \leq |\mathcal{I}|,\ \forall\, \rho \in \mathcal{B}_V,\ \mathcal{F}_i(\rho) = \Delta(\rho) \tag{5.17}$$

Now each interleaving in Equation 5.17 is separately checked for functionality as a sequential case, according to Proposition 5.2.

Proposition 5.3 *([11]) Checking the functionality of any concurrent I/O protocol in CCS^q is decidable.*

Proof *Let V be the input state of n qubits density matrices and $\mathcal{B}_V$ be the basis set. Assume $\mathcal{I}$ denote the interleavings set. Starting from the pure basis state $|b\rangle$, for each interleaving i, we calculate the effect of its superoperator on the basis b, $\mathcal{F}_i(|b\rangle)$. If i is branching, then first we run decision procedure of the Proposition 5.2. By iteration on basis vectors we show:*

$$\forall\, 1 \leq i, j \leq |\mathcal{I}|,\ \forall\, |b\rangle \in \mathcal{B}_V,\ \mathcal{F}_i(|b\rangle) =_{EqT} \mathcal{F}_j(|b\rangle)$$

The following algorithm describes how to check functionality of a given concurrent I/O protocol by first *scheduling*, then interpreting it in the stabilizer formalism, using stabilizer simulation algorithm (in Section 5.1.1), and finally testing equality of the reached final states, (in Section 5.2).

Remark 5.4 *In QEC, we represent density matrices (mixed states) implicitly. This is done by interpreting protocols with* pure *stabilizer states on different runs of the protocol's model. However it may possible to work with mixed stabilizer states directly [4].*

for a concurrent program P, let $\mathcal{I}(P, v)$ denote all possible interleavings of the program with initial state v in the stabilizer basis $\mathcal{B}$, produced by a scheduler and indexed by positive integers. Let I_i denote the ith interleaving and suppose $StabSim^*(P, v, i)$ shows the final state given by stabilizer simulation algorithm in [4] applied to I_i, on initial basis state v. Finally, let $EQ_S(v, w)$ be one of the the equality test algorithms in Section 5.2, then Figure 5.15 shows the equivalence checking algorithm for two concurrent programs P_1 and P_2, and establishes the following result.

Proposition 5.4 *Given two functional concurrent quantum protocols, which only use operations in the stabilizer formalism, one can decide whether they are equivalent with respect to their superoperator semantics on every possible input.*

The cost of running the above equivalence checking algorithm increases exponentially in the size of input qubits, because of iteration

1: **for all** $v \in \mathcal{B}$ **do**
2: **for all** $i \in \{1,2\}$ **do**
3: $|\phi_i^v\rangle = StabSim^*(P_i, v, 1)$
4: **for all** $j \in \mathcal{I}(P_i, v) - \{1\}$ **do**
5: **if** $\neg EQ_S(StabSim^*(P_i, v, j), |\phi_i^v\rangle)$ **then**
6: **return** P_i non-functional
7: **end if**
8: **end for**
9: **end for**
10: **if** $\neg EQ_S(|\phi_1^v\rangle, |\phi_2^v\rangle)$ **then**
11: **return** $P_1 \ncong P_2$
12: **end if**
13: **end for**
14: **return** $P_1 \cong P_2$

Figure 5.15: Algorithm for checking equivalence of concurrent quantum protocols.

over all basis states, and the number of concurrent processes, because of scheduling to determine interleavings. The following proposition gives the computational complexity of our equivalence checking algorithm. Note that in classical computing, the equivalence checking problem or *implementation verification* of concurrent systems (where only the containment problem is considered, not the simulation problem), is *PSPACE-complete* (see [63] for details).

Proposition 5.5 *Checking equivalence of concurrent quantum protocols has overall (time) complexity of* $O(N\,2^{2n}poly(m+n))$*, where n is the number of input qubits (basis size), m is the number of qubits inside a program (*i.e *those created by* ***newqbit****) and N is the number of interleavings of processes (where* $N = \frac{(\sum_i^M n_i)!}{\prod_i^M (n_i!)}$ *for M processes each having* n_i *atomic instructions) .*

The analysis in Proposition 5.5 is based on three phases of our algoritm namely: scheduling (where N comes from), basis generation (fac-

tor 2^{2n}) and stabilizer simulation and equality test (factor $poly(m+n)$).

Remark 5.5 *The complexity of stabilizer simulation algorithm is $\oplus L$ (parity L) This is the class of all problems that are solvable by a non-deterministic logarithmic-space Turing machine, that accepts if and only if the total number of accepting paths is odd (see [4] for more details).*

Approximate Equivalence Checking

So far we have described an *exact* equivalence checking method. That is, we prove that for a given specification P_S and implementation P_I, with their associated superoperators $\mathcal{E}_S$ and $\mathcal{E}_I$, respectively:

$$P_S \cong P_I \Longleftrightarrow \mathcal{E}_S \backsimeq \mathcal{E}_I \tag{5.18}$$

The questions that may arise here are what happens when we get non-equivalent implementation and specification in Equation 5.18? Due to the complicated physical implementation of QIP systems, we may have hidden noises/faults from the implementation protocol, so is there any way of *approximating* equivalence checking with respect to superoperator semantics? Assuming we stay in stabilizer formalism, one way of approximating our equivalence checking can be described as follows:

Let ρ_1 and ρ_2 be two density operators. A distance measure for density operators is defined by Bures [26], and it is in Definition 5.6.

Definition 5.6 *Let $F(\rho_1, \rho_2)$ denote fidelity as in Definition 2.3. Then a Bures distance between ρ_1 and ρ_2 is defined by:*

$$D_B(\rho_1, \rho_2) = 2\sqrt{1 - F(\rho_1, \rho_2)}$$

Now we define ε-equivalent protocols as follows:

Definition 5.7 *Suppose P_S and P_I, corresponding to specification implementation of a protocol, are:*

1. *Functional.*

2. *Defined in the stabilizer formalism.*

Let $\mathcal{B} = \{b_1 \dots b_n\}$ be the stabilizer basis, P_S and P_I are defined to be ε-equivalent if:

$$\max_{b \in \mathcal{B}} D_B(\mathcal{E}_S(b), \mathcal{E}_I(b)) = \varepsilon$$

The main drawback of Definition 5.7 is that it depends on the basis $\mathcal{B}$, so by choosing a different basis, a smaller approximation factor ε may be achieved. Nevertheless, our approximation scheme is defined in the stabilizer formalism and therefore is computationally efficient.

5.6 CONCLUDING REMARKS

In this chapter we have described how functionality and equivalence of concurrent quantum protocols can be checked by using and developing techniques in the stabilizer formalism. We have illustrated that for a certain class of quantum protocols with arbitrary input, one can check the equivalence of specifications and implementations, efficiently. This idea has been implemented in the tool QEC and applied

to various well known QIP protocols that will be presented in the Chapter 6

The equivalence checking algorithms are based on the use of stabilizer basis for the state of density matrices and the linearity of superoperators. This makes our algorithms computational complexity exponential in the number of input qubits and number of concurrent processes (see Proposition 5.5).

Equivalence checking can also be used to verify the synthesis of large stabilizer circuits, and we know our method is scalable for the verification of stabilizer circuits with bounded number of input/output, but with a large number of internal gates (like the circuits used in quantum error correction [27]). Now the question is how equivalence checking can be applied to the stabilizer circuits with a large number of input qubits? It turns out that with some preprocessing, one can check equivalence of such circuits: Gay has proposed [1] using the notion of *map-state duality*, that is a bijection between linear maps of the form $V_1 \longrightarrow V_2$ and the elements of their tensor product space $V_1 \otimes V_2$, assuming V_is are finite dimensional. It is shown [2] that for any stabilizer circuits with n inputs and m internal gates, its dual, which is an stabilizer state, can be constructed in $O(n^2 + mn)$. So for a given specification and implementation circuits one can obtain the dual stabilizer states and apply one of the quality tests, discussed in this chapter.

1 private communication
2 Unpublished work by S J Gay

6

QUANTUM EQUIVALENCE CHECKING IN PRACTICE

In the previous chapters we have studied the equivalence checking problem for the specification and implementation of QIP protocols. It turns out that many interesting and well known QIP protocols, those that are definable in the stabilizer formalism, can be verified by our equivalence checking method. This has led us to implement the equivalence checker tool, QEC. In the previous version of QEC [9], sequential quantum protocols were specified in QPL (see Figure 3.1) whereas in the concurrent version they are specified in CCS^q and applied to concurrent protocols. In particular QEC implements the core equivalence algorithms, studied in Chapter 5 on given specification and implementation of protocols, and checks whether they are functional and equivalent.

In this chapter we explain details of the QEC implementation and give a range of case studies. Each case study is implemented with different models, for example we specify protocols sequentially with QPL and in the concurrent CCS^q model, and then compare the results of verification.

In the following, first we explain the structure and implementation of the QEC tool. Then we present cases studies, their underlying structure, specifications and implementations of them. Finally in Section 6.2, experimental results and comparison of different models are discussed.

6.1 IMPLEMENTATION DETAILS

The Quantum Equivalence Checker is a tool that implements several algorithms around the stabilizer formalism, as elaborated in the previous chapters. This tool, similar to QMC, has been developed in the Java programming language. As a part of QEC, several stabilizer algorithms have been implemented in Java and used.

The development of QEC consists of 30,327 Source Lines of Code (SLOC). Comparing to QMC which contains 89,275 SLOC, our tool has smaller size while is significantly more efficient in the verification tasks and has a clearer structure because of the following design decisions:

1. Separating Scheduling of concurrent protocols from it semantics in contrast to QMC. This gave us a better understanding of resources needed to execute a concurrent quantum protocol.

2. In QEC Stabilizer arrays are directly represented and implemented rather than binary representation in [4], since we needed to manipulate stabilizer arrays a lot by applying different algo-

rithms on them. This gave us better picture of QEC operations on quantum states without great impact on efficiency.

The tool consists of the following three major components:

1. *Preprocessing*
2. *Model Construction*
3. *Verification*

The first component, preprocessing, protocol's implementation and specification, and builds the *abstract syntax tree* for each of them. It also produces the necessary stabilizer basis for later stages of verification. The second component constructs the model of the input protocols, by scheduling and then interpreting them in the stabilizer formalism. In order to check the complete model, all interleavings and branches are explored by QEC. Finally, the verification unit carries out functionality test and equivalence checking on the given protocols. The general architecture of QEC is depicted in the Figure 6.1.

QEC can be downloaded from [8]. It automatically constructs the model, for all basis states, and verifies the given protocols by the equivalence checking algorithm.

6.1.1 *Preprocessing*

Preprocessing in QEC is done in two phases. In the first phase, the input protocols are *parsed* and the Abstract Syntax Tree (AST) is pro-

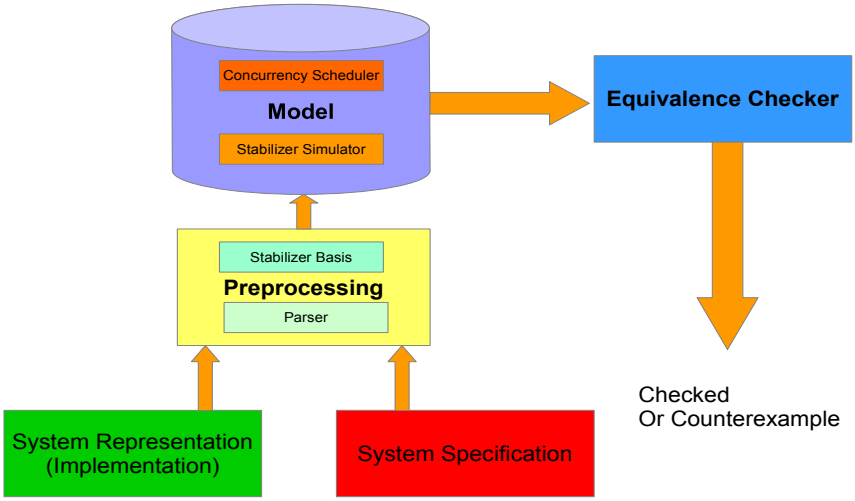


Figure 6.1: Structure of QEC

duced, using *SableCC* [49] tool. This is a general purpose compiler to compiler software that automatically generates parser and AST based on a given grammar. It also checks the given protocols against syntax errors. The back end of SableCC is then usable by Java Runtime Environment (JRE).

Another important part of preprocessing is the generation of stabilizer basis. To this end a *visitor* [49] to the automatically generated AST is implemented, based on the algorithm in Figure 5.13.

Remark 6.1 *In the previous version of QEC [9] we have used the simple QPL language and its grammar, i.e. described by the syntax in Figure 3.1 without recursions and loops, for the specification of quantum protocols. However, in concurrent QEC, the grammar is based on CCS^q, and in addition to the concurrent structure, it has other features such as matching expressions and explicit measurement expressions.*

6.1.2 *Model Construction*

The model construction unit of QEC consists of a *scheduler* and an *interpreter*. In the sequential QEC, the scheduler produces *branches* by implementing a visitor to the QPL's SableCC generated AST. This visitor constructs a binary tree with its branches corresponding to the quantum measurements with random outcomes, generated by the following command:

$$\textit{masure } q \textit{ then } \ldots \textit{else } \ldots \textit{end}$$

Each path from root (corresponds to the input point) to the leaves (correspond to the output point) in this binary tree, is encoded by a sequence of 0's and 1's, that guides us which measurement outcome has to be chosen.

Scheduling of concurrent protocols are more complicated since we need to consider communications and parallel compositions to produce interleavings. The core idea of QEC's concurrency scheduler is to *separate* the scheduling from the *semantics* of the input language. In that sense, our scheduling technique is very different from QMC, where scheduling was based on executability predicates, relating it to the operational semantics of QMCLang. The advantages of our scheduling method are reflected in the implementation of a more clear, faster and bug free scheduler.

QEC scheduler uses a Java interface, called *Schedulable* with three methods in it : $Options()$, $Select()$ and $Reset()$. The scheduler uses this

interface and determines for each process a list of available options in the schedule. Options for each process could be receiving, sending or internal (τ) actions. In CCS^q, Nil is a process without any options. Selecting an option triggers $Select()$ to pass an available action to the stabilizer simulator. Finally, $Reset()$ resets the scheduler to a previous scheduling point.

In the case of parallel composition of two process, the scheduler options list would be the sum of each process's options set. Therefore at each step of protocol execution, a list of scheduling options is maintained by the scheduler. This vector of options lists corresponds to a scheduling tree, where at each level of the tree we have a list of available options. QEC scheduler applies a depth-first search on this tree to explore all possible scheduling paths. Note that QEC does not construct the whole scheduling tree at once, but instead extracts possible schedules from AST. When one scheduling path is fully explored and interpretation of it is done, it will be removed. The above procedure will be iterated until the whole scheduling tree is explored and depth-first search is terminated.

The second part of model construction concerns with interpreting quantum protocols in the stabilizer formalism. To this end the stabilizer simulation algorithm discussed in Section 5.1.1 is implemented in Java.

Since we deal with I/O protocols, in addition to the stabilizer simulation, the partial trace algorithm (Section 5.1.2) is used to interpret the output command that terminates the execution of the protocols.

Once the model is constructed for a given input basis state, a vector of stabilizer arrays, corresponding to the reachable final states, will be stored.

6.1.3 *Verification*

The first phase of verification is to test whether an input protocol is functional. This is done using Proposition 5.2 for the sequential QPL protocols and is implemented in [9], whereas in the case of concurrent CCS^q protocols, Proposition 5.3 is implemented.

In the implementations of QEC, functionality of protocols are checked by repeating $IC - EQUALITY$ test (Section 5.2) on the vectors of stabilizer arrays (constructed models).

Once the functionality of specification and implementation protocols is checked, QEC enters the second phase where the equality test will be applied to a pair of stabilizer states, one from specification's constructed model and one from implementation's model. This process will be repeated for all basis states, where the equivalence checking algorithms (see Chapter 5) terminate. If no instance of non-equality is found, QEC reports equivalence is checked otherwise it shows on which basis , the specification and implementation are not equivalent.

QEC is a fully automatic tool such it can be used easily in the terminal as shown in Figure 6.2, where specification and implemen-

```
> check Specification.ccs Implementation.ccs
```

Figure 6.2: QEC usage

```
> Basis generated with N states.
> Equivalence is Checked on M runs of Specification and
  L runs of Implementations.
> Final evaluation is -.
> Verification time: - ms.
```

Figure 6.3: QEC output

tation are stored in two separate files, *Specification.ccs* and *Implementation.ccs*, respectively (see [8]). The output of tool represents how many basis is generated for the equivalence checking, the number of runs for specification and implementation protocols, final result of equivalence checking considering all runs and finally the verification time is reported. Figure 6.3 shows a how the output of the tool looks like.

6.2 CASE STUDIES

In the following sections we apply QEC tool to verify a range of QIP protocols. These examples are chosen from different areas of QIP such as quantum communication, quantum error correction, quantum cryptography and quantum fault tolerant computations.

For each example we explain the specification and implementation in different modelling languages such as quantum circuit, QPL and CCS^q.

The general structure of concurrent models is presented by diagrammatic representation, where boxes represent processes and arrows denote the interactions between processes (dotted arrows for classical communications and line arrows for quantum communications). In these diagrams, the input of protocols located in the process with $\bullet$ token and the output process is distinguished with the token $\star$.

6.2.1 *Communication and Cryptographic Protocols*

In this section we present three quantum protocols: Teleportation, Dense Coding and Quantum Secret Sharing, along with the details of their specification and implementation.

6.2.1.1 *Teleportation*

Teleportation [17] is a quantum communication protocol designed to use only local quantum operations and classical communications (LOCC). It is an important primitive in QIP and many computational schemes and models depend on it. Teleporting a given quantum state from Alice to Bob in this protocol can be achieved by only using entanglement, two instances of classical communications, local quantum operations and measurements.

Teleportation in Quantum Circuit

The implementation of Teleportation is usually described by a quantum circuit, shown in Figure 2.3, where three lines (from top) correspond to Alice's qubit, Alice's share of entangled pair and Bob's

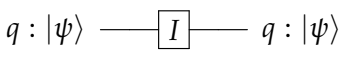


Figure 6.4: Teleportation Specification in Quantum Circuits

```
program Teleportation_Specification
input  q0:qbit
output q0:qbit
```

Figure 6.5: Teleportation Specification in QPL

qubit, respectively. The first two columns of the circuit, applied to the second and third qubits represent preparation of entangled pair. The third and forth columns show Alice's operations. Alice's measurements are shown by the fifth column and finally, Bob's operations depending on the outcomes of Alice's measurements, are shown in the fifth and sixth columns.

The specification of Teleportation can be described by the circuit in Figure 6.4, where I denotes identity gate applied to q with the state $|\psi\rangle$ and output q with the same state.

Teleportation in QPL

The sequential QEC [9] uses a programming interface to describe Teleportation i. e. the implementation is described by a QPL program, as it is shown in Figure 3.2. The spefication of Teleportation in the sequential equivalence checker is a *program* which has the effect of identity, i. e. given a qubit q_0, the specification outputs the same qubit without any operation on it, as is illustrated in Figure 6.5.

Teleportation in CCS^q

The concurrent model of Teleportation is specified in the language CCS^q as follows: there are three processes *EPR*, *Alice* and *Bob* that interact according to Figure 6.6. Each process runs in parallel to others, while the input occurs in Alice process and Bob ends up with

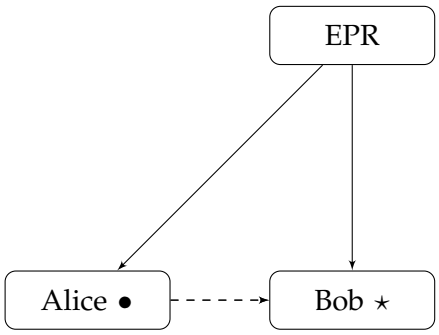


Figure 6.6: Diagrammatic Teleportation

```
//Specification of concurrent Teleportation:

input x.output x.nil
```

Figure 6.7: Teleportation Specification in CCS^q

the output of the protocol. The complete model Teleportation's implementation is shown in Figure 4.2.

In the concurrent model we can model the physical separation of Alice and Bob, a feature that can not be captured within the quantum circuit model or QPL programs. Also the communications between EPR process, Alice and Bob are made explicit. The specification of Teleportation in this case is a protocol consisting of a process, passing the input to the output without any prefix (action). This is shown in Figure 6.7.

As an interesting example to show how the design of concurrent quantum protocols can be non intuitive, suppose in the Alice process of Figure 4.2, quantum measurements and sending outcomes are run concurrently, e. g. we have the following term, instead:

$$(m := measure\, x \,.\, b!m \mid n := measure\, y \,.\, b!n)$$

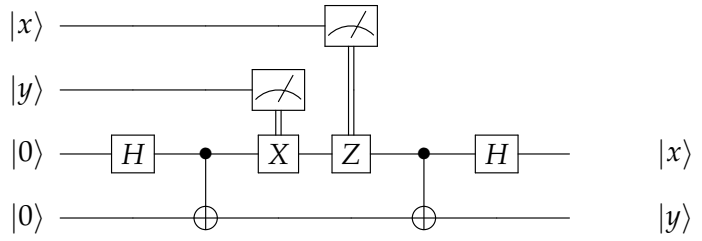


Figure 6.8: Dense Coding Implementation in Quantum Circuits

then specification and implementation become non-equivalent, because Bob's actions are determined by both m and n, and therefore the measurement and sending the outcomes can not be interleaved.

6.2.1.2 *Dense Coding*

Dense Coding [16] is another quantum communication protocol which takes advantage of entanglement. In this protocol, Alice communicates two pieces of classical information only by sending one qubit to Bob and using a pair of entangled qubits. The fact that Dense Coding needs an entangled pair means it does not offer a more efficient way of classical communications, nevertheless it illustrates how peculiar QIP could be.

Dense Coding in Quantum Circuit

One way to implement Dense Coding is to encode two classical bits of input, say x and y onto two qubits, resulting in the state $|x\rangle|y\rangle$, followed by the operations of Alice and Bob and get the output $|x\rangle|y\rangle$. Figure 6.8 shows Dense Coding protocol in the language of quantum circuits.

The circuit in Figure 6.8, in fact, computes a classical binary function with two inputs and outputs encoded on two qubits i. e. a binary

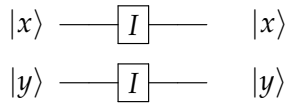


Figure 6.9: Dense Coding Specification in Quantum Circuits

function on the set $\{00, 10, 01, 11\}$. For this reason, in both sequential and QEC, Dense Coding is analysed on the mentioned set (standard basis) and not the complete stabilizer basis in Chapter 5. The Specification of Dense Coding is shown in Figure 6.9.

Dense Coding in QPL

It is shown in Figure 6.10 how Dense Coding is implemented in the sequential QEC. The specification in this case is a QPL protocol with two qubits input and output q_0, q_1, representing encoded classical bits, as illustrated in Figure 6.9.

Dense Coding in CCS^q

Finally, we have implemented Dense Coding in the concurrent QEC. Similar to Teleportation, there are three processes of *EPR, Alice* and *Bob*, where Alice holds the input and Bob has the output of the protocol. The general structure of concurrent Dense Coding is presented in Figure 6.12, and the implementation is described in Figure 6.13.

The specification of concurrent Dense Coding protocol consists of a process which passes two qubits input to the output and is shown in Figure 6.14.

A different way of implementing Dense Coding is to define directly inputs with classical bits. In that case we replace Alice measurements in Figure 6.13 with the following classical conditions:

$$if\ x\ then\ Z(a)\ .\ if\ y\ then\ X(a)$$

```
program DenseCoding_Specification
input  q0,q1:qbit
output q0,q1:qbit
```

```
program DenseCoding_Implementation

input q0,q1:qbit

//EPR pair(q3 is in possesion of Bob)

newqbit q2;

newqbit q3;

q2*=H;

q2q3*=CNot;

//Alice operations
measure q0 then q2*=Z else q0*=I end;

measure q1 then q2*=X else q1*=I end;

//Bob operations
q2q3*=CNot;

q2*=H

output q0,q1:qbit
```

Figure 6.10: Dense Coding Implementation in QPL

```
program DC_Spec
input  q0,q1:qbit
output q0,q1:qbit
```

Figure 6.11: Dense Coding Specification in QPL

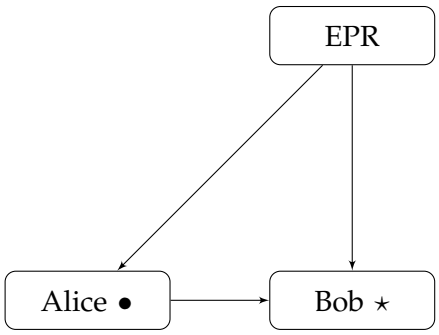


Figure 6.12: Dense Coding Diagrammatic

```
// Should only be checked on "standard" basis

//EPR prepares and send entangled pair:

newqubit a. newqubit b . H(a). CNOT(a,b) . c!a . d!b . nil

|

//Alice operations based on the classical outcome
of measurement as input:

(input x,y . c?a . m:= measure x . n:= measure y .

if m then Z(a) . if n then X(a) . q!a .nil |

|

//Bob

d?b.q?a.CNOT(a,b).H(a).output a,b.nil)
```

Figure 6.13: Dense Coding Implementation in CCS^q

```
//Specification of concurrent Dense Coding:

input x,y.output x,y.nil
```

Figure 6.14: Dense Coding Specification in CCS^q

However in this case the semantics of the given protocol is a classical binary function, rather than a superoperator, and therefore quantum equivalence checking does not apply.

6.2.1.3 *Quantum Secret Sharing*

Quantum Secret Sharing protocol was first introduced by Hillery et al. [65].[1] The original problem of secret sharing involves an agent Alice sending a classical message to two agents Bob and Charlie, one of whom is dishonest. Alice does not know which one of the agents is dishonest, so she must encode the message so that Bob and Charlie must collaborate to retrieve it. For the quantum version of this protocol, the three agents need to share a maximally entangled three-qubit state, called the *GHZ* state, prior to the execution of the protocol: $\frac{|000\rangle+|111\rangle}{\sqrt{2}}$. we assume that Charlie will end up with the original qubit (a variation of the protocol will allow Bob to end up with it or Alice can choose who gets the secret, randomly). The body of the protocol has two main phases: committing a secret by Alice and collaboration of agents to retrieve the secret. First, Alice entangles the input qubit with her entangled qubit from a previously distributed GHZ state. Then Alice measures her qubits and sends the outcome to Charlie (committing the secret). Bob also measures his qubit and sends the outcome to Charlie. Finally, Charlie is able to retrieve the original qubit once he has access to the bits from Alice and Bob (collaboration

1 There is another Quantum Secret Sharing protocol, called *Graph State Secret Sharing* [78], for sharing a classical bit or qubit, based on the same idea of using multiparty entangled states.

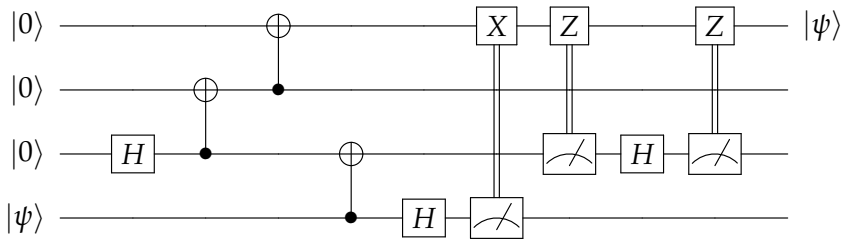


Figure 6.15: Secret Sharing Implementation in Quantum Circuit

and retrieval of the secret). The security of this protocol is a consequence of *no-cloning* theorem and is discussed in [65].

Secret Sharing in Quantum Circuit

Secret sharing can be modelled, *flatly*, by a quantum circuit. In Figure 6.15, the first line corresponds to Charlie, who retrieves the secret. Second line corresponds to Bob, who is Charlie' s collaborator. Finally the third and forth lines correspond to Alice, who gives the circuit and commits the secret. The goal in quantum circuit sharing is to retrieve a secret (quantum state of a qubit) as it was, so the specification circuit is the same as Teleportation (see Figure 6.4).

Secret Sharing in QPL

The sequential implementation of secret sharing in QPL is a program with one input and output, as it is shown in Figure 6.16, where the input qubit is $q0$ and the output qubit is $q3$. The specification in this case is the same as Teleportation in Figure 6.5.

The problem with sequential modelling (in circuits and QPL) is that it does not capture the *concurrent* and *collaborative* nature of quantum secret sharing. However, in concurrent QEC we can specify quantum secret sharing in a more expressive way with explicit communications between involved agents, according to the general structure presented in Figure 6.17. Figure 6.18 shows how to model this protocol in CCS^q

Secret Sharing in CCS^q

```
input q0:qbit
//Preparing GHZ state
newqbit q1 ; newqbit q2 ; newqbit q3 ;

q1*=H; q1q2*=CNot; q2q3*=CNot;

//Alice commits a secret
q0q1*=CNot ; q0*=H ;
measure q0 then q3*=X else q3*=I end;
measure q1 then q3*=Z else q3*=I end;

//Bob and Charlie retrieve the secret
q1*=H;
measure q1 then q3*=Z else q3*=I end;
output q3:qbit
```

Figure 6.16: Secret Sharing Implementation in QPL

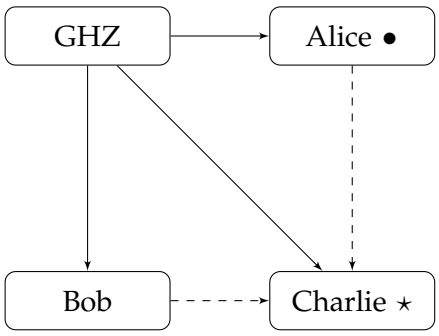


Figure 6.17: Secret Sharing Diagrammatic

with four separate and communicating processes. The specification of this protocols is the same as Teleportation in Figure 6.7.

6.2.2 *Quantum Error Correction Protocols*

In this section we present examples of quantum error correction protocols. The mathematical model behind these protocols is discussed in Section 2.4.

```
//Preparing GHZ state and sending to
//Alice, Bob and Charlie:
newqubit a . newqubit b . newqubit c . H(a) . CNOT(a,b) .

CNOT(b,c) . d ! a . e ! b . f ! c . nil

|

//Alice, who commit her qubit as a secret:
(input x . d ? a . CNOT(x,a) . H(x) . m := measure x .

n:=measure a . t ! m . w ! n . nil

|

//Bob, who is chosen as a collaborator:
(e ? b . H(b) . o:=measure b . u ! o .  nil

|

//Charlie, who recovers the original quit from Alice:
f ? c . t ? m . w ? n . u ? o . if o then Z(c) .

if m then X(c) . if n then Z(c) . output c . nil))
```

Figure 6.18: Quantum Secret Sharing Implementation in CCS^q

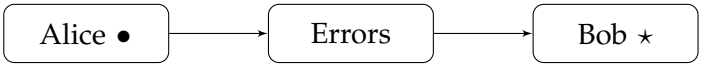


Figure 6.19: Error Correction Diagrammatic

Quantum systems in general are *continuous* however, discrete analysis of quantum systems as we do in verification of them, has a longer history in QIP. A beautiful example is *Stabilizer Codes* which protect quantum information against the environment's errors, an important requirement of building QIP systems. The intriguing feature of these codes is that they are completely definable in the stabilizer formalism, therefore can be analysed in our equivalence checking tool. The general structure of error correction protocols, as it is shown in Figure 6.19, consists of three main components of Alice, who encodes the information, Errors who alters the information by introducing errors, and Bob who decodes the information.

Several stabilizer codes have been developed [84, p. 453]. Here we explain the most compact codes such as bit error codes (bit flip and phase flip) and five qubit code, introduced in [73] and also implemented in the laboratory [68]. Interestingly, the five qubit code protects a single qubit against a *continuum* of errors only by protecting it against a *discrete* subset of them, namely, *bit flip* and *phase flip* errors. The design of error correcting protocols is very different than classical protocols in the sense that no-cloning theorem does not allow to encode and send several copies of information, in the same way that classical repetition codes work.

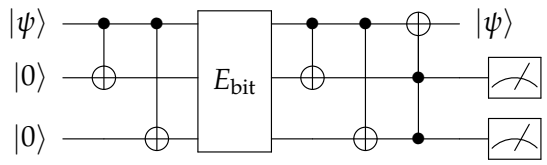


Figure 6.20: Bit Flip Code Implementation in Quantum Circuit

In the following first we explain bit error codes, then five qubit code will be studied.

6.2.2.1 *Bit Error Codes*

There are two kinds of bit error codes, namely bit flip and phase flip codes. The firsts one, which has a classical analogue, corrects qubit flipping error, that is when the effect of error is applying Pauli operator X on a qubit. A more interesting example is phase flip code, where there is no classical analogue, and which corrects errors with the effect of flipping the phase of encoded qubits, in other words applying a random Z operator on them.

The encoding phase of bit error codes entangles the input of the protocol, a qubit which we want to protect, with two ancillary qubits. Then errors are introduced, randomly, on one of the encoded qubits. Finally, the original qubit is retrieved by applying right correcting operations. In the last phase we need to introduce new qubits as error syndromes, since measurement destroys the original state of the qubit.

Bit flip code in Quantum Circuit

The implementation of bit flip code is described by the circuit in Figure 6.20. Here, the E_{bit} block means a random Pauli operator X (as error) is applied.

00 do nothing
01 apply σ to third qubit
10 apply σ to second qubit
11 aplly σ to first qubit

Figure 6.21: Corrections of bit error codes

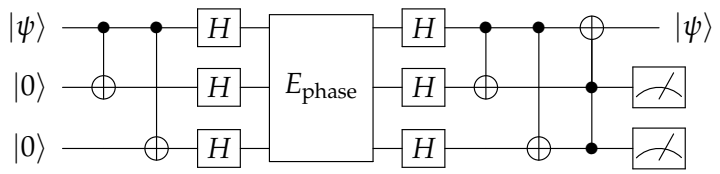


Figure 6.22: Phase flip Implementation in Quantum Circuit

By the end of this circuit, syndrome measurements are performed and errors corrected depending on the outcomes of those measurements. To do this, we follow the correction, shown in Figure 6.21, where in the bit flip case $\sigma = X$. The specification circuit is of course the same as Teleportation (Figure 6.4).

Phase flip code in Quantum Circuit

The implementation of phase flip is slightly different, because we need to apply syndrome measurements in a different basis. Thus we apply Hadamard gate to the qubits in the encoding and decoding phase. Phase flip can be modelled using the circuit in Figure 6.22, where E_{phase} denotes error block with a random Z error on one of the qubits. In this example, the correction is done according to Figure 6.21, where it is assumed $\sigma = Z$. The specification circuit is the same as bit flip code.

Bit Error codes in QPL

Error correction codes can be implemented in sequential QEC as well, where unlike quantum circuits, the correction phase can be in-

tegrated into a single QPL program, avoiding informal description of the protocol. Moreover, we use random measurement, as a good and natural source of randomness, to apply errors on the encoded qubits. Figure 6.23 shows the implementation of bit flip code in QPL. The specification of this protoocols is the same as Figure 6.5. Similarly, phase flip code is implemented in Figure 6.24, with the same specification program as bit flip code.

Bit Error codes in CCS^q

Finally, Figures 6.25 and 6.26 show the implementation of bit error codes in a concurrent model. There are three processes of Alice (encoder), Error (noises) and Bob (decoder), following the structure in Figure 6.19. An advantage of extending *CCS*^{q} with *match* expression is that we can deal with explicit error syndromes and apply necessary corrections in an easier way. The specification for these porotocol is the same as Figure 6.7.

We conclude this section by mentioning that in our models, errors (bit or phase flip) are introduced on **at most** one qubit. However, in general errors can occur in more than one encoded qubits, resulting in a more complicated scenario. In that case the original (input) qubit can not be *perfectly* recovered, but with high *probability* errors can be corrected.

6.2.2.2 *Five Qubits Code*

The idea of combining phase flip and bit flip codes into a single code first appeared in Shor's nine qubits protocol [84, p. 430]. Then it was optimized to Steane seven qubit protocol. Finally, Laflamme et al. [73]

```
program Bit_Flip_Implementation
input q0:qbit

//Encoding phase
newqbit q1; newqbit q2;

q0q1*=CNot; q0q2*=CNot;

//Generating random noise using measurement:
//either do nothing, or apply X to one of q0,q1,q2

newqbit q3; newqbit q4;

q3*=H; q4*=H;

measure q3
then

{measure q4 then q0*=X else q1*=X end}
else

{measure q4 then q2*=X else {} end}

end;

//Bob detects the error syndrome and corrects errors
newqbit q5; newqbit q6;

q0q5*=CNot; q1q5*=CNot;

q0q6*=CNot; q2q6*=CNot;

measure q5
then

{measure q6 then q0*=X else q1*=X end}
else

{measure q6 then q2*=X else {} end}

end;

//Bob recovers Alice's qubit
q0q1*=CNot; q0q2*=CNot;

output q0:qbit
```

Figure 6.23: Bit Flip Implementation in QPL

```
program Phase_Flip_Implementation
input q0:qbit

//Encoding phase
newqbit q1; newqbit q2;

q0q1*=CNot; q0q2*=CNot;
q0*=H ; q1*=H ; q2*=H ;

//Generating random noise using measurement:
//either do nothing, or apply X to one of q0,q1,q2

newqbit q3; newqbit q4;

q3*=H; q4*=H;

measure q3
then

{measure q4 then q0*=Z else q1*=Z end}
else

{measure q4 then q2*=Z else {} end}

end;

//Bob detects the error syndrome and corrects errors
q0*=H ; q1*=H ; q2*=H ;

newqbit q5; newqbit q6;

q0q5*=CNot; q1q5*=CNot;

q0q6*=CNot; q2q6*=CNot;

measure q5
then

{measure q6 then q0*=Z else q1*=Z end}
else

{measure q6 then q2*=Z else {} end}

end;

//Bob recovers Alice's qubit
q0q1*=CNot; q0q2*=CNot;

output q0:qbit
```

Figure 6.24: Phase Flip Implementation in QPL

```
//Alice Encoding the input
input x . newqubit a . newqubit b . CNOT(x,a) . CNOT(x,b).

c!x . d!a . e!b . nil

|

//Error generating random errors:
(c?x . d?a . e?b . newqubit w . newqubit z . H(w) . H(z) .

k:=measure w . l:=measure z . match k:0 and l:1 then X(x).
match k:1 and l:0 then X(a) . match k:1 and l:1 then X(b).

f!x . g!a . h!b . nil

|

//Bob corrects errors
f?x . g?a . h?b . newqubit s . newqubit t .

CNOT(x,s) . CNOT(a,s) . CNOT(x,t) . CNOT(b,t) .
m:= measure s . n:=measure t .

match m:1 and n:0 then X(a) . match m:0 and n:0 then X(b).
match m:1 and n:1 then X(x) .

CNOT(x,a) . CNOT(x,b) . output x . nil)
```

Figure 6.25: Bit Flip Implementation in CCS^q

```
//Alice Encoding the input
input x . newqubit a . newqubit b . CNOT(x,a) . CNOT(x,b).

H(x) . H(a) . H(b) .
c!x . d!a . e!b . nil

|

//Error generating random errors:
(c?x . d?a . e?b . newqubit w . newqubit z . H(w) . H(z) .

k:=measure w . l:=measure z . match k:0 and l:1 then Z(x).
match k:1 and l:0 then Z(a) . match k:1 and l:1 then Z(b).

f!x . g!a . h!b . nil

|

//Bob corrects errors
f?x . g?a . h?b .
H(x) . H(a) . H(b) .

newqubit s . newqubit t .

CNOT(x,s) . CNOT(a,s) . CNOT(x,t) . CNOT(b,t) .
m:= measure s . n:=measure t .

match m:1 and n:0 then Z(a) . match m:0 and n:0 then Z(b).
match m:1 and n:1 then Z(x).

CNOT(x,a) . CNOT(x,b) . output x . nil)
```

Figure 6.26: Phase Flip Implementation in CCS^q

showed that there is a five qubit protocol which does the same job as Shor's and Steane's codes. The interesting fact about these codes is they can protect a *single* qubit not only from bit flip and phase flip errors, but also from *arbitrary* errors.

In the followings we explain how the model of five qubit code is constructed: first the qubit that we want to protect against noises, is encoded along with four other qubits according to [73]. Secondly, we consider 16 different noises applied on a *single* qubit of the system, that is including no error, Pauli X (bit flip), Pauli Z (phase flip) and combined Pauli XZ (bit and phase flip) errors.

Five qubit code in Quantum Circuit

The circuit that implements five qubit code is presented in Figure 6.27. The input of the protocol is a qubit in the state $|\psi\rangle$. Here L denotes the unitary operator XH, applying a Hadamard operator followed by a Pauli operator X. The error block applies an arbitrary unitary on one of the encoded qubits, randomly. Finally, the correction operator U_{abcd} is determined by the syndrome measurement outcomes a, b, c and d. The protected qubit can be fully recovered by applying the correction operator on $|\psi\rangle$. The specification circuit is the same as Figure 6.4.

The sequential language QPL, as we have seen in Chapter 3, has a very simple structure. In particular measurement outcomes are implicit in the measurement terms. This is particularly unhelpful when we have need to read measurement syndromes in a explicit way, in order to correct errors. Therefore specification of five qubit code in QPL will lead to an unreadable model, and for this reason we have not

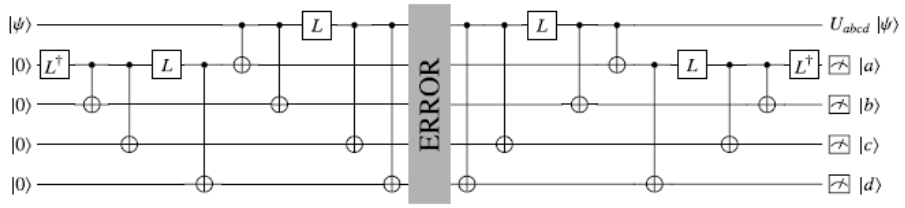


Figure 6.27: Five Qubit Code Implementation in Quantum Circuit

implemented five qubit code in the sequential equivalence checker. Nevertheless, a sequential model of this protocol in CCS^q with the use of matching terms for syndrome detection, is constructed and verified in the concurrent equivalence checker.

Five qubit code in CCS^q

Also, a concurrent model of five qubit code can be implemented in the concurrent QEC, conveniently with explicit errors syndrome handling, as illustrated in Figure 6.28. The specification is the same as Figure 6.7.

We conclude this section by the following remarks on the linear models of error corrections and another source of errors.

Remark 6.2 *An alternative verification method for quantum error correction is to construct models separately for each kind of errors e.g. in five qubit code we could build 16 separate models, instead of generating random errors with quantum measurements. This is another property of linearity in our protocols models.*

Remark 6.3 *Five qubit code can also protect a single qubit against* adversary *measurements, with high probability. Nevertheless, in the current version of QEC we are not able to implement such error correction with close to perfect recovery.*

```
//Five qubits Error Correction code:

// Alice Encoding process:
input x . newqubit a . newqubit b . newqubit c .
newqubit d . H(a) . X (a) .CNOT(a,b) . CNOT(a,c).
X(a) . H(a) . CNOT(a,d) . CNOT(x,a) . CNOT(x,b) .
X(x) . H(x) . CNOT(x,c) . CNOT(x,d) .
r ! x . s ! a . t ! b . u ! c . w ! d . nil

|

//Error process introducing random errors:
(r ? x . s ? a . t ? b . u ? c . w ? d . newqubit ea .
newqubit eb . newqubit ec . newqubit ed  . H(ea). H(eb).
H(ec) . H(ed) . em:=measure ea .
en:=measure eb . eo:=measure ec . ep:=measure ed.
match em:0 and en:0 and eo:0 and ep:0 then X(x).
match em:1 and en:0 and eo:0 and ep:0 then X(a).
match em:0 and en:1 and eo:0 and ep:0 then X(b).
match em:0 and en:0 and eo:0 and ep:1 then X(c).
match em:1 and en:1 and eo:0 and ep:0 then X(d).
match em:0 and en:1 and eo:1 and ep:0 then Z(x).
match em:0 and en:0 and eo:1 and ep:1 then Z(a).
match em:1 and en:0 and eo:0 and ep:1 then Z(b).
match em:1 and en:1 and eo:1 and ep:0 then Z(c).
match em:0 and en:1 and eo:1 and ep:1 then Z(d).
match em:1 and en:0 and eo:1 and ep:1 then X,Z(x).
match em:1 and en:1 and eo:0 and ep:1 then X,Z(a).
match em:1 and en:1 and eo:1 and ep:1 then X,Z(b).
match em:0 and en:1 and eo:0 and ep:1 then X,Z(c).
match em:0 and en:0 and eo:1 and ep:0 then X,Z(d).
rr ! x . ss ! a . tt ! b . uu ! c . ww ! d . nil

|

//Decoding process, detecting syndromes and correction:
rr ? x . ss ? a . tt ? b . uu ? c . ww ? d . CNOT(x,d).
CNOT(x,c) . X(x) . H(x) . CNOT(x,b) . CNOT(x,a) . CNOT(a,d).
X(a) . H(a) . CNOT(a,c) . CNOT(a,b) . H(a) . X(a).
sm:=measure a . sn:=measure b . so:=measure c . sp:=measure d.
match sm:1  and sn:0  and so:1  and sp:1  then X,Z(x).
match sm:1  and sn:0  and so:0  and sp:0  then X(x).
match sm:0  and sn:0  and so:1  and sp:0  then X,Z(x).
match sm:1  and sn:1  and so:1  and sp:1  then X,Z(x).
match sm:1  and sn:0  and so:0  and sp:1  then X,Z(x).
match sm:1  and sn:0  and so:1  and sp:0  then X,Z(x).
match sm:0  and sn:1  and so:1  and sp:0  then Z(x).
match sm:1  and sn:1  and so:0  and sp:1  then X(x).
match sm:0  and sn:0  and so:1  and sp:1  then X(x).
match sm:1  and sn:1  and so:1  and sp:0  then Z(x).
match sm:0  and sn:0  and so:0  and sp:0  then Z(x).
match sm:0  and sn:1  and so:0  and sp:1  then X,Z(x).
match sm:0  and sn:1  and so:0  and sp:1  then X,Z(x).
match sm:0  and sn:1  and so:0  and sp:0  then X(x).
match sm:0  and sn:1  and so:1  and sp:1  then X(x).
match sm:1  and sn:1  and so:0  and sp:0  then Z(x).
match sm:0  and sn:0  and so:0  and sp:1  then Z(x).
output x . nil )
```

Figure 6.28: Five Qubit Code Implementation in CCS^q

6.2.3 *Fault Tolerant Protocols*

In this section, examples of fault tolerant protocols are demonstrated. The main idea of fault tolerant computation is mentioned in Section 2.4. On the other hand, there is a close relationship between quantum error correction and fault tolerant computation i. e. using error correction codes in the course of quantum computation to protect quantum systems against noises and faults. These protocols are essential for realising QIP systems, and their existence stems from Theorem 2.4.

In the following four fault tolerant protocols are analysed: two variations of Teleportation [105] along with two protocols for the fault tolerant implementation of $CNOT$ gates [105, 59].

6.2.3.1 *X and Z Teleportation*

The implementation of fault tolerant protocols involves designing blocks of qubits and gates in which joint quantum operations are only allowed inside of each block and not between different blocks. For example, in Teleportation protocol we only apply quantum operations inside Alice, EPR or Bob process. However if we were allowed to apply a CNOT operation on two qubits one from Alice and one from Bob, (this is called prohibited CNOT) we would end up in a different model for Teleportation. These are called X-teleporation and Z-Teleportation and explained below.

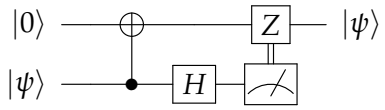


Figure 6.29: Z-Teleportation Implementation in Quantum Circuit

```
input q0:qbit

newqbit q1;

q0q1*=CNot;

q0*=H ;

measure q0 then q1*=Z else q1*=I end;

output q1:qbit
```

Figure 6.30: Z-Teleportation Implementation in QPL

X Teleportation in Quantum Circuit and QPL

The implementation of X-Teleportation in quantum circuit contains one prohibited CNOT gate and only one classical bit that is sent from Alice to Bob, following a measurement by Alice. The Bob's correction therefore is applying one Pauli Z operation, if the bit value is 1, as illustrated in Figure 6.29. The specification circuit is the same as Teleportation, in Figure 6.4.

Z Teleportation in Quantum Circuit and QPL

Similarly, Z-Teleportation is implemented in QPL as a program, where only one measurement is needed. This is shown in Figure 6.30, where its specification is identical to the one in Figure 6.5.

The implementation of X-Teleportation has a similar structure to Z-Teleportation. Again there is only one measurement, where depending on its outcome, Bob applies a single Pauli X on his qubit. Figure 6.31 and Figure 6.32 shows the implementation of X-Teleportation in circuit and QPL model.

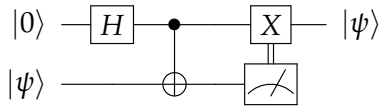


Figure 6.31: X-Teleportation Implementation in Quantum Circuit

```
input q0:qbit

newqbit q1;

q1*=H;

q1q0*=CNot;

measure q0 then q1*=X else q1*=I end ;

output q1:qbit
```

Figure 6.32: X-Teleportation Implementation in QPL

A more interesting model for the above protocols can be built in our concurrent QEC. In order to apply a prohibited joint operation in these protocols, we can use **concurrency**, by sending each of the involved qubits in a prohibited operation to a common process, followed by applying the joint operation and sending them back to their original process (block). In the concurrent implementation of X and Z Teleportation, specified in CCS^q, there are three process : Alice, who

X/Z Teleportation in CCS^q

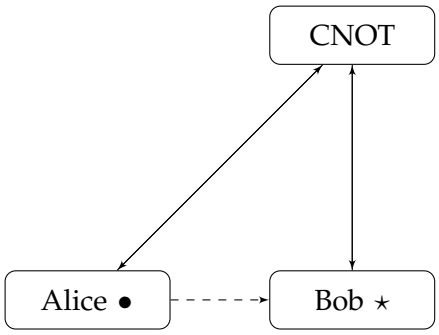


Figure 6.33: X/Z-Teleportation Diagrammatic

```
//Bob process:
newqubit a . c ! a . g ? a . e ? b . if b then Z(a) .
output a . nil

|

//Intermediate process for applying joint operations:
( c ? a . d ? x . CNOT(x,a) . f ! x . g ! a . nil

|

//Alice process:
input x . d ! x . f ? x . H(x) . b:= measure x .
e ! b . nil)
```

Figure 6.34: Z-Teleportation Implementation in CCS^q

sends her qubit, an intermediate process for joint operations and Bob who receives only one bit of classical information and retrieves Alice's qubit state, interacting to each other according to Figure 6.33. The implementation models of these protocols are shown in Figure 6.34 and Figure 6.35, where the specification is the same as Teleportation.

6.2.3.2 *Remote CNOT*

In remote *CNOT* protocol, Alice and Bob want to perform a joint *CNOT* gate between their qubit without exchanging any qubit to each other. However, they are allowed to use prior entanglement and classical communication. In this section we present two constructions of such protocols introduced in [105] and [59].

The idea of the first remote *CNOT* protocol is that Alice runs a *X*-Teleportation and Bob runs a *Z*-Teleportation , in parallel to each other. To achieve that, two pairs of entangled qubits must be used along with communicating four classical bits.

```
//Alice process:
input x . d ! x . f ? x . b:=measure x . g ! b . nil

|

//Intermediate process for applying joint operations:
(c ? a . d ? x . CNOT(a,x) . e ! a . f ! x . nil

|

//Bob process:
newqubit a . H(a). c ! a . e ? a . g ? b . if b then X(a).
output a.nil)
```

Figure 6.35: X-Teleportation Implementation in CCS^q

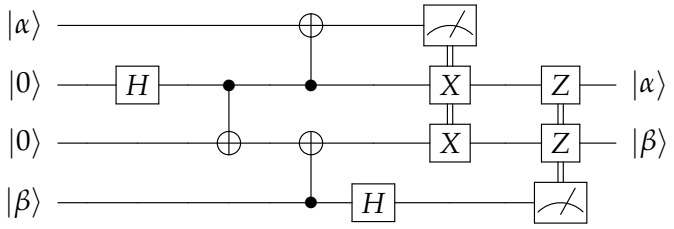


Figure 6.36: Remote CNOT Implementation in Quantum Circuit

remote CNOT in Quantum Circuit

Figure 6.36 shows how remote *CNOT* is implemented in the quantum circuit model. The specification circuit in this example receives two states $|\alpha\rangle$ and $|\beta\rangle$ and applies a single *CNOT* operation, as presented in Figure 6.37.

remote CNOT QPL

Alternatively, remote *CNOT* can be implemented in sequential QEC. Figure 6.38 shows the implementation of remote *CNOT* in QPL. In this model there are two measurements, corresponding to Alice's and Bob's correction. The specification in this case is a QPL program

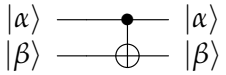


Figure 6.37: Remote *CNOT* Specification in Quantum Circuit

```
input q0,q1:qbit

newqbit q2;
newqbit q3;

q2*=H;
q2q3*=CNot;

q2q0*=CNot ;
q1q3*=CNot ;q3*=H;

measure q0 then {q2*=X;q3*=X} else q2*=I end ;
measure q1 then {q2*=Z;q3*=Z} else q2*=I end ;

output q2,q3:qbit
```

Figure 6.38: Implementation of remote CNOT in QPL

```
input q0,q1:qbit

q0q1*=CNot;

output q0,q1:qbit
```

Figure 6.39: Specification of remote CNOT in QPL

which receives $q0, q1$ as input, applies a $CNOT$ between them and return them as output, shown in Figure 6.39.

Finally, we have implemented remote $CNOT$ in concurrent QEC. In this model there are four processes: Feeder, EPR, Alice and Bob. Joint operations between two blocks, namely Alice and Bob, are not allowed. The role of Feeder is to distribute input qubits to Alice and Bob. EPR process shares an entangled pair to Alice and Bob, another pair of qubits will be entangled later by Bob.

The general structure of this protocol's concurrent model follows from Figure 6.40. The implementation and specification of remote

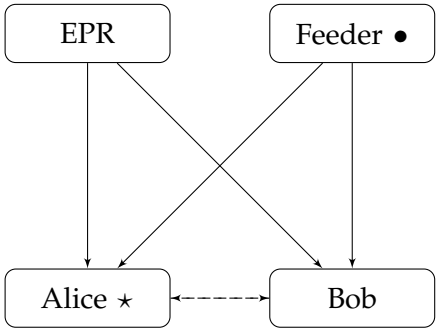


Figure 6.40: Remote CNOT Diagrammatic

CNOT in CCS^q are illustrated in Figures 6.41 and 6.42, respectively. In the implementation, Alice will have the output of the protocol, alternatively Bob could posses the output of the protocol. One can see from the implementation model that this protocol is in fact a concurrent run of a *X*-Teleportation with a *Z*-Teleportation. However, the concurrent nature of this protocol is not expressible in neither circuit diagram, nor sequential QPL programs.

remote CNOT CCS^q

The second version of remote *CNOT* has been introduced in [59]. The structure of this protocol is different in the sense that it does not run two Teleportations, and therefore reduces classical communications to only three bits. The quantum circuit that implements this protocol (as we call it Remote *CNOT* (A)) is shown in Figure 6.43. Similarly the QPL model for this example is shown in Figure 6.44.

In the concurrent model, Remote *CNOT* (A) has similar general structure as before, with the only difference that the original input qubits are returned as the output of protocols, not the ancillary qubits as in the previous version. Figure 6.45 illustrates the implementation of the alternative remote *CNOT* protocol is CCS^q.

```
//Feeder of inputs
input x,y . e ! x . p ! y . nil

|

//EPR process sharing entanglement
(newqubit a . newqubit b . H(a) . CNOT(a,b) .

c ! a . d ! b . nil

|

//Alice process (Block 1)
( e ? x . c ? a . CNOT(a,x) . u := measure x .

if u then X(a) . f ! u . g ? t . if t then Z(a) . h ? b .

output a,b . nil

|

//Bob process (Block2)
p ? y . d ? b . CNOT (y,b) . H(y). f ? u . if u then X(b) .

t:=measure y . if t then Z(b) . g ! t . h ! b . nil))
```

Figure 6.41: Implementation of remote CNOT in CCS^q

```
input x,y . CNOT(x,y) . output x,y . nil
```

Figure 6.42: Specification of remote CNOT in CCS^q

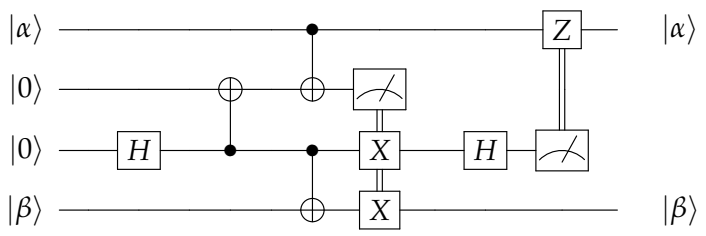


Figure 6.43: Implementation of Remote CNOT(A) in Quantum Circuit

```
input q0,q1:qbit

newqbit q2; newqbit q3 ;

q2*=H ; q2q3*=CNot ;

q2q0*=CNot ; q1q3*=CNot ;
q1*=H ;

measure q0 then {q2*=X ; q3*=X} else q2*=I end ;

measure q0 then {q2*=Z ; q3*=Z} else q3*=I end ;

output q2,q3:qbit
```

Figure 6.44: Implementation of remote CNOT(a) in QPL

```
//Feeder of inputs
input x,y . e ! x . p ! y . nil

|

//EPR process sharing entanglement
(newqubit a . newqubit b . H(b) . CNOT(b,a) .

c ! a . d ! b . nil

|

//Alice process (Block 1)
( e ? x . c ? a . CNOT(x,a) . u := measure a . f ! u .

g ? t . if t then Z(x) . h ? y .

output x,y . nil

|

//Bob process (Block2)
p ? y . d ? b . CNOT (b,y) . f ? u . if u then X(b) .

if u then X(y) . H(b) . t:=measure b .

g ! t . h ! y . nil))
```

Figure 6.45: Implementation of remote CNOT(a) in CCS^q

Protocol	Number of Interleavings	CM	Number of Branches	SM	SEC
Teleportation	400	343	16	39	43
Dense Coding	100	120	4	22	30
Bit flip code	16	62	16	60	61
Phase flip code	16	63	16	61	62
Five qubit code	64	500	64	451	n/a
X-Teleportation	32	63	8	18	25
Z-Teleportation	72	78	8	19	27
Remote CNOT	78400	12074	64	112	140
Remote CNOT(A)	23040	4882	64	123	156
Quantum Secret Sharing	88480	13900	32	46	60

Figure 6.46: Experimental results of equivalence checking of quantum protocols. The columns headed by CM and SM show the results of verification of concurrent and sequential models of protocols in the current tool. Column SEC shows verification times for sequential models in our previous tool [9]. The number of branches for SM and SEC models are the same. Times are in milliseconds.

6.3 EXPERIMENTAL RESULTS

In this section we report on the experimental results of verification of the protocols described in the previous section and also their comparison with [9]. The tool was run on a 2.5GHz Intel Core i3 machine with 4GB RAM, and is available at [8]. In the concurrent model, the number of interleavings is reported in addition to the time taken by QEC to apply equivalence checking, whereas in sequential models, the number of branches is presented along with running times. Figure 6.46 demonstrates the verification times and the comparisons.

For each protocol we have also implemented and verified sequential models in the concurrent version of QEC. Because scheduler in

this version extracts schedules directly from abstract syntax tree, as detailed in Section 6.1, the running times of sequential models verification are less than the previous version, where branches had been extracted from the complete program graph.

The experimental results show how concurrency affects quantum systems. Not surprisingly, with more sharing of entanglement and increased classical and quantum communication, we have to deal with a larger number of interleavings, particularly in the last three protocols of Figure 6.46.

However, error correction protocols are inherently sequential and therefore verifications of sequential and concurrent models in these cases produce similar results.

An advantage of our tool is that we can change the level of concurrency in the models, however we expect the impact of increasing concurrency to be significant as we can see in the results of the two remote *CNOT* protocols. The second version is three times faster than the other, because it has less classical communications.

We would like to compare our results with those produced by the model checker QMC [56], but we have not been successful in running all the examples. This is partly because QMC is based on a different approach to verification i. e., temporal logic model checking, rather than equivalence checking.

The tool Quantomatic [38] is not fully automatic, therefore we are not able to provide verification time comparisons of our case studies in that tool, as well.

We conclude this Chapter by the following final remarks: first, we can easily add more inputs to each of our protocols, which means that we are checking e. g. teleportation of one qubit in the presence of *entanglement* with other qubits. This follows from linearity, but it is never explicitly stated in standard presentations of Teleportation.

Secondly, we can model different implementations of a protocol, e. g. by changing the amount of concurrency. These differences are invisible at the level of circuit diagrams or sequential programs.

Quantum Information is more like the information in a dream. Trying to describe your dream changes your memory. Also you cannot prove to someone else what you dreamed.

— Charles H. Bennett

7

CONCLUSION AND FUTURE WORK

In this thesis we have developed the theory and practice of equivalence checking for the veriifcation of quantum information systems. We have introduced a process algebraic language to specify concurrent quantum protocols, and for the class of functional protocols in this language, a superoperator semantics is defined. By taking advantage of the linearity of superoperators, we developed techniques to check the equivalence of the implementation and specification of concurrent quantum protocols. We have built a tool QEC and applied quantum equivalence checking to a number of case studies. While we are restricted to the stabilizer formalism, we were able to verify protocols with arbitrary input due to the linearity of superoperators.

In the following, we highlight the main results that have been achieved in this thesis and the limitations of our work:

- A process algebraic language is introduced to model various concurrent QIP protocols. For this language, an operational semantics and a superoperator semantics are defined. We also defined the notion of functional protocols, which are used for our case studies.

- For our verification technique, the requirement that protocols should satisfy is the equivalence of their specification and implementation. We used the the superoperator semantics to check this requirement, by devising a verification algorithm that checks the equivalence of implementation and specification. This is done by showing the equivalence of their corresponding superoperators for all quantum states in the stabilizer basis. Therefore, this algorithm *proves* that for all inputs (where there is a continuum of input states), the implementation of protocols is equivalent to its specification, in a finite number of steps.

- We have designed a tool, QEC, to automate the above verification procedure by implementing of stabilizer simulation algorithms and other algorithms that we introduced and used such as independence checking equality test, all of them based on the stabilizer formalism. Unlike Quantomatic, our tool has a programming interface and is fully automatic. Also, the results of QEC stand as proofs of correctness of protocols rather than evidence (similar to QMC) of correctness.

- Our tool cannot be used directly when the correctness of protocols are not specified as an equivalence, such as the security property of QKD.

- In contrast to QMC, our tool can be only applied to functional protocols. Also, continuously running protocols with input/output at intermediate points, need a more general notion of equivalence such as bisimulation, which our tool lacks at this stage.

- In this work we are not able to analyse algorithms with non-stabilizer elements, like Shor's and Grover's algorithm.

- Despite the mentioned limitations, QEC has been successfully applied to verify several case studies in Chapter 6.

Finally, we discuss how we can further develop our tool and technique in order to analyse and verify more complex quantum systems:

- A natural way of extending our work is to define and automate construction of *bisimulation* between processes that represent QIP protocols. We have already mentioned in Section 3.4, the theoretical results surrounding bisimulation for quantum processes. However, the complexity of quantum systems necessitates developing automated analysis. One approach to this problem is to compare our notion of equivalence, which is based on input-output relations, with bisimulation relations that describe the behavioural equivalence. For example, in [44] the *weak* bisim-

ulation of dense coding with the following process is presented.

$$c?x \,.\, \mathrm{Set}^{x}[q_1, q_2] \,.\, d!x \,.\, nil \tag{7.1}$$

Where $\mathrm{Set}^{x}[q_1, q_2]$ sets the joint state of $q1$ and $q2$ depending on the value x. Here it is assumed that $q1$ and $q2$ are entangled, in contrast to our case studies (Chapter 6) where we explicitly expressed preparation of entangled qubits. In Equation 7.1 no internal (τ) action is specified, so the bisimulation in this case only checks input-ouput actions. Now the question is, does our equivalence relation in this case imply weak bismulation, considering the similarity of our language to the one in [44]? This is a practically important question since there is no available tool for checking bisimilarity of quantum processes yet. On the other hand, for the same example, [44] illustrated *strong* bisimulation where specification is:

$$c?x \,.\, \tau^{7} \,.\, \mathrm{Set}^{x}[q_1, q_2] \,.\, d!x \,.\, nil \tag{7.2}$$

The specification in Equation 7.2 considers internal actions, which is beyond the scope of our notion of equivalence.

In general, we are interested in seeing how the linearity arguments can be extended in order to construct bisimulation relations in a feasible way and implement it into QEC.

While in [44], the bisimulation relation is defined using an infinite number of superoperators, Feng et al. [46] introduced a bisimulation algorithm to check symbolic ground bisimilarity of quantum processes in a feasible way. It will be useful to implement this algorithm in a tool and apply it to our case studies.

- One important technical step in extending the scope of QEC is going beyond the stabilizer formalism. Although [4] has already mentioned the possibility of implementing a limited number of non-Clifford operators with the stabilizer arrays, no one has yet integrated this idea in a tool.

  It may also be possible to extend our tool with a newly developed toolkit by Microsoft Research, called Liquid [80], for simulation of quantum circuits with arbitrary quantum gates. This tool is implemented in the F# language and includes an optimised compiler and a fault tolerant architecture. While in general, the issue of scalability will arise for a larger number of qubits, we may be able to extend QEC to analyse more complicated case studies using the Liquid simulation tool.

- Adding more features to the CCS^q language would make it more expressive and certainly broaden the applicability of QEC. In particular implementing loops and recursions will be very useful. We have already seen [94] how such language constructs can be described by superoperators.

- Fault tolerant protocols are essential for realisation of QIP systems. As we have seen in Section 2.4 and Theorem 2.4, fault tolerant protocols are possible, in principle, because of viability of quantum error correction. The close interconnection between error correction and the stabilizer formalism suggests further investigation of the formal analysis of quantum fault tolerant systems. In a very recent study, Gottesman has investigated the notion of *overhead* in fault tolerant protocols, that is the ratio of logical qubits to physical qubits (e. g. ancillary qubits). A large overhead in these protocols indicate a costly experimental implementation. While Gottesman showed [60] that at the asymptotic limit this ratio is constant, there is no threshold similar to Theorem 2.4 for overhead. So it might be possible to formally analyse fault tolerant protocols with a relatively high overhead, using our verification techniques in the absence of a theoretical lower bound for the overhead.

- Cryptographic protocols are of great importance in QIP. Extending our verification technique to formally analyse such protocols is a very interesting line for future work. The challenge here is to integrate probabilistic reasoning capabilities into our tool, since without such abilities analysing quantum cryptographic protocols is not possible. Recently the area of device independent quantum cryptography has emerged. In this formalism, it is assumed that the device performing quantum encryption

may malfunction due to adversary or noises. In particular, Vazirani and Vidick have showed that Quantum Key Distribution can be implemented in a fully device independent protocol [99]. The only assumption they have made is that the devices in the protocol are physically separated. Formal analysis of such powerful protocols is a desirable goal to pursue.

- Wiesner in [101] introduced the idea of *quantum money*, a banknote that is impossible to counterfeit, using the no-cloning theorem in quantum mechanics. Several quantum money schemes have been introduced [3]. In particular Aaronson, introduced the first public-key quantum money, known as *Stabilizer Money* [2] based on using stabilizer states. However, later a successful attack on stabilizer money scheme has been discovered in [75]. Now it may be possible to analyse stabilizer money schemes and the attacks on them using QEC, since stabilizer states are already implemented in QEC. However, we need extra features such as probabilistic reasoning, in order to fully analyse the stabilizer money protocols.

BIBLIOGRAPHY

[1] The 2012 Nobel Prize in Physics (press release). http://www.nobelprize.org/nobel_prizes/physics/laureates/2012/press.html.

[2] S. Aaronson. Quantum copy-protection and quantum money. In *Proceedings of the 2009 24th Annual IEEE Conference on Computational Complexity*, CCC '09, pages 229–242, Washington, DC, USA, 2009. IEEE Computer Society.

[3] S. Aaronson and P. Christiano. Quantum money from hidden subspaces. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 41–60, New York, NY, USA, 2012. ACM.

[4] S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70:052328, 2004.

[5] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[6] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 415–425, 2004.

[7] S. Anders and H. J. Briegel. Fast simulation of stabilizer circuits using a graph-state representation. *Phys. Rev. A*, 73:022334, 2006.

[8] E. Ardeshir-Larijani. Quantum Equivalence Checker (QEC). `http://go.warwick.ac.uk/eardeshir/qec`, 2013.

[9] E. Ardeshir-Larijani, S. J. Gay, and R. Nagarajan. Equivalence checking of quantum protocols. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 478–492. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36741-0.

[10] E. Ardeshir-Larijani, S. J. Gay, and R. Nagarajan. Automated verification of quantum protocols. *arXiv:1312.5951*, 2014.

[11] E. Ardeshir-Larijani, S. J. Gay, and R. Nagarajan. Verification of concurrent quantum protocols by equivalence checking (in press). In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 500–514. Springer Berlin Heidelberg, 2014.

[12] K. M. R. Audenaert and M. B. Plenio. Entanglement on mixed stabilizer states: normal forms and reduction procedures. *New Journal of Physics*, 7(1):170, 2005.

[13] C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[14] P. Baltazar, R. Chadha, and P. Mateus. Quantum computation tree logic: model checking and complete calculus. *International Journal of Quantum Information*, 6(2):219–236, 2008.

[15] C. H. Bennett and G. Brassard. Quantum Cryptography: Public Key Distribution and Coin Tossing. In *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, New York, 1984. IEEE Press.

[16] C. H. Bennett and S. J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Phys. Rev. Lett.*, 69:2881–2884, 1992.

[17] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, 70:1895–1899, 1993.

[18] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[19] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[20] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker BLAST: Applications to software engineering. *Int. J. Softw. Tools Technol. Transf.*, 9(5):505–525, 2007. ISSN 1433-2779.

[21] A. Bove, P. Dybjer, and U. Norell. A brief overview of Agda: A functional language with dependent types. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, pages 73–78, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03358-2.

[22] J. W. Britton and B. C. Sawyer. Engineered two-dimensional ising interactions in a trapped-ion quantum simulator with hundreds of spins. 2012.

[23] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984. ISSN 0004-5411.

[24] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986. ISSN 0018-9340.

[25] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transistion relations. In Peter B. Denyer Arne Halaas, editor, *Proceedings of the International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991.

[26] D. Bures. An extension of Kakutani's theorem on infinite product measures to the tensor product of semifinite W-algebras. *Transactions of the American Mathematical Society*, 135:199–212, 1969. ISSN 0002-9947.

[27] A. R. Calderbank and P. W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, August 1996.

[28] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[29] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[30] R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In Rajeev Alur and ThomasA. Henzinger, editors, *Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-61474-6.

[31] B. Coecke and R. Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

[32] V. Danos, E. Kashefi, and P. Panangaden. The measurement calculus. *J. ACM*, 54(2), 2007. ISSN 0004-5411.

[33] T. A. S. Davidson. *Formal Verification Techniques Using Quantum Process Calculus*. PhD thesis, University of Warwick, 2011.

[34] Y. Deng and Y. Feng. Open bisimulation for quantum processes. In J. C. M. Baeten, T. Ball, and F. S. Boer, editors, *Theoretical Computer Science*, volume 7604 of *Lecture Notes in Computer Science*, pages 119–133. Springer Berlin Heidelberg, 2012.

[35] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818): 97–117, 1985.

[36] D. Deutsch and R. Jozsa. Rapid Solution of Problems by Quantum Computation. *Royal Society of London Proceedings Series A*, 439:553–558, 1992.

[37] P. A. M. Dirac. *The Principles of Quantum Mechanics*. Oxford University Press, 1958.

[38] L. Dixon and R. Duncan. Graphical reasoning in compact closed categories for quantum computation. *Annals of Mathematics and Artificial Intelligence*, 56(1):23–42, 2009.

[39] R. Duncan and M. Lucas. Verifying the steane code with quantomatic. *arXiv:1306.4532*, 2013.

[40] R. Laflamme E. Knill and G. J. Milburn. A scheme for efficient quantum computation with linear optics. *Nature*, 409:45–52, 2001.

[41] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780, 1935.

[42] C. Elliott. Building the quantum network. *New Journal of Physics*, 4(1):46, 2002.

[43] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum Computation by Adiabatic Evolution. *arXiv:quant-ph/0001106*, 2000.

[44] Y. Feng, R. Duan, and M. Ying. Bisimulation for quantum processes. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 523–534, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0490-0.

[45] Y. Feng, N. Yu, and M. Ying. Model checking quantum markov chains. *Journal of Computer and System Sciences*, 79(7):1181 – 1198, 2013.

[46] Y. Feng, Y. Deng, and M. Ying. Symbolic bisimulation for quantum processes. *(to appear in) ACM Transactions on Computational Logic (TOCL)*, 2014. ISSN 1529-3785.

[47] Richard P. Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[48] S. Franke-Arnold, S. J. Gay, and I. V. Puthoor. Quantum process calculus for linear optical quantum computing. In G. W. Dueck

and D. M. Miller, editors, *Reversible Computation*, volume 7948 of *Lecture Notes in Computer Science*, pages 234–246. Springer Berlin Heidelberg, 2013.

[49] E. Gagnon. SableCC, an object-oriented compiler framework. Master's thesis, School of Computer Science, McGill University, 1998.

[50] S. J. Gay. Quantum programming languages: survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006.

[51] S. J. Gay. Stabilizer states as a basis for density matrices. *arXiv:1112.2156*, 2011.

[52] S. J. Gay and I. Mackie, editors. *Semantic Techniques in Quantum Computation*. Cambridge University Press, 2010.

[53] S. J. Gay and R. Nagarajan. Communicating Quantum Processes. *Proc. POPL*, pages 145–157, 2005.

[54] S. J. Gay, R. Nagarajan, and N. Papanikolaou. Probabilistic model-checking of quantum protocols. In *Proceedings of 2nd International Workshop on Developments in Computational Models (DCM)*, 2006.

[55] S. J. Gay, R. Nagarajan, and N. Papanikolaou. QMC: A model checker for quantum systems. *LNCS 5123 (Proc. CAV)*, pages 543–547, 2008.

[56] S. J. Gay, R. Nagarajan, and N. Papanikolaou. QMC: A model checker for quantum systems. In *Proceedings of the 20th International Conference on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 543–547. Springer, 2008. ISBN 978-3-540-70543-7.

[57] M. Gordon. Introduction to the HOL System. In *HOL Theorem Proving System and Its Applications, 1991., International Workshop on the*, pages 2–3, 1991.

[58] D. Gottesman. Class of quantum error-correcting codes saturating the quantum hamming bound. *Phys. Rev. A*, 54:1862–1868, September 1996.

[59] D. Gottesman. The Heisenberg representation of quantum computers. Technical Report LA-UR-98-2848, Los Alamos National Laboratory, 1998.

[60] D. Gottesman. Fault-tolerant quantum computation with constant overhead. *arXiv:1310.2984*, 2014.

[61] D. Gottesman and I. L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402:390–393, 1999.

[62] A. S. Green, P. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. An Introduction to Quantum Programming in Quipper. In G. W. Dueck and D. M. Miller, editors, *Reversible Computation*,

volume 7948 of *Lecture Notes in Computer Science*, pages 110–124. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38985-6.

[63] D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173(2):143 – 161, 2002. ISSN 0890-5401.

[64] K. Havelund and T. Pressburger. Model checking JAVA programs using JAVA PathFinder. *International Journal on Software Tools for Technology Transfer*, 2(4):366–381, 2000. ISSN 1433-2779.

[65] M. Hillery, V. Bužek, and A. Berthiaume. Quantum secret sharing. *Phys. Rev. A*, 59:1829–1834, 1999.

[66] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., NJ, USA, 1985. ISBN 0-13-153271-5.

[67] G. Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley, 2003. ISBN 0-321-22862-6.

[68] E. Knill, R. Laflamme, R. Martinez, and C. Negrevergne. Benchmarking quantum computers: The five-qubit error correcting code. *Phys. Rev. Lett.*, 86:5811–5814, June 2001.

[69] E. H. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.

[70] D. E. Knuth. Computer Programming as an Art. *Communications of the ACM*, 17(12):667–673, December 1974.

[71] K. Kraus, A. Böhm, J. D. Dollard, and W. H. Wootters, editors. *States, Effects, and Operations Fundamental Notions of Quantum Theory*, volume 190 of *Lecture Notes in Physics, Berlin Springer Verlag*, 1983.

[72] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004. ISSN 1433-2779.

[73] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek. Perfect quantum error correcting code. *Phys. Rev. Lett.*, 77:198–201, July 1996.

[74] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1 (1-2):134–152, 1997. ISSN 1433-2779.

[75] A. Lutomirski, S. Aaronson, E. Farhi, D. Gosset, A. Hassidim, J. Kelner, and P. Shor. Breaking and making quantum money: toward a new quantum cryptographic protocol. *arXiv:0912.3825*, 2009.

[76] X. Ma, T. Herbst, T. Scheidl, D. Wang, S. Kropatschek, W. Naylor, B. Wittmann, A. Mech, J. Kofler, E. Anisimova, V. Makarov, T. Jennewein, R. Ursin, and A. Zeilinger. Quantum teleportation over 143 kilometres using active feed-forward. *Nature*, 489 (7415):269–273, 2012. ISSN 0028-0836.

[77] Y. Manin. Computable and uncomputable (in Russian). *Sovetskoye Radio*, 1981.

[78] D. Markham and B. C. Sanders. Graph states for quantum secret sharing. *Phys. Rev. A*, 78:042309, 2008.

[79] D. Mayers. Unconditional security in quantum cryptography. *J. ACM*, 48(3):351–406, 2001. ISSN 0004-5411.

[80] Microsoft. Language-Integrated Quantum Operations. `http://research.microsoft.com/en-us/projects/liquid/`, 2013.

[81] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989. ISBN 978-0-13-115007-2.

[82] U. Montanari. True concurrency: Theory and practice. In R.S. Bird, C.C. Morgan, and J.C.P. Woodcock, editors, *Mathematics of Program Construction*, volume 669 of *Lecture Notes in Computer Science*, pages 14–17. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-56625-0.

[83] M. A. Nielsen. Quantum computation by measurement and quantum memory. *Physics Letters A*, 308(2):96–100, 2003.

[84] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[85] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. ISBN 3-540-43376-7.

[86] Office of the President of the United States. A federal vison for quantum information science. http://www.nist.gov/pml/div684/upload/FederalVisionQIS.pdf, 2009.

[87] N. Papanikolaou. *Model Checking Quantum Protocols*. PhD thesis, University of Warwick, 2009.

[88] A. Phillips and L. Cardelli. A correct abstract machine for the stochastic pi-calculus. In *Concurrent Models in Molecular Biology*, 2004.

[89] Inria Coq project. The Coq Proof Assistant. http://coq.inria.fr/.

[90] R. Raussendorf and H. J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, 2001.

[91] A.W. Roscoe. Understanding concurrent systems, 2010.

[92] D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001. ISBN 0521781779.

[93] M. Sasaki, M. Fujiwara, H. Ishizuka, W. Klaus, K. Wakui, M. Takeoka, S. Miki, T. Yamashita, Z. Wang, A. Tanaka, K. Yoshino, Y. Nambu, S. Takahashi, A. Tajima, A. Tomita, T. Domeki, T. Hasegawa, Y. Sakai, H. Kobayashi, T. Asai, K. Shimizu, T. Tokura, T. Tsurumaru, M. Matsui, T. Honjo, K. Tamaki, H. Takesue, Y. Tokura, J. F. Dynes, A. R. Dixon, A. W. Sharpe, Z. L. Yuan, A. J. Shields, S. Uchikoga, M. Legré,

S. Robyr, P. Trinkler, L. Monat, J.-B. Page, G. Ribordy, A. Poppe, A. Allacher, O. Maurhart, T. Länger, M. Peev, and A. Zeilinger. Field test of quantum key distribution in the tokyo qkd network. *Opt. Express*, 19(11):10387–10409, 2011.

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

[95] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 124–134, Washington, DC, USA, 1994. IEEE Computer Society. ISBN 0-8186-6580-7.

[96] D. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[97] A. M. Steane. A tutorial on quantum error correction. In G. Casati, D. L. Shepelyansky, P. Zoller, and G. Benenti, editors, *Proceedings of the International School of Physics "Enrico Fermi", course CLXII*, pages 1–32. IOS Press, 2006.

[98] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[99] U. Vazirani and T. Vidick. Fully device independent quantum key distribution. *arXiv:1210.1810*, 2012.

[100] G. F. Viamontes, I. L. Markov, and J. P. Hayes. *Quantum Circuit Simulation*. Springer, 2009. ISBN 978-90-481-3064-1.

[101] S. Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, 1983.

[102] M. Ying, Y. Feng, R. Duan, and Z. Ji. An algebra of quantum processes. *ACM Trans. Comput. Logic*, 10(3):19:1–19:36, 2009. ISSN 1529-3785.

[103] A. Zeilinger, D. Stucki, A. J. Shields, A. W. Sharpe, L. Salvail, S. Robyr, G. Ribordy, E. Querasser, A. Poppe, J. B. Page, S. Nauerth, L. Monat, O. Maurhardt, M. Suda, C. Tamas, T. Themel, H. Zbinden, Z. L. Yuan, I. Wimberger, H. Weinfurter, H. Weier, N. Walenta, F. Vannel, R. Tualle-Brouri, P. Trinkler, A. Treiber, Y. Thoma, R. T. Thew, T. Matyus, A. Marhold, M. Fuerst, S. Fossier, S. Fasel, J. F. Dynes, M. Dianati, E. Diamanti, T. Debuisschert, W. Boxleitner, J. Bouda, C. Barreiro, R. Alleaume, C. Pacher, J. D. Gautier, O. Gay, N. Gisin, N. Lutkenhaus, T. Lorunser, J. Lodewyck, R. Lieger, M. Legre, T. Laenger, G. Humer, H. Hubel, M. Hentschel, Y. Hasani, A. Happe, P. Grangier, and M. Peev. The SECOQC quantum key distribution network in Vienna. .

[104] A. Zeilinger, R. Ursin, T. Jennewein, and B. Furch. Quantum communications at ESA: Towards a space experiment on the ISS. pages 165–178, .

[105] X. Zhou, D. W. Leung, and I. L. Chuang. Methodology for quantum logic gate construction. *Phys. Rev. A*, 62:052316, 2000.