Verification of Concurrent Quantum Protocols by Equivalence Checking

Ebrahim Ardeshir-Larijani[1,2]*, Simon J. Gay[2], and Rajagopal Nagarajan[3]**

[1] Department of Computer Science, University of Warwick
E.Ardeshir-Larijani@warwick.ac.uk
[2] School of Computing Science, University of Glasgow
Simon.Gay@glasgow.ac.uk
[3] Department of Computer Science,
School of Science and Technology, Middlesex University
R.Nagarajan@mdx.ac.uk

Abstract. We present a tool which uses a concurrent language for describing quantum systems, and performs verification by checking equivalence between specification and implementation. 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. In particular, we consider concurrent quantum protocols that behave *functionally* in the sense of computing a deterministic input-output relation for all interleavings of the 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. Despite the limitations of the stabilizer formalism and also the range of protocols that can be analysed using this approach, we have applied our equivalence checking tool to specify and verify interesting and practical quantum protocols from teleportation to secret sharing.

1 Introduction

There have been significant advances in quantum information science over the last few decades and technologies based on these developments are at a stage well suited for deployment in a range of industrial applications. The construction of practical, general purpose quantum computers has been challenging. The only large scale quantum computer available today is manufactured by the Canadian company D-Wave. However, it does not appear to be general purpose and

* Supported by the Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, EPSRC award EP/D063191/1.

** Partially supported by "Process Algebra Approach to Distributed Quantum Computation and Secure Quantum Communication", Australian Research Council Discovery Project DP110103473.

not everyone is convinced that it is truly quantum. On the other hand, quantum communication and cryptography have made large strides and is now well established. Physical restrictions of quantum communication, like preserving photon states over long distances, are gradually being resolved, for example, by quantum repeaters [11] and using quantum teleportation. Various Quantum Key Distribution networks have been built, including the DARPA Quantum Network in Boston, the SeCoQC network around Vienna and the Tokyo QKD Network. There is no doubt that quantum communication and quantum cryptographic protocols will become an integral part of our society's infrastructure.

On the theoretical side, quantum key distribution protocols such as BB84 have been proved to be unconditionally secure [20]. It is important to understand that this an information-theoretic proof, which does not necessarily guarantee that *implemented systems* are unconditionally secure. That is why alternative approaches, such as those based on formal methods, could be useful in analysing behaviour of implemented systems.

The area of *formal verification*, despite being a relatively young field, has found numerous applications in hardware and software technologies. Today, verification techniques span a wide spectrum from model checking and theorem proving to process calculus, all of them have helped us to grasp better understanding of interactive and complicated distributed systems. This work represents another milestone in our ongoing programme of applying formal methods to quantum systems. In this paper, we present a concurrent language for describing quantum systems, and perform verification by equivalence checking. The goal in equivalence checking is to show that the implementation of a program is identical to its specification, on all possible executions of the program. This is different from property based model checking, where an intended property is checked over all possible execution paths of a program. The key idea of this paper is to check equivalence of quantum protocols by using their superoperator semantics (Section 4). Superoperators are linear, so they are completely defined by their effect on a basis of the appropriate space. To show that two protocols are equivalent, we show that their associated superoperators are equivalent by simulating the protocols for every state in a basis of the input vector space. By choosing a basis that consists of stabilizer states, we can do this simulation efficiently.

One of the main challenges here is the *explosion of states* arising from branching and concurrency of programs. This is in addition to the need to deal with the *explosion of space* needed for specifying quantum states. For a quantum state with n *qubits*(quantum bits), we need to consider 2^n complex coefficients. To avoid this problem we restrict ourself to the *stabilizer formalism* [1]. In this formalism, quantum states can be described in polynomial space and also for certain quantum operations, the evolution of stabilizer states can be done in polynomial time. Although one cannot do universal quantum computation within the stabilizer formalism, many important protocols such as Teleportation [7], Quantum Error Correction [8] as well as *quantum entanglement* can be analysed within it. Crucially, quantum error correction is a *prerequisite* for *fault tolerant* quantum computing [24]. The latter is necessary for building a scalable quantum computer, capable of doing universal quantum computing.

Contributions. This paper extends our previous work [4] substantially in two ways. First, we now use a concurrent modelling language, which means that we can explicitly represent concurrency and communication in order to model protocols more realistically. Second, we have analysed a much wider range of examples, including several standard quantum protocols.

The paper is organised as follows. In Section 2 we give preliminaries from quantum computing and the stabilizer formalism. In Sections 3 and 4, we give the syntax and semantics of our concurrent modelling language. Section 5 describes our equivalence checking technique and Section 6 presents example protocols. In Section 7 we give details of our equivalence checking tool, and present experimental results. Section 8 reviews related work and Section 9 concludes.

2 Background

In this section, we give a very concise introduction to quantum computing. For more details, we refer to [22]. The basic element of quantum information is a *qubit* (quantum bit). Qubits are vectors in an *inner product* vector space which is called Hilbert space.[4] Quantum states are description of qubits with the general form: $|\Psi\rangle = \alpha_1 |00\ldots0\rangle + \ldots + \alpha_n |11\ldots1\rangle$, where $\alpha_i \in \mathbb{C}$ are called *amplitudes* satisfying $|\alpha_1|^2 + \ldots + |\alpha_n|^2 = 1$. The so-called Ket notation or Dirac's notation is used to distinguish unit vectors $|0\rangle$ and $|1\rangle$ from classical bits 0 and 1. Also $|00\ldots0\rangle$ corresponds to *tensor product* of unit vectors (i.e $|0\rangle \otimes 0 \ldots \otimes |0\rangle$). There are two kinds of operations on quantum states, *unitary transformations* and *measurement*. The side effect of the measurement operation is classical information, for example, the outcome of measuring the above state $|\Psi\rangle$ is a classical bit string $(00\ldots0)$ with probability $|\alpha_1|^2$, to $(11\ldots1)$ with probability $|\alpha_n|^2$. Note that measurement is a destructive operation and it changes the state of a qubit permanently. Qubits can be *entangled*. For example, a two qubit entangled state $|00\rangle + |11\rangle$, which is called a *Bell* state, cannot be decomposed into two single qubit states. Measuring one of the qubits will fix the state of the other qubit, even if they are physically separated.

Some basic quantum operations and their matrix representation are shown in the Figure 1. A model for describing a quantum system is the quantum circuit

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

Fig. 1. Pauli operators

model, analogous to the classical circuit model. Each quantum circuit consists of

[4] Normally Hilbert space is defined with additional conditions which we are not concerned with in this paper.

unitary and measurement gates. Unitary gates can be applied to one and more qubits. In a certain kind of multiple qubit gates, which are called *controlled* gates, there is one or more control qubits and one or more target qubits. Depending on the value of the control qubit, a unitary gate is applied to the target qubit. Controlled-X (or *CNot*) and *Toffoli* [22, p. 29] gates are examples of controlled gates. Quantum circuits are normally described in the following way: single wires represent qubits, double wires represent classical bits. Single gates and measurement gates are depicted with squares, whereas controlled gates are shown with a point representing the control qubit and a circle depicting the target qubit with a vertical wire connecting them.

The *stabilizer* formalism is a useful scheme which characterises a small but important part of quantum mechanics. The core idea of the stabilizer formalism is to represent certain quantum states, which are called *stabilizer states*, with their *stabilizer group*, instead of an exponential number of complex amplitudes. For an n-qubit quantum stabilizer state $|\varphi\rangle$, the stabilizer group is defined by $Stab(|\varphi\rangle) = \{S|S\,|\varphi\rangle = +1\,|\varphi\rangle\}$. This group can be represented elegantly by its n generators in the Pauli group (i.e P^n for P in Figure 1). Several algorithms have been developed for specifying and manipulating stabilizer states using group representation of quantum states, see [1]. Importantly, the effect of *Clifford Operators* (members of the normaliser of Pauli group, known as Clifford group) on stabilizer states can be simulated by a polynomial time algorithm. Consequently, we have the following important theorem which guarantees stabilizer states can be specified in polynomial space and certain operations and measurement can be done in polynomial time:

Theorem 1. *(Gottesman-Knill, [22, 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 on a classical computer.

The *density operator* is an alternative way of describing a quantum state where we need to deal with uncertainty. For instance, an ensemble of a quantum state $\{(|\phi_i\rangle, p_i)\}$, where p_is are probabilities, can be represented by the following density operator:

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

where $|\phi_i\rangle\,\langle\phi_i|$ denote outer product. Density operators are *positive* and *Hermitian*, meaning they satisfy $|\varphi\rangle$: $\langle\varphi|\,\rho\,|\varphi\rangle \geq 0$ and $\rho^\dagger = \rho$ ($\dagger$ denotes transpose of the complex conjugate) respectively. Also a composite quantum system can be elegantly described in the language of density operators. In particular, one can obtain a *reduced* density operator by applying a *partial trace* operation on the density operator of a composite system, see [22, p. 105].

Superoperators are linear transforms on the space of density operators. Note that for an n-qubit system, the space of density operators has dimension of 2^{2n}. Quantum information systems can be abstracted using superoperators. In the present paper, we take advantage of the linearity of superoperators: a superoperator is uniquely defined by its action on the elements of a *basis* of the space on which it acts, which in our case is a space of density operators. We can therefore check equality of superoperators by checking that for a given basis element as input, they produce the same output. In this paper, we are interested in systems which are in the stabilizer formalism. The following result [16] explicitly constructs a basis for the space of density operators which only consists of stabilizer states and hence it can be used in our equivalence checking technique.

Theorem 2. *The space of density operators 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.*

Equality test: Another useful property of stabilizer states is that there is an efficient way of checking whether two stabilizer states are equal. One may test equality by using an algorithm for inner product of two states as in [1]. However, in [4] a novel approach is introduced which checks the linear dependence of stabilizer generators of two states. This is possible using polynomial time algorithms for obtaining stabilizer normal forms, introduced in [5]. Let $|\phi\rangle$ and $|\psi\rangle$ be stabilizer states and $Stab(|\phi\rangle)$, $Stab(|\psi\rangle)$ be their stabilizer groups, it can be easily seen that:

$$|\phi\rangle = |\psi\rangle \Longleftrightarrow Stab(|\phi\rangle) = Stab(|\psi\rangle)$$

Therefore it suffices to show that $Stab(|\phi\rangle) \subseteq Stab(|\psi\rangle)$ and $Stab(|\phi\rangle) \supseteq Stab(|\psi\rangle)$, using independence checking approach, which results in the following proposition, proved in [4].

Proposition 1. *There is a polynomial time algorithm which decides for any stabilizer states $|\phi\rangle$ and $|\psi\rangle$, whether or not $|\phi\rangle = |\psi\rangle$.*

3 Specification of Concurrent Quantum Protocols

In this section we present a concurrent language for specifying quantum protocols, based on CCS [21]. This is different from our previous work [4] since our focus now is on concurrent communicating quantum systems. We will, however, restrict attention to protocols that receive input at the beginning and produce output at the end of their execution, rather than considering more general continuous interaction. One reason to use this language is to illustrate how designing concurrent quantum protocols can be difficult and non intuitive compared to classical protocols. For example, quantum measurement is a destructive action and if it used wrongly between parallel processes, it can destroy the effect of the whole system. Our language is similar to *qCCS*. [27] Communication in our system is done using *synchronous message passing or handshaking.* This structure can be extended to describe *asynchronous* communication similar to the approach of *CCS*. However, there is another reason to consider synchronicity, namely the lack of *quantum memory.* Therefore a synchronous system is closer to the current technological level of development of quantum communication. The syntax of our language is summarised in Figure 2.

```
p ::= t |   t || t

t ::= nil | c!x.t | c?x.t | a:= measure x . t | U(x) . t |
      newqubit x . t | input list . t | output list . t |
      if x then U(y) . t | match list then U(x) . t

list ::= x:val | list,x:val

val  ::= 0 | 1
```

Fig. 2. Syntax of the Concurrent Language

Here `t||t` denotes parallel composition of processes and **.** represents process prefixes, similar to CCS. Terminated process is represented by **`nil`**. Prefix **`input`** `list` defines the input state of a protocol and **`output`** `list` stands for the intended output of a protocol. Often for the output we need to deallocate qubits. In [23], allocation and deallocation of qubits is presented by assuming that there is an operating system which gives/reset access to a pool of qubits (i.e qubits are not created or destroyed). However in this paper, we don't make that assumption and therefore allocation/deallocation of qubits has the physical meaning of creating or applying partial trace operation on quantum states.

For sending and receiving classical and quantum bits we use prefixes `c!x` and `c?x`. Measurement is done using prefix `a:=measure x`, where the outcome of measuring qubit x is assigned to a classical variable a. Conditionals, **`if`** and **`match`** impose classical conditions on the system in order to apply particular unitaries. The classical values `val` correspond to classical bits 0 and 1.

4 Semantics of Quantum Protocols

In this section we explain how the semantics of our specification language can be understood using superoperators. We apply π-calculus style reduction rules to our concurrent syntax in order to derive sequential interleavings, removing all communication and parallel composition. Finally, inspired by the semantics of Selinger's *quantum programming language (QPL)* [23], we argue that each sequential interleaving is equivalent to a QPL program, and thus can be characterised by a *superoperator.*

The reduction rules are in Figure 3. Here α denotes prefixes other than communication (i.e. not $c!x$ or $c?x$). Structural congruence is defined as associativity and commutativity of parallel composition, similarly to π-calculus, and is denoted by $\equiv$. Finally, τ represents the silent action and substitution of v with x is denoted by $[v/x]$. Using these rules, the transition graph for a concurrent

$$\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q \equiv Q'}{P \xrightarrow{\alpha} Q}\ \textsc{R-Cong} \qquad \alpha.P \xrightarrow{\alpha} P\ \textsc{R-Act}$$

$$c!v.P \parallel c?x.Q \xrightarrow{\tau} P \parallel Q[v/x]\ \textsc{R-Com} \qquad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\ \textsc{R-Par}$$

Fig. 3. Reduction Rules

process is a tree, because we do not have any form of loop or recursion, in which *interleavings* are represented by paths from the root to the leaves. For a protocol P, each interleaving $i \in \mathfrak{I}(P)$ (i.e. the set of all interleavings of P), can be translated to a QPL program, whose semantics is defined by a superoperator [23]. In this setting we assume that the `input` expression can be placed at the beginning of i, and only used once. Similarly, the `output` expression occurs at the end of each i, where we stop interpreting sequential interleavings. The sequential QPL program obtained in this way contains measurement operators which, when simulated, may generate branching behaviour due to quantum randomness. The semantics of QPL merges these branches by calculating a weighted sum of the density matrix representations of their final output quantum states. In the stabilizer formalism we cannot calculate these weighted sums in general, so instead we require that all of the final output quantum states are equal and hence the weighted sum becomes trivial. We further restrict our attention to protocols that are *functional* in the sense of computing a deterministic input-output relation despite their internal non-determinism due to concurrency and quantum randomness.

Definition 1 (Configuration). *Let $I^{\mathbb{Q}} : \mathbb{Q} \mapsto \mathbb{N}$ denote index of qubit variables in a quantum state. Configurations of a concurrent quantum program are defined by a tuples of the form $(\sigma, \kappa, |\phi\rangle)$, where σ represents classical variables assignment, κ represents channel variables assignment and $|\phi\rangle$ represents a quantum state corresponding to the qubit variables mapped by $I^{\mathbb{Q}}$.*

Definition 2. *A concurrent quantum protocol P is* functional *if*

1. *for a given quantum input $|\psi\rangle$ and a given interleaving $i \in \mathfrak{I}(P)$, all execution paths produce the same output quantum state; and*
2. *for a given quantum input $|\psi\rangle$, the unique output quantum state is the same across all interleavings.*

Proposition 2. *Any functional concurrent protocol P, specified with the language in Figure 2, defines a unique superoperator, which we denote by $[\![P]\!]$.*

Proof. The QPL programs corresponding to all of the interleavings of P map inputs to outputs in the same way, and therefore all define the same superoperator, which we take to be $[\![P]\!]$.

Remark 1. Quantum measurement in this paper is slightly different from QPL. Here the classical outcomes of measurement are assigned to classical variables for the modelling convenience. Nevertheless, one can easily translate our measurement expressions into QPL's if-then-else style form.

Remark 2. We *separate* classical and quantum data as in Definition 1 for a simpler implementation. Nonetheless, classical and quantum data can be mixed in a similar way as [23], using tuples of stabiliser states.

For example, the concurrent protocol

```
input x . a!x . nil | a?y . X(y) . nil |
newqubit u . b!u . nil | b?u . Z(u) . output u . nil
```

has the following interleavings, among others:

$$\begin{aligned}
I_1 &: \mathit{input}\, x; X(x); \mathit{newqubit}\, u; Z(u); \mathit{output}\, u \\
I_2 &: \mathit{input}\, x; \mathit{newqubit}\, u; X(x); Z(u); \mathit{output}\, u \\
I_3 &: \mathit{newqubit}\, u; \mathit{input}\, x; Z(u); X(x); \mathit{output}\, u.
\end{aligned}$$

5 Checking Equivalence of Quantum Protocols

In the following we define the equivalence between two concurrent protocols. In the classical theory of computation there are several ways of checking equivalence of concurrent systems using semantics namely: bisimulation based, automata based, game semantics and trace semantics. In this paper we focus on superoperator semantics where each interleaving of a system is described by a *superoperator*. We require that protocols be functional as in Definition 2.

In our system, 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 [1].

Having defined superoperators for concurrent quantum protocols, we explain how to verify the correctness of the protocols. We show that the specification and implementation of a protocol are equivalent by proving their corresponding supeoperators are equal. Knowing that supeoperators are linear, it suffices to execute the protocol for all elements in the stabilizer basis set of Theorem 2, to capture their effects on the input. This process is done in two phases: first functionality of specification and implementation is checked. Secondly equivalence of them will be established, in any case we require to schedule, interpret protocols in stabilizer formalism and apply equality tests on the reached states on every basis input.

The following algorithm describes how to do these two steps: for a concurrent program P, let $\mathfrak{I}(P, v)$ denote all possible interleavings of the program with initial state v (from basis set in the Theorem 2, denoted by $\mathfrak{B}$), produced by a scheduler and indexed by integers from 1 upwards. Let I_i denote the ith interleaving and suppose $StabSim^*(P, v, i)$ shows the final state given by that stabilizer simulation algorithm in [1] applied to I_i, on initial basis state v. Finally, let $EQ_S(v, w)$ be the equality test algorithm from Section 2. Then Figure 4 shows the equivalence checking algorithm for two concurrent programs P_1 and P_2, and establishes the following result.

```
for all $v \in \mathfrak{B}$ do
  for all $i \in \{1, 2\}$ do
    $|\phi_i^v\rangle = StabSim^*(P_i, v, 1)$
    for all $j \in \mathfrak{I}(P_i, v) - \{1\}$ do
      if $\neg EQ_S(StabSim^*(P_i, v, j), |\phi_i^v\rangle)$ then
        return $P_i$ non-functional
      end if
    end for
  end for
  if $\neg EQ_S(|\phi_1^v\rangle, |\phi_2^v\rangle)$ then
    return $P_1 \ncong P_2$
  end if
end for
return $P_1 \cong P_2$
```

Fig. 4. Algorithm for checking equivalence of concurrent protocols.

Proposition 3. *Given two functional concurrent quantum protocols, which only use operations in the stabilizer formalism, one can systematically decide whether they are equivalent with respect to their superoperator semantics on every possible input, by iteration on the stabilizer basis.*

Proposition 4. *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) .*

Remark 3. 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 [18] for details).

6 Examples

We have analysed a range of quantum protocols covering quantum communication, quantum fault-tolerance and quantum cryptography using our equivalence

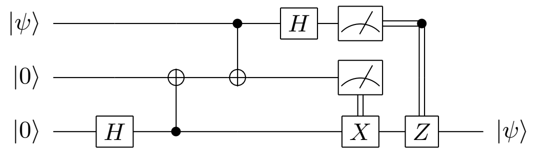

Fig. 5. Teleportation circuit

checking tool. In this section, we present two of the protocols we have verified. The remaining protocols that we have analysed, in particular, fault tolerant protocols such as one bit teleportation and remote CNOT as well as error correction protocols can be found at `http://go.warwick.ac.uk/eardeshir/qec`.

Teleportation [7]: The goal in this protocol is to *teleport* a quantum state from Alice to Bob without physically transferring qubits, using quantum entanglement. Before starting the communication between the two parties, Alice and Bob, an entangled pair is established and shared between them. Alice then entangles the input qubit with her half of the entangled qubit by applying a controlled-not gate followed by a Hadamard gate. She then measures her qubits and sends the classical outcome to the Bob. Depending on the four classical outcomes of Alice measurements, Bob applies certain X and Z operations and recovers the input state on his entangled qubit. The circuit which implements Quantum Teleportation is shown in the Figure 5.

However the circuit model does not provide a high level interface and does not capture the notion of *physical separation* between Alice and Bob. Through our concurrent language, we provide a programming interface and we can also describe the implementation of teleportation reflecting physical separation. The specification and implementation programs for teleportation in our concurrent language are shown in Figure 6.

```
//Implementation:
//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)
```

```
//Specification:
input x.output x.nil
```

Fig. 6. Teleportation: Specification and Implementation

```
// 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 wants to share her qubit:
(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))
```

Fig. 7. Secret Sharing Implementation. The specification is the same as Teleportation.

Quantum Secret Sharing: This protocol was first introduced by Hillery et. al. [19]. The original problem of secret sharing involves an agent Alice sending a message to two agents Bob and Charlie, one of whom is dishonest. Alice doesn't 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: $|000\rangle + |111\rangle$. In Figure 7, we assume that Charlie will end up with the original qubit (a variation of the protocol will allow Bob to end up with it). First, Alice entangles the input qubit with her entangled qubit from the GHZ state. Then Alice measures her qubits and sends the outcome to Charlie. 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. The specification of secret sharing is similar to teleportation, expressed in Figure 6. The security of this protocol is a consequence of *no-cloning* theorem and is discussed in [19]. The specification for this protocol is the same as for teleportation.

We conclude with some 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. Second, 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.

7 Equivalence Checker and Experimental Results

We have implemented a concurrent equivalence checker in Java [3]. The parser for the concurrent language is produced using SableCC [15]. The input of this tool is a concurrent protocol as described in Section 3. The *scheduler* generates all possible interleavings arising from execution of concurrent protocols. Each interleaving on a given input basis is passed to the interpreter which interprets programs using the Aaronson-Gottesman algorithm. The verification procedure consists of two steps. First *functionality* will be checked for a given input protocol and then, equivalence of two protocols will be determined. Both steps use the equality test algorithm in Section 2. The experimental results of verification of protocols based on the models presented in Section 6 and in [3], are summarized in the Table 8. Note that the specification of error corrections and Z/X-teleportation are the same as Figure 6, whereas remote CNOT's are specified by a single application of CNOT gate on two qubit inputs. The tool was run on a 2.5GHz Intel Core i3 machine with 4GB RAM. We would like to compare our results with those produced by the model checker QMC, 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 is not a fully automatic tool, therefore we were not able to provide a comparison with case studies in that tool as well. 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. We have verified (Figure 8) sequential models of protocols in our current and previous tools [4]. Because of the complex structure of measurements in the five qubit code, we were not able to model this protocol in the sequential equivalence checker. The scheduler in our previous tool is slower than the one in our current work. This is because we were building program graphs for extracting schedules, whereas in this work schedules are directly obtained from abstract syntax tree. Comparing the results in Figure 8 shows that sequential models are analysed more quickly because they do not deal with concurrency.

Protocol	No. Interleaving	CM	No. Branch	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	*
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

Fig. 8. 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 [4]. The number of branches for SM and SEC models are the same. Times are in milliseconds.

However, error correction protocols are inherently sequential, so their sequential and concurrent models are very similar and produce similar results.

8 Related Work

In recent years there have been several approaches to the formal analysis of quantum information systems. In this section we review some of the work that is most relevant to this paper.

We have already mentioned the QMC system. QMC checks properties in *Quantum Computation Tree Logic (QCTL)* [6] on models which lie within the stabilizer formalism. It can be used to check some protocols in a process-oriented style similar to that of the present paper; however, it simulates the protocols on *all* stabilizer states as inputs, not just the smaller set of stabilizer states that form a basis for the space of density matrices, and is therefore less efficient.

Our previous work [4] uses a similar approach to the present paper, but limited to sequential protocols. It therefore lacks the ability to explore, for a given protocol, different models with different degrees of concurrency.

Process calculus can also be used to analyse quantum systems. Gay and Nagarajan introduced CQP [17] based on the π-calculus; bisimulation for CQP has been developed and applied by Davidson *et al.* [10, 9]. Ying *et al.* have developed qCCS [27] based on classical CCS, and studied its theory of bisimulation [14]. These are theoretical investigations which have not yet produced tools.

Wille *et al.* [26] consider two reversible circuits and then check their equivalence with respect to a target functionality (specification). To this end, techniques based on Boolean SAT and Quantum Binary Decision Diagrams [25] have been used. However, these methods are only applicable to quantum circuits with classical inputs/outputs.

Abramsky and Coecke [2] have developed diagrammatic reasoning techniques for quantum systems, based on a category-theoretic formulation of quantum mechanics. Quantomatic [12] is a tool based on this formalism, which uses graph rewriting in order to reason about quantum systems. The interface of Quantomatic is graphical, in contrast to our tool which uses a programming language syntax. Also, our tool verifies quantum protocols in a fully automatic way, whereas Quantomatic is a semi-automatic tool which needs a considerable amount of user intervention (see [13] for an example and discussion).

9 Conclusions and Future work

We have presented a technique and a tool for the verification of concurrent quantum protocols, using equivalence checking. In our work, we require that a concurrent protocol computes a function, in the sense that, (1) each interleaving yields a deterministic input-output relation; and (2) all interleavings yield the same input-output relation. The semantics of each interleaving of a quantum protocol is defined by a *superoperator*, which is a *linear* transformation from a protocol's input space to its output space. In particular, since superoperators are linear, we can analyse the behaviour of a concurrent protocol on arbitrary inputs using a suitable stabilizer basis [16]. This enables us to reduce the problem of checking equivalence over a continuum of quantum states to a tractable problem of checking equivalence over a discrete set of states. We have presented results comparing the execution times of three approaches to verification: (1) analysis of concurrent models; (2) analysis of sequentialised models with the concurrent equivalence checker; and (3) analysis of sequential models with our previous sequential equivalence checker [4].

In this work we are not able to analyse algorithms with non-stabilizer elements, like Shor's and Grover's algorithm. Also, continuously running protocols with input/output at intermediate points, need a more general notion of equivalence such as bisimulation. And of course our tool cannot be used directly when the correctness of protocols can not be specified as an equivalence, such as the security property of QKD. Nevertheless, our tool has been successfully applied to a range of examples as shown in Figure 8.

One area for future work is extending the scope of our tool to go beyond the stabilizer formalism. There are number of cases studied in [1] which involve non-stabilizer states and operations. We should be able to extend our techniques in a straightforward way, if the number of non-stabilizer operations is limited.

A more comprehensive modelling language *e.g.* implementing functional features, as well as improvement of the classical aspects of our tool, is highly desirable. Another area for research is extending our equivalence checking technique to other quantum modelling languages, such as CQP or qCCS. It would be interesting to investigate whether the bisimulation relations in [10] or [14] can be automated. In this paper, we have only considered an interleaving model of concurrency. Of course, one could consider other models of concurrency, for example *true concurrency*, and see whether it can be characterized by superoperators.

Finally, it will be interesting to develop a stabilizer based technique for analysis of security protocols such as QKD.

References

1. S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70:052328, 2004.
2. 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.
3. E. Ardeshir-Larijani. Quantum equivalence checker. `http://go.warwick.ac.uk/eardeshir/qec`, 2013.
4. E. Ardeshir-Larijani, S. Gay, and R. Nagarajan. Equivalence checking of quantum protocols. In N. Piterman and S. 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.
5. 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.

6. 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.
7. 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.
8. A. R. Calderbank and P. W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, Aug 1996.
9. T. A. S. Davidson, S. J. Gay, R. Nagarajan, and I. V. Puthoor. Analysis of a quantum error correcting code using quantum process calculus. *EPTCS*, 95:67–80, 2012.
10. T. A. S. Davidson. *Formal Verification Techniques Using Quantum Process Calculus.* PhD thesis, University of Warwick, 2011.
11. H. de Riedmatten, I. Marcikic, W. Tittel, H. Zbinden, D. Collins, and N. Gisin. Long distance quantum teleportation in a quantum relay configuration. *Physical Review Letters*, 92(4):047904, 2004.
12. 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.
13. R. Duncan and M. Lucas. Verifying the Steane code with quantomatic. *arXiv:1306.4532*, 2013.
14. 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*, pages 523–534. ACM, 2011.
15. E. Gagnon. SableCC, an object-oriented compiler framework. Master's thesis, School of Computer Science, McGill University, 1998.
16. S. J. Gay. Stabilizer states as a basis for density matrices. *arXiv:1112.2156*, 2011.
17. S. J. Gay and R. Nagarajan. Communicating Quantum Processes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages*, pages 145–157. ACM, 2005.
18. D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173(2):143–161, 2002.
19. M. Hillery, V. Bužek, and A. Berthiaume. Quantum secret sharing. *Phys. Rev. A*, 59:1829–1834, Mar 1999.
20. D. Mayers. Unconditional Security in Quantum Cryptography. *Journal of the ACM*, 48(3):351–406, 2001.
21. R. Milner. *Communication and concurrency.* Prentice Hall, 1989.
22. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.
23. P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.
24. P. W. Shor. Fault-tolerant quantum computation. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, Washington, DC, USA, 1996. IEEE Computer Society.
25. G. F. Viamontes, I. L. Markov, and J. P. Hayes. *Quantum Circuit Simulation.* Springer, 2009.
26. R. Wille, D. Grosse, D. Miller, and R. Drechsler. Equivalence checking of reversible circuits. In *39th International Symposium on Multiple-Valued Logic*, pages 324–330, 2009.
27. M. Ying, Y. Feng, R. Duan, and Z. Ji. An algebra of quantum processes. *ACM Trans. Comput. Logic*, 10(3):19:1–19:36, April 2009.