Equivalence Checking of Quantum Protocols

Ebrahim Ardeshir-Larijani[1]*, 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. Quantum Information Processing (QIP) is an emerging area at the intersection of physics and computer science. It aims to establish the principles of communication and computation for systems based on the theory of quantum mechanics. Interesting QIP protocols such as quantum error correction, teleportation, and blind quantum computation have already been realised in the laboratory and are now in the realm of mainstream industrial applications. The complexity of these protocols, along with possible inaccuracies in implementation, demands systematic and formal analysis. In this paper, we present a new technique and a tool, with a high-level interface, for verification of quantum protocols using equivalence checking. Previous work by Gay, Nagarajan and Papanikolaou used model-checking to verify quantum protocols represented in the stabilizer formalism, a restricted model which can be simulated efficiently on classical computers. Here, we are able to go beyond stabilizer states and verify protocols efficiently on all input states.

Keywords: quantum protocols, equivalence checking, model checking, stabilizers

1 Introduction

With the emergence of quantum computation and quantum information processing, there is now a need for high level understanding and techniques in the design and analysis of quantum protocols. To this end, we are pursuing a programme of applying formal methods, developed for the analysis of classical computing and communication systems, to analyse and verify quantum systems. The present paper concerns model-checking, in which the behaviour of a system (defined in a formal modelling language) is exhaustively explored in order to verify that a

* 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.

desired specification is satisfied by all possible execution paths. There are two distinct styles of model-checking. The first style is *property-oriented*, in which a specification is expressed as a logical formula, usually in temporal logic, and the truth of the formula is checked along every possible execution path. The second style is *process-oriented*, in which a specification is expressed as a simple ideal system whose correctness is self-evident, and verification consists of checking that the (model of the) implementation has exactly equivalent behaviour to the specification.

Previous work by Gay, Nagarajan and Papanikolaou [19, 24] has developed QMC, a property-oriented model-checking system for quantum protocols. The present paper explores the process-oriented approach. The main novelty is to exploit the fact that quantum operators are *linear*, in the sense of linear algebra, to reduce the number of inputs on which two quantum protocols must be executed in order to check their equivalence. Interpreting quantum protocols as linear operators on a certain vector space, we can check that two protocols denote the same operator by executing them on inputs which form a basis for the space; linearity means that their behaviour on the whole space is determined by their behaviour on a basis. We have implemented a prototype software tool which uses this idea to automatically check the equivalence of two given quantum protocols.

In addition to the usual problem of large state-spaces arising from the possible execution paths and interactions within a system, quantum model-checking presents another challenge. A quantum state on n qubits (quantum bits) is defined by a basis vector expansion involving 2^n complex coefficients, so representing a quantum state as a classical data structure appears to require exponential space. Indeed, much of the interest in quantum computing arises from the fact that in general, quantum systems cannot be efficiently simulated by classical computers. To avoid this problem, we work with the *stabilizer formalism* [1], which allows efficient classical simulation of a restricted set of quantum states and operations on them. Although not sufficient for general-purpose quantum computing, stabilizer states support many interesting quantum protocols such as teleportation [7], superdense coding [8], and quantum error correction [23, Chapter 10], as well as the essential quantum phenomenon of entanglement. The QMC system [19, 24] also uses the stabilizer formalism.

We can explain the advantages of the tool described in the present paper, in comparison with QMC, by considering the problem of verifying a quantum teleportation protocol. Quantum teleportation transfers an unknown quantum state from one physical carrier to another, by carrying out a certain sequence of operations. Its specification is that it should be equivalent to the identity operator on a single qubit. To verify teleportation with QMC, first the condition that the output state is the same as the input state is expressed in a property-oriented style. Then the protocol is executed with every one-qubit stabilizer state (there are six of them) as input. Correctness on all of these inputs is interpreted as evidence for, although not absolute proof of, correctness of the protocol on arbitrary inputs. The equivalence checker described in the present paper executes the teleportation protocol on a set of stabilizer states that form a basis for the appropriate vector space; this involves only four states, and correspondingly less computation. Moreover, by linearity, correctness on these four states guarantees that the protocol is correct for arbitrary inputs. Because QMC tests the protocol on these four states (as well as others), we can retrospectively see that QMC also guaranteed correctness, assuming that the protocol satisfies the semantic conditions that we introduce in Section 4.

The remainder of the paper is organised as follows. In Section 2, we give all the necessary preliminaries for our equivalence checking method. In Section 3, we introduce the language QPL with its syntax and a summary of its semantics. We also present some examples of quantum protocols written in QPL. In Section 4, we explain how our equivalence checker works and give details of the implementation. In Section 5, we present some results comparing our equivalence checker with the QMC system, in terms of running time. Finally, in Sections 6 and 7, related work and future research directions are discussed.

2 Technical Foundations

The unit of quantum information is a *qubit* (quantum bit). A vector space equipped with an *inner product* is called Hilbert space.[4] The *state* of a qubit is a vector in the Hilbert space and is specified by $|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ are amplitudes and $|\alpha|^2 + |\beta|^2 = 1$. We use Dirac's notation to denote unit vectors $|0\rangle$ and $|1\rangle$. States are transformed by unitary linear operators in Hilbert space. An interesting quantum operation is *measurement*, which is not unitary. The outcome of measuring the above state $|\Psi\rangle$ is the classical bit 0 with probability $|\alpha|^2$ or 1 with probability $|\beta|^2$. Moreover measurement changes the state of the qubit permanently to $|0\rangle$ or $|1\rangle$. The quantum circuit model is similar to the classical circuit model, except that there are quantum gates acting on qubits. Quantum circuits are usually described in the following way: each line (wire) represents a qubit and boxes represent quantum gates and also measurement. There are also two-qubit gates, which act on two qubits at the same time, for example, *controlled* gates. Each controlled gate consist of a control qubit (depicted by a point) and target qubit (depicted by a circle). If the value of the control qubit is one, then the corresponding unitary gate is applied. In the quantum circuit, control qubit and target qubit are connected by a vertical line. After measurement, the outcome is classical and this is denoted by a double line. For example, *quantum teleportation* [7] is a protocol which transmits a quantum state from a sender to a receiver using a classical channel and an *entangled* pair (i.e. the qubit is not physically transmitted but it is teleported). The circuit which implements teleportation is illustrated in Figure 1. This circuit uses X (not), Z (phase shift), H (Hadamard) and controlled-not (controlled-X) gates:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

[4] There are other conditions, which will not concern us.

Prior to the execution of teleportation protocol, two parties (we call them Alice and Bob) share an entangled pair which can be prepared by applying a Hadamard and a controlled-not gate.

The protocol proceeds as follows: Alice combines the qubit to be teleported with her part of the entangled pair, by applying a controlled-not gate followed by a Hadamard gate. Then she measures the qubits in her possession. If the outcome is one, then she applies Z or X to the corresponding qubit (see double lines in Figure 1). Now Bob's part of the entangled pair ends up in the same state as Alice, which demonstrates that a qubit has been successfully transferred from Alice to Bob.

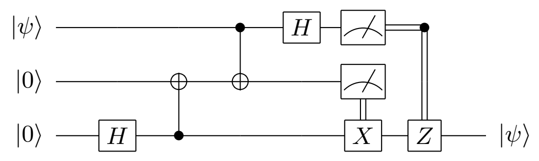

Fig. 1. Teleportation

Stabilizer states are a small but useful subset of quantum states which can be represented in polynomial space [1]. The main idea of the stabilizer formalism is to represent a quantum state $|\phi\rangle$, not by 2^n complex amplitudes (here n is the dimension of $|\phi\rangle$) but by a *stabilizer group*, $Stab(|\phi\rangle)$. This group can be represented by its set of generators G_i, $i \leq n$ such that $G_i |\phi\rangle = |\phi\rangle$. For example the two qubit entangled state $|\phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is represented by $\{X \otimes X, Z \otimes Z\}$:

$$X \otimes X |\phi\rangle = |\phi\rangle$$
$$Z \otimes Z |\phi\rangle = |\phi\rangle$$

More importantly the effects of a certain class of operations and *measurement* on the stabilizer states can be described by a polynomial time algorithm:

Theorem 1. *(Gottesman-Knill, [23, 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 notion of *density operator* was introduced by Von Neumann and in fact the whole quantum mechanics can be rewritten in the language of density operators.

Let $\{(|\phi_i\rangle, p_i)\}$ denote an ensemble of quantum states (the system is in the state $|\phi_i\rangle$ with probability $|p_i\rangle$). The density operator ρ can be defined by

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

Here $|\phi_i\rangle \langle\phi_i|$ denotes the outer product of $|\phi_i\rangle$. When $p_i = 1$ we say the state is pure; otherwise the state is mixed. It is useful to note that the density operator ρ is a Hermitian operator: $\rho^\dagger = \rho$ (here $\dagger$ denotes transpose of the complex conjugate) and has two properties. It is positive (for any state $|\varphi\rangle$: $\langle\varphi| \rho |\varphi\rangle \geq 0$) and has a trace condition $Tr(\rho) = 1$. Linear transformations of the form $F : \rho \to \rho'$ where ρ and ρ' are density operators, are called *superoperators*. Suppose we have two systems A and B. Let $|a_1\rangle$, $|a_2\rangle$, $|b_1\rangle$ and $|b_2\rangle$ be any vectors in the state space of A and B. The *partial trace* [23, page 105] of the composite system AB is defined by:

$$Tr_B(|a_1\rangle \langle a_2| \otimes |b_1\rangle \langle b_2|) \equiv |a_1\rangle \langle a_2|\, Tr(|b_1\rangle \langle b_2|)$$

The mathematical interpretation of a quantum information processing system, which is given some quantum input and produces some quantum output, is a linear operator. This is very specific to quantum systems and has no analogue in classical computing. In particular, quantum systems with no measurement can be abstracted by unitary operators, which are linear. In order to check a property of such system, it is sufficient to examine the standard basis of Hilbert space (e.g. for a system operating on one qubit, we check only $|0\rangle$ and $|1\rangle$ and because of linearity we can extend our argument to any state of the general form $\alpha|0\rangle + \beta|1\rangle$). In the case where quantum systems involve measurement, the mathematical interpretation is superoperators, instead of unitaries. Superoperators operate on the space of density matrices, with dimension 2^{2n} for n qubits. Then the behaviour of a quantum system with measurement can be examined by a basis of the space of density matrices. However, in general, for verification of such quantum systems (especially using model-checking), it is impossible to specify and manipulate quantum states on classical computers because there is a continuum of quantum states. Therefore, we use stabilizer states which we can manipulate efficiently. The following theorem [17] finds a stabilizer basis for the space of density matrices, which we shall use later for equivalence checking.

Theorem 2. *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.*

Notation 1 *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 1. *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 2. *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 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 1 and the closure properties. The same argument applies to $|x\rangle + i\,|y\rangle$, using iGHZ_n. □

Proof of Theorem 2. This is the space of Hermitian matrices and its obvious basis is the union of

$$\{|x\rangle\,\langle x| \mid 0 \leqslant x < 2^n\} \tag{1}$$

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

$$\{-i\,|x\rangle\,\langle y| + i\,|y\rangle\,\langle x| \mid 0 \leqslant x < y < 2^n\}. \tag{3}$$

Now consider the union of

$$\{|x\rangle\,\langle x| \mid 0 \leqslant x < 2^n\} \tag{4}$$

$$\{(|x\rangle + |y\rangle)(\langle x| + \langle y|) \mid 0 \leqslant x < y < 2^n\} \tag{5}$$

$$\{(|x\rangle + i\,|y\rangle)(\langle x| - i\,\langle y|) \mid 0 \leqslant x < y < 2^n\}. \tag{6}$$

This is also a set of $(2^n)^2$ states, and it spans the space because we can obtain states of forms (2) and (3) by subtracting states of form (4) from those of forms (5) and (6). Therefore it is a basis, and by Lemma 2 it consists of stabilizer states. □

Equality test: States in the stabilizer formalism are represented by sets of Pauli generators. This representation is not unique since different sets of generators can produce the same state. Therefore a direct comparison of generators cannot establish the equality of two stabilizer states. To check equality of two stabilizer states, which we will require later, we check the linear independence of their corresponding set of generators. If two sets of generators are independent, then indeed they are not equal; otherwise they are equal. Let $|\phi\rangle$ and $|\psi\rangle$ be stabilizer states and $\mathit{Stab}(|\phi\rangle)$, $\mathit{Stab}(|\psi\rangle)$ be their stabilizer groups. It is easy to show that:

Lemma 3.

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

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

Proof From Lemma 3 we have $Stab(|\phi\rangle) = Stab(|\psi\rangle) \Longrightarrow |\phi\rangle = |\psi\rangle$. So it suffices to show $Stab(|\phi\rangle) \subseteq Stab(|\psi\rangle)$ and $Stab(|\phi\rangle) \supseteq Stab(|\psi\rangle)$. If generators of the group $Stab(|\phi\rangle)$ are linearly dependent on the generators of $Stab(|\psi\rangle$ then $Stab(|\phi\rangle) \subseteq Stab(|\psi\rangle)$. To check this, first we represent each $Stab(|\psi\rangle)$ then $Stab(|\phi\rangle)$ by their stabilizer array, an $m \times n$ matrix of Pauli operators, where n is the number of qubits and m is the number of generators of the stabilizer group. Now we consider elementary row operations on the stabilizer array [3]. Here we have two operations: row transpose and row multiplication. These operations do not alter the stabilizer group and hence do not change stabilizer states. In the case of row multiplication, the generators of the stabilizer are altered. Using these two operations a normal form, *Row Reduced Echelon Form (RREF)*, is introduced [3]. It is also shown in [3] that dependencies of stabilizer generators result in I rows in RREF form. We use this result in the following way: we form a combined stabilizer array consisting of generator sets of $Stab(|\psi\rangle$ then $Stab(|\phi\rangle)$ and apply the RREF algorithm on the combined array. The dependencies between generators result in I rows in the combined array. If the number of I rows in the RREF form of combined array is equal to the size of each generator set, then these two sets are dependent. Otherwise they are independent and hence produce different states. The complexity of the RREF algorithm is $O(n^3)$ [3]. $\square$

3 The Language

Many languages have already been proposed for quantum programming; for a survey see [16]. Depending on the underlying model of quantum computation, these languages are designed in different ways. In this paper, we use Selinger's *Quantum Programming Language(QPL)* [25]. This language assumes *QRAM* [20] as a realistic model of quantum computation and follows the slogan of "classical control over quantum computation". Also, QPL has a functional programming style.

In the following we give the textual and structured syntax of QPL (Figure 2). Here, we have a new type **qbit** which stands for qubits variables (for complete typing rules see [25]). Furthermore, we have *unitary operators* on qubits and *measurements*. In the case of qubits, *discard x* means deallocation of qubits which we interpret as *partial trace* of qubits in a composite system. QPL has many useful high-level features like recursive procedures, structured data types and loops. We can formalise different quantum protocols as well as quantum algorithms in QPL. For our equivalence checking, we formalise a protocol at different levels of abstraction and then we check they are equivalent. Typically, for each protocol we specify two quantum programs; one corresponding to its specification and the other to its implementation.

```
P,Q ::= newbit b|newqbit q:=0 | 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
```

Fig. 2. QPL Syntax

Example 1. **Teleportation**. We have discussed this protocol in Section 2, in the circuit model, and it is depicted in Figure 3. At the specification level, we can think of teleportation as a protocol which transfers the state of a qubit from Alice to Bob. At the implementation level, we apply different operations of the protocol on Alice's qubit and quantum resources (entangled pair) and Bob is able to recover the state of the qubit. The specification and implementation are shown in Figure 3.

Remark 1. This model of teleportation (circuit model and sequential QPL) does not show the physical separation of Alice and Bob. Extending our approach to a concurrent language with communication is a topic for future work.

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

```
program Teleportation_Implementation
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 q2:qbit
```

Fig. 3. Teleportation: Specification and Implementation

Example 2. **Bit Flip Error Correction Code** [23, p. 427]. In this protocol Alice sends her qubit to Bob over a noisy channel and the effect of noise is flipping qubits (by applying the Pauli gate X to random qubits). The implementation of this protocol has three phases: encoding the qubit, sending it over a noisy channel (applying random X) and recovery. The specification and implementation are shown in Figure 4).

```
program Error_Correction_Specification
input  q0:qbit
output q0:qbit
```

```
program Error_Correction_Implementation
input q0:qbit
//Encoding
newqbit q1; newqbit q2;
q0q1*=CNot; q0q2*=CNot;
//Random noise: 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 { } end} else end;
measure q3 then else {measure q4 then q1*=X else q2*=X 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 end;
measure q5 then else {measure q6 then q2*=X else q0*=I end} end;
//Bob recovers Alice's qubit
q0q1*=CNot; q0q2*=CNot;
output q0:qbit
```

Fig. 4. Error Correction: Specification and Implementation

The significance of QPL lies in its semantics [25]. It admits a denotational semantics in terms of superoperators. This means that the input and output of quantum programs can be in mixed states and the effect of executing a quantum program can be elegantly described by a superoperator, operating on the density matrices of the input.

Let $D_n = \{A \in \mathbb{C}^{n\times n} | \text{A is positive hermitian and Tr(A)=1}\}$. The Löwner partial order for D_n is defined in the following way: if $A \sqsubseteq B$ then $B - A$ is positive. The domain of denotations for QPL, $(D_n, \sqsubseteq)$, is a poset and a complete partial order (cpo) [25]. Now, the formal semantics of a program in QPL can be defined by a superoperator F of the form: $F : (D_n, \sqsubseteq) \rightarrow (D_n, \sqsubseteq)$. For more details about the formal semantics, as well as a static type system for QPL and several examples, see [25].

Remark 2. In the usual definition of density matrices we have that the trace is equal to 1. But in [25] this has been relaxed to $\leqslant 1$ in order to handle infinite loops. However, in this version of equivalence checker we only deal with protocols without loops, so the original definition is sufficient here.

4 The Equivalence Checker

Given QPL programs P_1 and P_2, representing the specification and implementation of a protocol, we want to check their equivalence: $P_1 \cong P_2$. By definition, this means $S_1 = S_2$, where S_i is the superoperator denoted by P_i.

$S_1 = S_2$ means $\forall \rho . S_1(\rho) = S_2(\rho)$, where ρ ranges over all (mixed) quantum states in the input space. By linearity, this is equivalent to $\forall \rho \in \mathfrak{B} . S_1(\rho) = S_2(\rho)$, where $\mathfrak{B}$ is a basis for the input space (and we choose a basis consisting of stabilizer states).

Because a QPL program may contain measurement operators, and quantum measurements have probabilistic results in general, executing P_i on an input ρ leads to a number of possible paths, ranged over by j, and the output is a weighted sum of the final state of execution along each path:

$$S_i(\rho) = \sum_{ij} p_{ij} \left|\varphi_{ij}\right\rangle \left\langle\varphi_{ij}\right|$$

where the p_{ij} are probabilities.

To avoid explicitly representing and computing these weighted sums, we require (and check) that P_1 and P_2 define deterministic functions. This means that for each input ρ we compute the output state $S_i(\rho)^{(j)}$ for each branch j, and check that they are all equal. If they are all equal then we write $S_i(\rho)$ for the common value.

What our equivalence checker outputs, given P_1 and P_2, is the value of the following (informal) expression:

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

Let $\mathit{paths}(P, s)$ denote the set of possible paths, indexed by integers from 1 upwards, when executing program P on input state s. Let $\mathit{StabSim}(P, s, j)$ denote the final state produced by the stabilizer simulation algorithm as in [1], starting with input state s and executing path j of program P. Let $EQ_S(v, w)$ be the equality test algorithm from Section 2. Then the above procedure corresponds to the algorithm in Figure 5.

Remark 3. The overall complexity of the above algorithm is $O(2^{2n} \mathit{poly}(m+n))$, where n is the number of input qubits and m is the number of qubits inside the programs (i.e those created by **newqbit**).

We have implemented our equivalence checker in Java. The compiler for the specification language (QPL) is produced using SableCC [15]. We used a Java implementation of Aaronson-Gottesman's algorithm for interpreting QPL in the stabilizer formalism [24]. The main components of our tool are the following:

– QPL parser (by specifying QPL grammer for SableCC).
– QPL Interpreter/Simulator (using stabilizer array algorithms and their implementation [1, 24]).
– Basis Generator (Based on Theorem 2, generates all basis states constructively)
– Equality Test for States (Based on Proposition 1).
– Quantum Measurement Scheduler (to instruct the interpreter to explore all execution paths, arising from quantum measurements).

Remark 4. The result of our equivalence checker is whether a protocol satisfies its specification on all inputs. Therefore, it stands as a proof of correctness of the protocol.

```
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 $j \in paths(P_i, v) - \{1\}$ do
      if $\neg EQ_S(StabSim(P_i, v, j), |\phi_i^v\rangle)$ then
        return $P_i$ non-deterministic
      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. 5. Algorithm for checking equivalence of QPL programs.

5 Results

We now present some initial experimental results, comparing our equivalence checking technique with the QMC model-checking system. We performed the experiments on a 2.1 GHz Intel Dual Core machine with 3.7 GB RAM, running Windows.

The main results use the examples from Section 3: teleportation (Figure 3) and error-correction (Figure 4). The timings are shown in Figure 6. In both cases

Protocol	equivalence checking (this paper)	QMC [19, 24]
Teleportation	21	75
Quantum error correction	63	72

Fig. 6. Running times in milliseconds for equivalence checking and model-checking (QMC) quantum protocols

our equivalence checker is faster, although the improvement is much smaller for the error-correction example than for the teleportation example.

To observe the effect of non-determinism in quantum systems, consider the simple QPL program in Figure 7, which simulates a coin-toss by using the random results of measuring a quantum superposition. If the input state is such

```
program Simple-Coin-Toss_Specification
input  q0:qbit
output q0:qbit
```

```
program Simple-Coin-Toss_Implementation
input q0:qbit
//Applying H to q0 creates a superposition in some cases
q0*=H;
measure q0 then q0*=I else q0*=I end
output q0:qbit
```

Fig. 7. Simple-Coin-toss: Implementation

that applying H produces a superposition state, then the measurement has two possible outcomes which occur with equal probability. The output of this program is therefore not a deterministic function of its input, and so it is rejected by our equivalence checker, independently of the specification program. Detecting non-determinism of this example takes 14ms.

6 Related work

The most closely related work is the QMC system [19, 24], with which we compared our equivalence checker in Section 5. Both QMC and our equivalence checker are based on the stablizer formalism. There are two main differences. First, QMC uses a more general modelling language which supports concurrency and synchronous communication on channels. Extending our system to a concurrent language is a topic for future work. Second, QMC is property-oriented, using *Exogenous Quantum Propositional Logic (EQPL)* [22] and its temporal extension *Quantum Computation Tree Logic (QCTL)* [4] to express specifications.

However, the existing applications of QMC have not used the full power of these logics.

Recently, Feng *et al.* [13] have studied model checking of quantum systems using quantum Markov chains. In their setting, a transition system is determined by set of states consisting of density matrices and transitions in terms of superoperators. They considered model checking of an extension of CTL to the quantum case, using quantum Markov chains. In particular, in the presence of maximal entanglement, they need to compute *accumulated superoperators* [13] from Markov chains. Their paper establishes the foundations for an approach to quantum model checking but we are not aware that they have implemented it.

An alternative style of modelling language is given by quantum process calculus, which has been developed by Gay and Nagarajan [18] (CQP) and Ying *et al.* [27] (qCCS). Bisimulation-based equivalences have been studied by Davidson [10] for CQP and by Feng *et al.* [14] for qCCS. These equivalences provide a foundation for process-oriented specification and verification of quantum systems, but they have not yet been developed into tools.

For synthesis of quantum circuits, Hayes *et al.* [26] introduced *Quantum Information Decision Diagrams (QUIDD)*, which extend *Binary Decision Diagrams(BDD)* [9] to the representation and evaluation of quantum circuits. This technique has been implemented in a tool called QuIDD Pro [26] and applied to many examples. The input of QuIDD Pro is a quantum circuit which is then represented by a QUIDD. This is in contrast with QMC and our approach, which use higher level modelling languages amenable to programming.

Abramsky and Coecke [2] started an extensive line of research on a graphical calculus for reasoning about quantum protocols. Diagrammatic reasoning is supported by an underlying categorical semantics. By using graph rewriting techniques, this idea has been implemented in the tool Quantomatic [12]. We have not compared execution times between Quantomatic and our system, because the input formats are so different (textual vs. graphical). A detailed comparison of the Quantomatic approach and our approach would be an interesting topic for future work.

Belardinelli *et al.* [5] introduced a technique for the verification of quantum protocols using a classical model checker for multi-agent systems, MCMAS [21]. They used the framework of D'Hondt and Panangaden [11] to specify protocols with respect to epistemic properties, implemented a compiler to translate the epistemic description of protocols to the input language of MCMAS. However, their technique represents quantum states by their matrix representation, which imposes scalability restrictions, and it also does not support classical control flow in the protocols.

7 Conclusion and Future Work

We have demonstrated a new approach to the verification of quantum protocols by equivalence checking. We used the stabilizer formalism and its efficient algorithms to represent and manipulate quantum states. This enabled us to develop an equivalence checker for quantum protocols. Using Theorem 2, we were able to take the further step to prove the correctness of protocols for all inputs, not just inputs that are stabilizer states. This provides stronger results than the original conclusions from the stabilizer-based model-checking system QMC, when specifications are expressed in terms of input/output behaviour. Our implementation is also faster than QMC on the examples that we have tested.

There is an important point to make in comparison with QMC. QMC does not require a program to denote a determinstic function, so it is more general, and in cases in which the program is not a deterministic function, the fact that QMC checks it on all stabilizer states as inputs can be interpreted as evidence for correctness. By Theorem 2, in cases when the program is a deterministic function, our equivalence checker gives a guarantee of correctness for all inputs. This result can be retrospectively applied to some QMC verifications, including the examples from Section 5, and could be used to speed up QMC.

The main area for future work is the extension of our techniques to concurrent systems. The idea is to allow a system to be constructed from communicating concurrent components, but still require its overall input/output behaviour to be a deterministic function. This requires extension of the syntax of QPL to a concurrent language, and an argument that every possible interleaving gives a sequentialized system which still has a superoperator semantics and can therefore be analyzed by the same techniques that we have used in this paper. Extending our language and system in this way will support more realistic models of quantum protocols, in which the participants are represented by separate concurrent processes and communication is explicit.

Because we are working within the stabilizer formalism, we can only analyze protocols whose operations are restricted to those allowed in the stabilizer formalism (Theorem 1). There are techniques for extending the stabilizer formalism to a limited number of more general operations and states (for example, [1]) and we would like to investigate those techniques in the context of our equivalence checker. Finally, there is scope for extending the classical aspects of our modelling language and for improving the efficiency of the tool.

Acknowledgement. We would like to thank Nick Papanikolaou for useful discussions and for making the source code of QMC available.

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. 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.
4. 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.

5. F. Belardinelli, P. Gonzalez, and A. Lomuscio. Automated verification of quantum protocols using MCMAS. *EPTCS 85 (Proc. QAPL)*, pages 48–62, 2012.
6. 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, 1984.
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. 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.
9. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677 –691, 1986.
10. T. A. S. Davidson. *Formal Verification Techniques Using Quantum Process Calculus*. PhD thesis, University of Warwick, 2011.
11. E. D'Hondt and P. Panangaden. Reasoning about quantum knowledge. *LNCS 3821 (Proc. FSTTCS)*, pages 553–564, 2005.
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. Y. Feng, N. Yu, and M. Ying. Model checking quantum Markov chains. *arXiv:1205.2187*, 2012.
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. Quantum programming languages: survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006.
17. S. J. Gay. Stabilizer states as a basis for density matrices. *arXiv:1112.2156*, 2011.
18. 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.
19. S. J. Gay, R. Nagarajan, and N. Papanikolaou. QMC: A model checker for quantum systems. *LNCS 5123 (Proc. CAV)*, pages 543–547, 2008.
20. E. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.
21. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *LNCS 5643 (Proc. CAV)*, pages 682–688, 2009.
22. P. Mateus and A. Sernadas. Weakly complete axiomatization of exogenous quantum propositional logic. *Information and Computation*, 204(5):771–794, 2006.
23. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
24. N. Papanikolaou. *Model Checking Quantum Protocols*. PhD thesis, University of Warwick, 2009.
25. P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.
26. G. F. Viamontes, I. L. Markov, and J. P. Hayes. *Quantum Circuit Simulation*. Springer, 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.