Model Checking for Communicating Quantum Processes

TIM DAVIDSON[1⋆], SIMON J. GAY[2†], HYNEK MLNAŘÍK[3‡], RAJAGOPAL NAGARAJAN[1¶], NICK PAPANIKOLAOU[4§]

[1] *Department of Computer Science, University of Warwick, UK*
[2] *Department of Computing Science, University of Glasgow, UK*
[3] *Warwick Institute for Financial Computing, University of Warwick, UK*
[4] *International Digital Laboratory, WMG, University of Warwick, UK*

Received 31 August 2010; In final form 31 August 2010

Quantum communication is a rapidly growing area of research and development. Quantum cryptography has already been implemented for secure communication, and commercial solutions are available. The application of formal methods to classical computing and communication systems has been very successful, and is widely used by industry. We expect similar benefits for the verification of quantum systems. Communicating Quantum Processes (CQP) is a process calculus based on the π-calculus with the inclusion of primitives for quantum information. Process calculi provide an algebraic approach to system specification and behavioural analysis. The Quantum Model Checker (QMC) is a tool for the automated verification of system correctness. Through an exhaustive search of the possible executions, QMC can check that correctness properties expressed using temporal logic formulae are satisfied. In this paper we describe our approach to the verification of quantum systems using a combination of process calculus and model checking. We also define a

⋆ email: tim@dcs.warwick.ac.uk
† email: simon@dcs.gla.ac.uk
‡ email: hmlnarik@post.cz
¶ email: biju@dcs.warwick.ac.uk
§ email: N.Papanikolaou@warwick.ac.uk

formal translation from CQP to the modelling language used by QMC and prove that this preserves the semantics of all supported CQP processes.

Key words: Quantum computing, quantum communication, process calculus, model-checking, translation, semantics

1 INTRODUCTION

Quantum computing [23, 16] offers the prospect of a radically new paradigm for information processing which takes advantage of the physical phenomena occurring on the atomic scale, including the superposition of states, entanglement and non–locality, and probabilistic measurement. Widespread interest in the field is partly due to the discovery of an efficient quantum algorithm for prime factoring [29], which, if implemented on a practical quantum computer, poses a potential threat to the security of well–known cryptosystems. Quantum computers are still in the experimental stage and are a long way from entering mainstream computing. On the other hand, proposals for quantum cryptography open up the possibility for secure communications [3, 6] even against a quantum computer, and commercial implementations already exist.

Techniques from the classical computer science field known as *formal methods* [28] have been very successful in analysing classical communication protocols, including security protocols. The characteristics of formal methods are the use of *formal languages*, with precisely-defined semantics, to model systems and their behaviour combined with the use of *formal reasoning systems*, often supported by automated software tools, to verify that systems satisfy their specifications. An important principle is *compositional analysis*, meaning analysis of a system by combining the results of analysing separate subsystems.

Experience in classical computing science has shown that it is extremely valuable to be able to analyse a model which is as close as possible to an implemented system, and the results complement mathematical proofs about an idealized protocol. We expect this complementary view to be valuable for quantum systems too. For example, there are several published proofs of correctness of the well-known BB84 quantum key distribution protocol (eg. [21]); we have no reason to doubt their validity, but we argue that they do not tell the whole story about correctness of an implemented quantum cryptosystem. An implemented system contains a significant amount of classical

computation and communication in addition to the core quantum protocol, and the correctness of the program code for the complete implementation is not directly addressed by the mathematical proof. By contrast, the approach of formal methods is to construct a formal model of the whole system, write a logical specification of correctness, and then, in an ideal situation, use an automated software tool to verify the specification or discover a fault. The emphasis of formal methods is on the development of generic theories and tools that can be applied uniformly to a range of specific systems.

Process calculus is a formal language for describing the behaviour of distributed communicating systems. It is a well-developed subfield of formal methods, and within the last few years several researchers, including ourselves, have developed quantum versions of process calculus [13, 18, 9, 31].

The key feature of *quantum* process calculus is the ability to represent quantum information and operations so that the quantum–mechanical laws are obeyed. Quantum information differs from classical information, for example, in the ability of quantum bits (*qubits*) to be in a *superposition* of many basis states, and to be *entangled* with other qubits. Other interesting properties include the *no-cloning principle* and *probabilistic measurement*. Measuring a quantum state — the equivalent of reading a classical bit — will always produce a probabilistic result based on the initial superposition, collapsing the initial state while doing so. Measurement is therefore *destructive* and, unlike other quantum operations, it is not reversible. A quantum state can therefore contain more information than can be directly extracted (this is indeed one of the most powerful features of a quantum state), but it makes it impossible to copy an unknown quantum state [30].

Process calculi provide a framework in which to describe a process and consider various properties such as equivalence from an equational point of view. While this may help to verify some properties of a system, it is necessary to use automated techniques for the verification of more complex properties. *Model–checking* [5] is an automated technique for formal verification and works by performing an exhaustive search on the state space of the system in question. The exhaustive nature of model–checking is intended to locate unexpected behaviours of a system which may easily go unnoticed by a human tester.

Classical model checking tools are not adequate for the representation of quantum information, although it is possible to model simple systems partially. Examples of early attempts of model checking quantum systems using existing tools are the modelling of BB84 [3] using CCS and the CWB-NC tool by Nagarajan and Gay [22], and the use of the PRISM probabilistic model

checker [17] by Gay et al. [10] to check a selection of quantum protocols including quantum teleportation [4] and quantum error correction.

Since model checking searches the complete state space of a system, it is computationally expensive, and efficient search algorithms are very important. Since arbitrary quantum computations cannot be simulated efficiently on a classical computer, the possibility of efficient model checking for general quantum systems is precluded from the outset. However, there exists a restricted class of quantum systems, namely those expressible within the so–called *stabilizer formalism*, for which polynomial–time simulation is possible. The class of stabilizer circuits captures a significant class of practical quantum protocols, but it falls short of the full power of quantum computation needed to implement quantum algorithms.

The Quantum Model Checker (QMC) [10, 14, 25] was built to take advantage of the efficient representation and simulation algorithms offered by the stabilizer formalism, and it has been tested with a number of small but practical case studies. However, the stabilizer formalism is too restrictive in the long term, since the verification of larger systems and complex protocols with security requirements requires support for arbitrary quantum operators.

Combining process calculus (for the high–level description of a system) with model–checking algorithms and tools (for verification of the same system's properties) is a popular approach to the analysis of systems. We believe that there are numerous benefits in using the CQP formalism for the description of quantum protocols and complex systems, due to its expressive power; this paper presents a method that would enable the translation of such descriptions to a form usable by QMC. The merits of our approach lie in the combination of two techniques which have proven successful in their own right; the translation presented here is likely to be of much practical use.

Our interest is in the analysis of protocols and implemented systems for quantum communication and quantum cryptography. In this paper we describe a translation from the high–level specification language Communicating Quantum Processes (CQP) [12] to a form that is usable by the Quantum Model Checker (QMC) [14]. We prove that this translation preserves the semantics of CQP processes, therefore establishing it as a practical verification technique.

2 BACKGROUND

In this section we review the basic concepts of quantum information. We refer the reader to standard texts for further details (e.g. [23] and [16]). An

account aimed at Computer Scientists is given in [27]. In this paper we are interested in communication systems which may involve both classical computing devices and quantum components. Due to the limitations of QMC, we are primarily interested in states and operations which arise in the so-called stabilizer formalism; according to the Gottesman-Knill theorem [15], quantum circuits in this formalism are simulable in polynomial time on a classical computer.

In quantum information, the most basic system of interest is the *qubit,* or quantum bit, which is represented by a particle with two degrees of freedom (such as a polarised photon, or a spin-$\frac{1}{2}$ particle). The state of a qubit is represented by a unit vector in a 2-dimensional complex Hilbert space $\mathcal{H}_2$. The *standard basis*, conventionally written in Dirac's *braket* notation, is $\{|0\rangle, |1\rangle\}$ where $|0\rangle = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ and $|1\rangle = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. The states $|0\rangle$ and $|1\rangle$ are often associated with the two states of a classical bit, however in contrast to a bit, the general state of a qubit is a superposition of basis states

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$; equivalently $|\psi\rangle = \left[\begin{smallmatrix}\alpha\\\beta\end{smallmatrix}\right]$. Multiple qubit systems are represented using the *tensor product* (as opposed to the cartesian product for classical systems). For example, a 2-qubit system has basis states $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ (in $\mathcal{H}_4 = \mathcal{H}_2 \otimes \mathcal{H}_2$) where $|00\rangle$ is an abbreviation of the tensor product $|0\rangle \otimes |0\rangle$. The general state of a 2-qubit system is then $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ where $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. Such a system is *separable* if it can be expressed as the tensor product of two single qubit systems, otherwise the two systems are said to be *entangled*.

The evolution of a quantum system is expressed by unitary linear operators. An operator U is unitary if $UU^\dagger = U^\dagger U = I$, where $U^\dagger$ is the conjugate-transpose of U and I is the identity operator. Examples of single qubit operators include the Pauli gates X, Y, Z, the phase gate P and the Hadamard operator H. These are defined by the following matrices

$$X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} \quad Y = i\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix} \quad Z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$$

$$P = \begin{bmatrix}1 & 0\\0 & i\end{bmatrix} \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}.$$

The Hadamard operator is often used to create and destroy superpositions, for example $H(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = \frac{1}{2}(|0\rangle + |1\rangle) + \frac{1}{2}(|0\rangle - |1\rangle) = |0\rangle$. The standard

2-qubit operator is the controlled-not gate C, which has the matrix

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

On basis states, this has the effect of applying the X gate to the second qubit based on the state of the first qubit; it extends to superpositions by linearity.

The actual state of a quantum system is unknown until it is measured; however, the act of measurement is destructive and collapses the system probabilistically to a basis state. Measuring a qubit in state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with respect to the standard basis will collapse the qubit into either state $|0\rangle$ (with probability $|\alpha|^2$) or state $|1\rangle$ (with probability $|\beta|^2$). The destructive nature of measurement is particularly significant when it comes to entangled states because a measurement of one qubit can affect the states of other qubits. For example, measuring the first qubit of an entangled system in the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ results in the state $|00\rangle$ with probability $\frac{1}{2}$ and state $|11\rangle$ with probability $\frac{1}{2}$. This has the effect of fixing the state of the second qubit even though it was not measured.

The stabilizer formalism (originally due to Gottesman [15]) is concerned with the quantum states that arise in computations involving only the operations mentioned above. In order to perform arbitrary quantum computations, a universal set of gates would be required; however, the Clifford gates $\{H, C, P\}$ do not constitute a universal set. What is distinctive about this set of gates is that the operations may be simulated efficiently on a classical computer (this is known as the Gottesman–Knill theorem; see [15, 23]).

3 THE NATURE AND STRUCTURE OF QUANTUM PROTOCOLS

A *quantum protocol* is loosely defined as a set of operations and measurements on a global quantum state, which may be distributed amongst a number of users; measurement outcomes may be communicated classically between users. Often quantum protocols are expressed schematically using the quantum circuit model, however this model only allows one to describe the computational parts of a protocol. For the sake of illustration, we consider the quantum teleportation protocol [4], shown using standard quantum circuit notation in Figure 1.

Teleportation is a process which allows two users who share an entangled pair of qubits, to exchange an unknown qubit state by communicating only

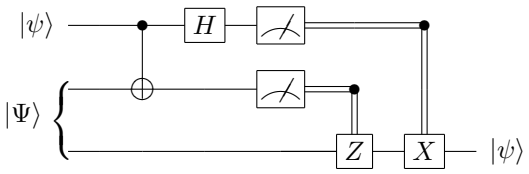

FIGURE 1
Quantum circuit diagram for the teleportation protocol.

two classical values, namely, the outcomes of two measurements. The protocol is described below.

Alice wishes to send Bob the qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$; we shall call this qubit '1'. To begin with, a pair of qubits (labelled '2' and '3') are placed in the quantum state

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

Qubit 2 is given to Alice and qubit 3 is given to Bob. Since $|\Psi\rangle$ is an *entangled state*, any measurement performed on the one qubit will affect the state of the other qubit irreversibly. For instance, if Alice were to measure qubit 2 with respect to the standard basis, she would collapse the state of qubits 2 and 3 to either $|00\rangle$ or $|11\rangle$ at random with equal probability $\left(\frac{1}{\sqrt{2}}\right)^2 = 0.5$. So we have a system of three qubits described by the overall quantum state $|\psi\rangle \otimes |\Psi\rangle$.

Alice starts by applying the controlled-not gate $C_{1,2}$ (to qubits 1 and 2) and then the Hadamard H_1 (to qubit 1). She subsequently measures qubits 1 and 2 with respect to the standard basis, and records the outcomes of her measurements M_1 and M_2. She sends these two classical values to Bob (these classical data are represented by the double lines in the figure), who then applies the operator $X^{M_1} \cdot Z^{M_2}$. At the end of the protocol, qubit 3 will be in the state $|\psi\rangle$, thus achieving Alice's goal of sending the original state of qubit 1.

This example shows some typical characteristics of a quantum protocol. This particular protocol is quite simple, as it involves only two users and just a handful of operations and measurements. As such, it is amenable to analysis by hand and can easily be shown to be correct. However, large systems (such as quantum key distribution networks [7]) would typically include the telepor-

$$
\begin{array}{rcl}
T & ::= & \mathsf{Int} \mid \mathsf{Qbit} \mid \widehat{\ }[\tilde{T}] \mid \mathsf{Op}(1) \mid \mathsf{Op}(2) \mid \cdots \\
v & ::= & \mathsf{0} \mid \mathsf{1} \mid \cdots \mid q \mid c \mid \mathsf{H} \mid \cdots \\
e & ::= & v \mid x \mid \mathsf{measure}\,\tilde{e} \mid \tilde{e} *\!= e^e \mid e + e \\
P & ::= & \mathbf{0} \mid (P \mid P) \mid P + P \mid e?[\tilde{x} : \tilde{T}].P \mid e![\tilde{e}].P \mid \{e\}.P \mid \\
 & & (\mathsf{new}\ x{:}\widehat{\ }[T])P \mid (\mathsf{qbit}\ x)P
\end{array}
$$

FIGURE 2
CQP process syntax.

tation protocol as a primitive and would combine it with other computations and transmissions, both quantum and classical.

The quantum circuit formalism is able to describe the quantum operations involved in a computation, but cannot represent the structural aspects (the components and interactions) of a protocol. Various programming and specification formalisms have been proposed in order to address the shortcomings of the quantum circuit model when describing quantum protocols. These include quantum programming languages and quantum process calculi (see [11] for a survey).

4 SPECIFYING QUANTUM PROTOCOLS USING PROCESS CALCULUS

Process calculus is used to describe the interactions (synchronizations and communications) between the components, or *processes*, of a system. Generally, process calculi are able to express sequential and parallel execution, and may often feature constructs for non-deterministic choice, conditional choice, repetition and more. Quantum process calculi, including QPAlg [18, 19], qCCS [9, 31, 32] and CQP [12, 13], also have the ability to represent quantum information as a condition on the execution. In this section we introduce the quantum process calculus Communicating Quantum Processes (CQP).

The syntax of CQP is defined by the grammar in Figure 2. This consists of types T, values v, expressions e and processes P. We use the notation $\tilde{e} = e_1, \ldots, e_n$, and write $|\tilde{e}|$ for the length of a tuple. Types consist of integer and qubit types Int and Qbit, channel types $\widehat{\ }[\tilde{T}]$, and n-qubit operator types

$$
\begin{aligned}
\mathit{Teleport} &= (\mathsf{qbit}\ y, z)(\{z \mathrel{*}\!= \mathsf{H}\}.\{z, y \mathrel{*}\!= \mathsf{CNot}\}. \\
&\qquad\qquad (\mathsf{new}\ e{:}\widehat{\ }[\mathsf{Int}, \mathsf{Int}])(\mathit{Alice} \mid \mathit{Bob})) \\
\mathit{Alice} &= (\mathsf{qbit}\ x).\{x \mathrel{*}\!= \mathsf{H}\}.\{z, x \mathrel{*}\!= \mathsf{CNot}\}.\{z \mathrel{*}\!= \mathsf{H}\}. \\
&\qquad\qquad e![\mathsf{measure}\ z, \mathsf{measure}\ x].\mathbf{0} \\
\mathit{Bob} &= e?[r{:}\mathsf{Int}, s{:}\mathsf{Int}].\{y \mathrel{*}\!= \mathsf{Z}^r\}.\{y \mathrel{*}\!= \mathsf{X}^s\}.\mathbf{0}
\end{aligned}
$$

FIGURE 3
Quantum teleportation modelled in CQP.

$\mathsf{Op}(n)$. Values consist of literal data types $(0, 1, \dots)$ and unitary operators $(\mathsf{H}, \dots)$. Expressions consist of values v, variables $(x, y, \dots)$, measurements $(\mathsf{measure}\ \widetilde{q})$, application of unitary operators ($\widetilde{q} \mathrel{*}\!= \mathsf{U}^e$, that is, operator U^e is applied to qubits $\widetilde{q}$) and data operators (e.g. $e + e$).

Processes consist of the nil process $\mathbf{0}$ which has no execution; parallel composition $P \mid Q$ in which the executions of P and Q interleave; non-deterministic choice $P + Q$ which can behave as P or as Q; inputs $c?[\widetilde{x}{:}\widetilde{T}].P$ where values and qubit names are received via channel c and substituted for the variables $\widetilde{x}$ in P; outputs $c![\widetilde{v}].P$ in which the values v are sent on channel c and then the process behaves as P; actions $\{e\}.P$ which may, for example, involve unitary operations before continuing as P; the restriction of channel c to the process P, $(\mathsf{new}\ c{:}\widehat{\ }[\widetilde{T}])P$; and qubit declaration $(\mathsf{qbit}\ x)P$ in which a fresh qubit is prepared in the state $|0\rangle$ for use by P.

The syntax is complemented by a type system which restricts the construction of processes to meaningful terms. For example, a measurement or unitary operator should only be applied to qubits. An important result [13, Theorem 1] arising from the type system guarantees that each qubit is "owned" by only one process at any given time. In conjunction with type preservation [13, Theorem 2], this ensures that CQP processes obey the no-cloning property of quantum mechanics.

The teleportation protocol, modelled in CQP, is shown in Figure 3. This model clearly specifies the interactions between the two parties, Alice and Bob, who act in parallel with one another. The initial actions of $\mathit{Teleport}$ are the creation of the two qubits (y and z) and their preparation into the entan-

gled state $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ using the Hadamard (H) and controlled-not (CNot) operators. The process then proceeds with Alice and Bob acting in parallel and sharing the channel e.

Alice prepares a new qubit x in the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ using the Hadamard operator — this is the state that will be "teleported" to Bob. After applying the controlled-not and Hadamard operations, Alice measures the qubits x and z. Evaluating the measurements leads probabilistically to one of four possible outcomes, e.g. $e![1, 0].\mathbf{0}$. Communication in CQP is synchronous, therefore the sending by Alice and receiving by Bob occurs in one step. The values received from Alice are substituted for the variables r and s which appear as the conditions for applying the operators X and Z to Bob's qubit y.

4.1 Verification using Process Calculus

The typical approach to verification using process calculus is to define two models; one which describes the implementation of a system, and another (the specification) which describes the intended *high–level* behaviour. The correctness of the implementation with respect to the specification can then be established by using a notion of equivalence, such as *bisimulation* [26].

Bisimulation captures the ability of two systems to match the actions of each another. A relation $\mathcal{B}$ is called a *strong bisimulation* if whenever $(P, Q) \in \mathcal{B}$ then

- if $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{B}$
- if $Q \xrightarrow{\alpha} Q'$ then there exists P' such that $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{B}$

where α is an input, output or internal action.

Strong bisimulation requires each action to be matched by a corresponding action, however for verification purposes this is often too discerning. Instead, it may be desirable to treat the internal actions of a process abstractly and only require external interactions (input and output) to be matched. This concept, called *weak bisimulation*, can identify processes that are indistinguishable to an *observer* yet may have different internal behaviours.

Observational equivalence is an important concept when considering the behaviour of a subsystem in the context of a larger system. The ability to verify individual components separately and then draw conclusions about a combined system is very advantageous; for example, it reduces the overall complexity of the verification, resulting in simpler and more accurate models, and also enables components to be swapped for equivalent parts without requiring the complete system to be re-analysed. This compositional approach

to verification relies on congruence relations. A bisimulation is a congruence if it is preserved by all contexts, however not all bisimulations are congruences.

Due to the presence of entanglement and probabilistic measurement, congruence relations for quantum processes are difficult to find. A number of bisimulation relations have been defined for QPAlg [19] and qCCS [9, 31, 32], in particular, these include congruence relations for quantum processes without classical data. Congruence for general quantum processes, which is required for the analysis of many interesting quantum protocols that combine quantum and classical information, has been an open problem for a few years. As noted in Section 7, our own research programme includes the investigation of process equivalences using CQP, focussing on the factors that determine congruence.

5 MODEL–CHECKING FOR QUANTUM PROTOCOLS

In this section we introduce the Quantum Model Checker (QMC) and its use in the verification of quantum protocols. QMC is a software tool that automatically explores all possible behaviours arising from a protocol model, and enables logic properties expressed with Quantum Computation Tree Logic (QCTL) [2] to be checked over the resulting structure.

A protocol model will always consist of definitions of one or more processes; the commands performed by each of these processes must be interleaved (so as to emulate concurrent execution), and non–determinism (which occurs explicitly in selection structures (**`if`** `::` `a -> ...` `::` `b -> ...` **`fi`**) and implicitly when measurements are performed) must be resolved, producing an execution tree for the modelled system.

It is important to explain the way in which the global state is represented for QMC models. Each node in the execution tree for a given model contains a tuple $(P, \kappa, \Sigma, |\psi\rangle)$ where P and $|\psi\rangle$ in particular are respectively the process term and the overall quantum state at that point in the simulation (κ and Σ are global and local registers). The quantum state $|\psi\rangle$ is represented internally in an implicit way: rather than storing the so-called *state vector representation* of $|\psi\rangle$ (which grows exponentially in length as a function of the total number of qubits in $|\psi\rangle$), we use the *stabilizer array representation,* which is a binary representation of the set of Pauli operators that fix (or stabilize) $|\psi\rangle$. Using the stabilizer array representation, we gain significant computational benefits in terms of both space and time when simulating a given protocol, given that simulation of stabilizer circuits is performed using

$$
\begin{array}{rcl}
t & ::= & \texttt{integer} \mid \texttt{bool} \mid \texttt{real} \mid \texttt{qubit} \mid \texttt{channel of}t \\
e & ::= & n \mid r \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1/e_2 \\
 & \mid & \texttt{true} \mid \texttt{false} \mid \texttt{not}\ e \mid e_1\ \texttt{and}\ e_2 \mid e_1\ \texttt{or}\ e_2 \\
 & \mid & e_1 = e_2 \mid e_1 < e_2 \mid e_1 > e_2 \mid \texttt{meas}\ e \mid \texttt{newqubit} \\
S & ::= & e \mid x := e \mid x_1!x_2 \mid x_1?x_2 \mid \mathbf{cnot}\ x_1\ x_2 \mid \mathbf{had}\ x \\
 & \mid & \mathbf{ph}\ x \mid \mathbf{X}\ x \mid \mathbf{Y}\ x \mid \mathbf{Z}\ x \mid S_1; S_2 \\
H & ::= & :: SH \mid :: S \\
GC & ::= & \texttt{if}\ H\ \texttt{fi} \mid \texttt{do}\ H\ \texttt{od} \\
C & ::= & S; \mid GC \mid C_1\ C_2 \mid \epsilon \\
VD & ::= & \texttt{var}\ x : t; VD \mid \epsilon \\
P & ::= & \texttt{process}\ p\ VD\ \texttt{begin}\ C\ \texttt{end}\ P \mid \epsilon \\
M & ::= & \texttt{program}\ p\ VD\ \texttt{begin}\ P\ \texttt{end}
\end{array}
$$

FIGURE 4
QMC Concrete Syntax

a polynomial time algorithm [1], and the representation of the state grows polynomially with the number of total qubits. There is scope for optimising the simulation further, notably by using low-level binary operations and by packing the stabilizer array representation into 32-bit registers, as Aaronson and Gottesman chose to do in the final implementation of the CHP simulator [1].

5.1 Modelling Quantum Protocols in QMC

We have built an imperative-style concurrent specification language for the needs of the quantum model-checking tool QMC. The syntax of QMC is defined by the grammar in Figure 4. Expressions e consist of names, values, arithmetic operators, boolean operators, quantum measurement and initialisation of new qubits. Statements S consist of expressions, assignment, communication, quantum operations and sequences of statements. Options H allow a choice between one or more statements. Guarded commands GC consist of the **if** and **do** constructs. Commands C consist of statements,

guarded commands, or a sequence of commands. Variable declarations VD allow a (possibly empty) sequence of declarations. Processes P are a (possibly empty) sequence of process constructs, each containing variable declarations and commands. A program M is a single construct containing (global) variable declarations and processes.

For the purpose of this paper, we will demonstrate the syntax of this language by example. In this language the teleportation protocol (assuming we are trying to teleport the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$) may be expressed by the program in Figure 5.

In our setting, we allow for global variables (such as `e1`, `e2`), typed communication channels (such as `ch`) which are always global, and local (private) variables for each process (such as `a`,`b`,`c`,`d`,`q`). Communication is asynchronous, with executability rules restricting the way in which process interleaving is performed. For instance, the process `Bob` cannot start unless channel `ch` is filled with a value.

5.2 Specifying Properties

The properties of quantum protocols which we are interested in reasoning about are properties of the quantum state (e.g. which qubits are 'active' in a given state, which qubits are entangled with the rest of the system) over time. We are also interested in the outcomes of measurements, and the way in which the values of classical variables evolve. We have elected to use quantum computational temporal logic (QCTL) [2] for this purpose.

QCTL adds the usual temporal connectives (AX, EF, EU) of computational tree logic [8] to the propositional logic EQPL [20]. The meaning of formulae in EQPL is expressed in terms of valuations, which are truth-value assignments for the symbols $\mathsf{qb}_0, \mathsf{qb}_1, \ldots, \mathsf{qb}_n$ corresponding to each qubit in the system. For instance, the quantum state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is understood as a pair of valuations (v_1, v_2) for a 2-qubit system such that $v_1(\mathsf{qb}_0) = 0$, $v_1(\mathsf{qb}_1) = 0$, $v_2(\mathsf{qb}_0) = 1$, $v_2(\mathsf{qb}_1) = 1$.

The formulae accepted by the QMC tool for verification allow the user to reason about the state of individual qubits, and involve usual logical connectives such as negation and implication. There are two levels of formulae: classical formulae, which hold only if all valuations in a state satisfy them, and quantum formulae, which are essentially logical combinations of classical formulae. For instance, the quantum conjunction in the formula $\phi_1 \curlywedge \phi_2$ is only satisfied if both the classical formulae ϕ_1 and ϕ_2 are satisfied in the current state. A particularly distinctive type of quantum formula is of the form $[Q]$, where Q is a list of qubit variables $\mathsf{qb}_i, \mathsf{qb}_j, \ldots$; this type of formula is

```
program Teleport;
var e1,e2:qubit; ch:channel of integer;
process Alice;
var q:qubit; a,b:integer;
begin
 q := newqubit; had q;
 e1 := newqubit; e2 := newqubit;
 had e1; cnot e1 e2;
 cnot q e1; had q;
 a := meas q;
 b := meas e1;
 ch!a; ch!b;
end;
process Bob;
var c,d: integer;
begin
 ch?c; ch?d;
 if
 :: ((c=1) and (d=0)) -> X q; break;
 :: ((c=0) and (d=1)) -> Z q; break;
 :: ((c=1) and (d=1)) -> X q; Z q; break;
 :: ((c=0) and (d=0)) -> break;
 fi
end;
endprogram.
```

FIGURE 5
QMC source program for quantum teleportation.

satisfied only if the qubits listed are disentangled from all other qubits in the system. The syntax of QCTL is given below (from [2]):

$$
\begin{array}{rcl}
\text{Classical formulae: } \alpha & = & \bot \mid \mathsf{qb} \mid \alpha \Rightarrow \alpha \mid \alpha \vee \alpha \mid \alpha \wedge \alpha \\
\text{Terms: } t & = & x \mid (t+t) \mid (tt) \mid \mathrm{Re}(|\top\rangle_A) \mid \mathrm{Im}(|\top\rangle_A) \mid \int \phi \\
\text{Quantum formulae: } \gamma & = & (t \leq t) \mid \bot \mid (\alpha \sqsupset \alpha) \mid (\alpha \curlyvee \alpha) \mid (\alpha \curlywedge \alpha) \mid \\
 & & [\mathsf{qb}_i, \mathsf{qb}_j, \ldots] \\
\text{Temporal formulae: } \theta & = & \gamma \mid \theta \sqsupset \theta \mid (\mathsf{EX}\theta) \mid ([\theta \ \mathsf{EU} \ \theta]) \mid (\mathsf{AF}\theta)
\end{array}
$$

Example of Property for Verification

The requirement for the teleportation protocol is that, at the end of the protocol, no matter the measurement outcomes, the third qubit will be in the same state as the first qubit was to begin with, and this qubit will be disentangled from the rest of the system. We can express this requirement, for the case where the input is the quantum state $|0\rangle$, in the input language of QMC using the specification

```
finalstateproperty ([q2]) #/\ (!q2);
```

which corresponds to the EQPL formula $[q_2] \curlywedge (\neg q_2)$. The first part of the formula asserts that the last qubit (`q2`) is disentangled from the rest of the system, while the second part asserts that the current valuation assigns to this qubit a value of 0. The entire formula is true if both parts are true, indicated by the connective of quantum conjunction (we represent $\curlywedge$ in ASCII form by `#/\`). We can also use a temporal formula:

```
property true EU (([q2]) #/\ (!q2));
```

5.3 Verification Algorithms and Complexity

We turn now to the algorithms which QMC uses for the *verification* of QCTL formulae over protocol models.

Firstly, one should note that the logic QCTL comprises a purely propositional fragment, namely the exogenous quantum propositional logic (EQPL) proposed by Mateus and Sernadas [20]. This fragment may be interpreted, without much loss of generality, over a single quantum state $|\psi\rangle$. The general definition of the semantics of EQPL has been given [20] in terms of a so-called quantum interpretation structure, which includes not only a quantum state $|\psi\rangle$, but also a classical state ρ and a means of specifying entanglement partitions of $|\psi\rangle$. Note that in our setting we also have a global classical state, which takes the place of ρ.

Evaluating EQPL formulae over any state $|\psi\rangle$ arising from the simulation of a protocol model requires being able to determine all the valuations in that state, so that the truth value of any propositional constant (e.g.: $\mathtt{qb}_i$ where $0 \leq i \leq N$ for an N-qubit system — this constant corresponds to the state of the ith qubit in the quantum state) can be computed. What this means in more practical terms is that, in order to determine whether a given qubit has valuation *true* (1) or *false* (0) in the current state, it is necessary to extract all the basis vectors which are present in the state vector expansion of $|\psi\rangle$. The process of extracting all the basis vectors requires converting from the space–efficient stabilizer array representation to the state vector corresponding to $|\psi\rangle$, and this conversion can take up to a maximum of 2^N steps if all the 2^N basis vectors appear in $|\psi\rangle$. Even when $|\psi\rangle$ is a stabilizer state, it may contain all of the basis vectors with non-zero coefficients. Therefore in general, evaluating a classical formula requires solving a SAT problem, and of course this is **NP**-complete. This observation seems rather discouraging given that the process of verifying a state formula requires us to lose the efficient state representation which is used during simulation.

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

6 SYNTHESISING THE TWO APPROACHES: A TRANSLATION FROM CQP TO QMC

In this section we present a translation from CQP processes to QMC programs. The aim of this is to provide an approach to the formal verification of quantum protocols that combines the merits of process calculus and model checking. We prove that the translation preserves the semantics of CQP processes, thereby ensuring that a translated program has the same behavioural properties as the original process. This is important because it guarantees that the same protocol is being modelled in both languages instead of two subtly different protocols.

There are several differences between the languages that result in limitations or special treatment in the translation. The most significant, due to the inability to model universal quantum computation, is the restriction to processes that fall within the stablizer formalism. Other issues, which we discuss in more detail in the following sections, include the removal of channel mobility, translating from polyadic to monadic channels, and allowing only single qubit measurements. We also require that all variable names are unique among all CQP processes; this can be achieved by alpha conversion if necessary. As a result we are able to define all variables globally when translated to QMC without risk of collision.

In the following section we define a function $\text{TPROG}[\![]\!] : ProcCQP \longrightarrow ProgQMC$ where $ProcCQP$ and $ProgQMC$ are the sets of CQP processes and QMC programs respectively. We follow a similar approach to the translation in [24]. The translation is separated into several steps, with each consecutive step providing finer detail than the previous step.

6.1 Translation Functions

A QMC program consists of one or more named processes. Although the formal syntax of CQP does not have named processes, we choose to use them as standard (as, for example, the processes $Teleport$, $Alice$ and Bob in Figure 3) instead of introducing arbitrary names as part of the translation. In this approach, the parallel composition primitive must use process names instead of processes. For example, the process $P.(Q \mid R)$ would be represented by the named processes

$$\begin{aligned} Process1 &= P.(Process2 \mid Process3) \\ Process2 &= Q \\ Process3 &= R\,. \end{aligned}$$

We now define transcription functions from the syntactic elements of CQP to the corresponding QMC syntax. These are: $\text{TPROG}[\![]\!]$ for the program; $\text{TPROC}[\![]\!]$ for processes; $\text{TEXPR}[\![]\!]$ for expressions; $\text{TVAL}[\![]\!]$ for values; and $\text{TTYPE}[\![]\!]$ for types.

The QMC Program

The complete CQP "program", which consists of the list of named processes, must be rewritten to a QMC program. This top-level transcription is performed using the function $\text{TPROG}[\![]\!]$ define in Figure 6. The resulting QMC program is encapsulated in a **`program`** block and contains global variable dec-

$$
\begin{aligned}
&\textsc{Tprog}[\![P_1, \ldots, P_n]\!] = \\
&\qquad \texttt{program Translated; } gVars(\widetilde{P})\texttt{;} \\
&\qquad \textsc{Tprog}[\![P_1]\!] \cdots \textsc{Tprog}[\![P_n]\!] \\
&\qquad \texttt{endprogram} \\
&\textsc{Tprog}[\![\mathsf{ProcName}(x_1 : T_1, \ldots, x_n : T_n) = P]\!] = \\
&\qquad \texttt{process ProcName; } lVars(P) \\
&\qquad \texttt{begin } isInvoked(\mathsf{ProcName}) \\
&\qquad \textsc{Tproc}[\![P]\!] \\
&\qquad \texttt{end}
\end{aligned}
$$

FIGURE 6
Program transcription TPROG[[]].

larations followed by a list of process blocks. Each process block corresponds to a named CQP process, and these are translated in turn using TPROG[[]].

Each QMC process declaration defines a single process containing local variable declarations and a process body, structured with the **process**, **begin** and **end** keywords. Given a single CQP process definition as input, TPROG[[]] rewrites this definition into a QMC process declaration.

For processes that are invoked from (that is, nested within) other processes the function *isInvoked* inserts a receive statement (`Proc_ctrl?signal;`, where *Proc* is the process name) that is used to signal the start of execution. For non-invoked processes the function *isInvoked* produces no output. The set of invoke processes is found be analysis of the structure of the CQP program.

Processes

The CQP process body is transcribed via the function TPROC[[]] defined in Figure 7. The CQP **0** process is rewritten to an empty string. The "invocation" of parallel processes ($P1 \mid P2$) is achieved through signalling; as described in the previous section, processes that are invoked will await a signal before proceeding (determined by the function *isInvoked*) hence sending this signal allows the invoked process to begin execution.

$\textsc{Tproc}[\![\mathbf{0}]\!] = \varepsilon$

$\textsc{Tproc}[\![(P1(\tilde{x}) \mid P2(\tilde{y}))]\!] = \texttt{P1\_ctrl!signal; P2\_ctrl!signal;}$

$\textsc{Tproc}[\![e?[x_1 : T_1, \cdots, x_n : T_n].P]\!] =$

$\quad \textsc{Texpr}[\![e]\!]\texttt{1?}\textsc{Tval}[\![x_1]\!] \cdots \textsc{Texpr}[\![e]\!]\texttt{n?}\textsc{Tval}[\![x_n]\!]$

$\quad \textsc{Texpr}[\![e]\!]\texttt{\_ack!ack;}\textsc{Tproc}[\![P]\!]$

$\textsc{Tproc}[\![e![e_1, \cdots, e_n].P]\!] = \texttt{e\_1:=}\textsc{Texpr}[\![e_1]\!]\texttt{; } \ldots \texttt{e\_n:=}\textsc{Texpr}[\![e_n]\!]\texttt{;}$

$\quad \textsc{Texpr}[\![e]\!]\texttt{1!e\_1; } \ldots \textsc{Texpr}[\![e]\!]\texttt{n!e\_n;}$

$\quad \textsc{Texpr}[\![e]\!]\texttt{\_ack?ack;}\textsc{Tproc}[\![P]\!]$

$\textsc{Tproc}[\![\{e\}.P]\!] = \textsc{Texpr}[\![e]\!]\ \textsc{Tproc}[\![P]\!]$

$\textsc{Tproc}[\![(\mathsf{new}\ x{:}\widehat{\ }[T])P]\!] = \textsc{Tproc}[\![P]\!]$

$\textsc{Tproc}[\![(\mathsf{qbit}\ x)P]\!] = \texttt{x := newqubit;}\textsc{Tproc}[\![P]\!]$

FIGURE 7
$\textsc{Tproc}[\![]\!]$: Translation of processes.

$\textsc{Texpr}[\![v]\!] = \textsc{Tval}[\![v]\!]$

$\textsc{Texpr}[\![\mathsf{measure}\ e_1, \ldots, e_n]\!] = \texttt{meas}\textsc{Texpr}[\![e_1]\!], \ldots, \textsc{Texpr}[\![e_n]\!]\texttt{;}$

$\textsc{Texpr}[\![e_1, \ldots, e_n \mathrel{*}= e^f]\!] = \texttt{if ::(}\textsc{Texpr}[\![f]\!]\texttt{= 1) ->}$

$\textsc{Texpr}[\![e]\!]\textsc{Texpr}[\![e_1]\!]\texttt{, ..., }\textsc{Texpr}[\![e_n]\!]\texttt{; break;}$

$\texttt{::(}\textsc{Texpr}[\![f]\!]\texttt{= 0) -> break; fi}$

$\textsc{Texpr}[\![e + e]\!] = \textsc{Texpr}[\![e]\!]\texttt{+}\textsc{Texpr}[\![e]\!]$

FIGURE 8
$\textsc{Texpr}[\![]\!]$: Translation of expressions.

CQP and QMC use different models of communication; in the former, communication is synchronous, thus an input and output action must execute as one step. In contrast, communication in QMC is asynchronous, therefore an output action occurs strictly, but not necessarily immediately, before a corresponding input action. We therefore simulate synchronous communication in QMC by forcing the sending process to wait for an acknowledgement from the receiving process, thus each CQP output action will be followed by an input action when translated and similarly an output action will follow each CQP input action.

CQP channels are polyadic (allow multiple subjects) whereas in QMC channels are monadic (have a single subject), hence it is necessary to separate CQP communication actions into multiple QMC actions. Each resulting action must use a distinct channel name since channels are typed in both languages. Furthermore, QMC allows only variable names as the subject in send and receive actions, hence it is necessary to assign any value or expression to be sent to a fresh variable before sending. For example, the CQP action $ch![x, 3]$ is translated to

```
ch_1 := x; ch1!ch_1; ch_2 := 3; ch2!ch_2;
```

Although it is not necessary to make the assignment for x in this translation, this convention is extended to variables for the purpose of generalisation.

v	TVAL$[\![v]\!]$
$x, q, c, \ldots$	`x`, `q`, `c`, ...
$0, 1, \ldots$	`0`, `1`,...
$\mathsf{X}, \mathsf{Y}, \mathsf{Z}$	**`X`**, **`Y`**, **`Z`**
H	**`had`**
CNot	**`cnot`**

T	TTYPE$[\![T]\!]$
Int	**`integer`**
Qbit	**`qubit`**
$\hat{}[T]$	**`channel of`** TTYPE$[\![T]\!]$

FIGURE 9
TVAL$[\![]\!]$ and TTYPE$[\![]\!]$: Translation of values and types.

Expressions

CQP expressions consist of values, quantum measurements, quantum operations, and arithmetic expressions. We define the function TEXPR$[\![]\!]$ in Figure 8 for the translation of expressions. Values v are translated by TVAL$[\![]\!]$ (defined later). Quantum measurements on multiple qubits are possible in both languages, however they follow different conventions for the assignment of values to outcomes (e.g. values $0, 1, 2, 3$ in CQP as opposed to pairs $(0,0), (0,1), (1,0), (1,1)$ in QMC). We therefore restrict the translation to single qubit measurements (on which the resulting values correspond) and note that this does not impact expressivness. Quantum operations are transcribed with the quantum operator first, followed by a comma separated list of qubit names. Controlled operations U^e are implemented using an **`if`** construct. We only consider bit values (generally resulting from single qubit measurements) for controlled operations, although this could be extended to allow any integer using the fact that $U^4 = I$ for all operators in the stabilizer formalism. Addition is the only arithmetic operator formally defined in CQP, however the translation is easily extended to other arithmetic expressions.

Values and Types

The functions TVAL$[\![]\!]$ and TTYPE$[\![]\!]$ defined in Figure 9 are used for the translation of values and types respectively. Variable names and literal values are left unchanged, while quantum operators are mapped to their QMC equivalents as per the definition. There is no translation for arbitrary quantum operators since only the operators in the stabilizer formalism are supported by QMC. The types $\mathsf{Unit}, \mathsf{Op}(1), \ldots$ corresponding to arbitrary operator types need not be translated since they are not used in QMC programs. Channel types $\hat{}[\tilde{T}]$ make a recursive call to translate the component types. Since we

don't allow channel mobility in the translation, declarations which are not allowed in QMC such as **`channel of channel of`** `T` are excluded.

Variable Declarations
QMC requires variables to be explicitly declared prior to use, either in a local process scope or globally. We take the approach of placing all variables in the global scope to avoid issues arising from our implementation of invocation. In particular, since channel names cannot be sent over QMC channels whereas other variables can, it is not possible to simulate inheritance of channel names. Instead we state the requirement that variable names must be unique, hence scoping in QMC will have no effect.

The functions $gVars$ and $lVars$ are used by TPROG$[\![]\!]$ in the translation of programs to provide global and local variable declarations respectively for the QMC program. $gVars$ inspects the CQP program for binding operators including $c?[\widetilde{x}{:}\widetilde{T}]$ (gives declarations for each x_i), $(\mathsf{new}\ \widehat{x}[\widetilde{T}])$ (declares a series of channels x_i of type T_i), and $(\mathsf{qbit}\ x)$ (declares x as a qubit variable). Signalling channels (`procname_ctrl: channel of integer;`) for each process are also declared regardless of which processes will use them. Finally, $gVars$ adds declarations for `signal` and `ack` which may be used for signalling and acknowledgments respectively.

Since we have chosen to declare all variables globally, the only use for $lVars$ is to generate declarations for the intermediate assignments that arise from output actions; an output $c![\widetilde{v}]$ will results in declarations `c_i: T_i` where T_i are the types associated with channel c.

6.2 Example

We now apply this translation to the teleportation process defined in Figure 3. We expect the result to resemble the QMC program in Figure 5; the result is shown in Figure 10.

It is not surprising that the programs aren't identical since differences in the languages allow for alternate representations of various components. The first point to note is the use of the signalling channels `Teleport_ctrl`, `Alice_ctrl` and `Bob_ctrl`. Since the $Teleport$ process is not nested, the corresponding control channel is declared but never used. As it happens, in teleportation, the addition of the control channel in Bob is superfluous because execution cannot start until a value is received from $Alice$.

Another change is the conditional applications of the unitary operators by Bob; these have been compounded into one **`if`** statement in the QMC specification, however the simplistic support for conditionals by CQP leads to multiple statements in the translation.

```
program Translated;
var x: qubit; y: qubit; z: qubit;
 e1: channel of integer;
 e2: channel of integer;
 e_ack: channel of integer;
 Teleport_ctrl: channel of integer;
 Alice_ctrl: channel of integer;
 Bob_ctrl: channel of integer;
 r: integer; s: integer;
 signal: integer; ack: integer;
process Teleport;
begin
 y := newqubit; z := newqubit;
 had z; cnot z y;
 Alice_ctrl!1; Bob_ctrl!1;
end;
process Alice;
var e_1: integer; e_2: integer;
begin
 Alice_ctrl?signal;
 x := newqubit; had x; cnot z x; had z;
 e_1 := meas x; e1!e_1; e_2 := meas z; e2!e_2;
 e_ack?ack;
end;
process Bob;
begin
 Bob_ctrl?signal;
 e1?r; e2?s; e_ack!1;
 if :: (r = 1) -> Z y; break;
    :: (r = 0) -> break; fi
 if :: (s = 1) -> X y; break;
    :: (s = 0) -> break; fi
end;
endprogram
```

FIGURE 10
Translated version of quantum teleportation.

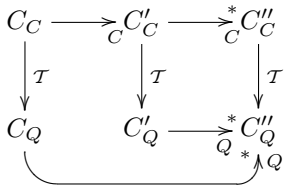

FIGURE 11
The requirement for a transition $\mathcal{T}$ to be semantics preserving.

6.3 Correctness of the Translation

In order to argue that a QMC program translated from CQP has the same meaning as the original CQP process, it is necessary to show that the semantics of CQP processes is preserved by the translation function – this is equivalent to saying that the result of translating a CQP process into QMC then executing it should be identical to executing it in CQP then translating the result to QMC [24].

The operational semantics of both languages are defined in terms of single step transitions from one configuration to another. A CQP configuration is a tuple $(q_1, \ldots, q_n = |\psi\rangle; \phi; P)$, which includes, alongside the process P, the assignment of quantum names to a quantum state $|\psi\rangle$, and a channel list ϕ. A QMC configuration is a tuple $(P, \kappa, \Sigma, |\psi\rangle)$, which includes the process P with global (κ) and local (Σ) registers and a quantum state $|\psi\rangle$.

The relationship between configurations of CQP (C_C) and QMC (C_Q), that is required for a semantics preserving translation $\mathcal{T}$, is illustrated in Figure 11. In the ideal case, each CQP transition $C_C \longrightarrow_C C'_C$ is matched by a QMC transition $\mathcal{T}(C_C) \longrightarrow_Q \mathcal{T}(C'_C)$. To account for the peculiarities of the translation we instead use a notion of semantic equivalence; that is, there exists a configuration C''_C such that $C'_C \longrightarrow^*_C C''_C$ ($\longrightarrow^*_C$ represents zero or more transitions) and $\mathcal{T}(C_C) \longrightarrow^*_Q \mathcal{T}(C''_C)$ and $\mathcal{T}(C'_C) \longrightarrow^*_Q \mathcal{T}(C''_C)$. This accounts for cases such as output where temporary variable assignments are used, and where extra steps are introduced such as process signalling.

Figure 11 refers to a translation of configurations, $\mathcal{T}$, which we must define in order to consider the preservation of semantics. The extension from the syntactic translation, given by the function TPROG$[\![]\!]$, to the translation $\mathcal{T}$ is a

straightforward one; let

$$\mathcal{T}((q_1, \ldots, q_n = |\psi\rangle; \phi; P)) = \\ eval_{VD}((\mathcal{T}_P(P), \mathcal{T}_\kappa(\phi), \mathcal{T}_\sigma(q_1, \ldots, q_n, P), \mathcal{T}_\psi(|\psi\rangle)))$$

where $\mathcal{T}_P$ translates the process using $\textsc{Tprog}[\![]\!]$, $\mathcal{T}_\kappa$ populates the global store with the channel names from ϕ, $\mathcal{T}_\sigma$ populates the local stores with the qubit variables in each process, and $\mathcal{T}_\psi$ is an identity map on the quantum state. Not all variables can be determined directly from the configuration, but are found by analysis of the program; as a result of the translation by $\textsc{Tprog}[\![]\!]$, variable declaration statements are created for these remaining variables. $eval_{VD}$ represents the execution of these variable declarations, thereby fully populating the global and local stores.

Theorem 1 (Preservation of Semantics)**.** *Let $C_C = (\widetilde{q} = |\psi\rangle; \phi; P)$. If P is well typed and $C_C \longrightarrow_C C'_C$ then there exists C''_C such that $C'_C \longrightarrow^*_C C''_C$ and $\mathcal{T}(C_C) \longrightarrow^*_Q \mathcal{T}(C''_C)$ and $\mathcal{T}(C'_C) \longrightarrow^*_Q \mathcal{T}(C''_C)$.*

Proof. By induction on the derivation of $C_C \longrightarrow_C C'_C$. In this paper we only show the case for communication; other cases follow similar reasoning.

Let $C_C = (\sigma; \phi; c![\widetilde{v}, \widetilde{q}].Q \mid c?[\widetilde{x}{:}\widetilde{T}, \widetilde{y}{:}\widetilde{\mathsf{Qbit}}].R)$ where $\sigma = q_1, \ldots, q_r = |\psi\rangle$, $\widetilde{v}$ are non-qubit values and $\widetilde{q} = q_1, \ldots, q_n$ for $n \leq r$. Then we have the transition

$$C_C \longrightarrow_C C'_C = (\sigma; \phi; Q \mid R\{\widetilde{v}, \widetilde{q}/\widetilde{x}, \widetilde{y}\}) \,.$$

The translation of C_C is

$$\begin{aligned}\mathcal{T}(C_C) = (&c_1 := v_1; \ldots, c_m := v_m; c_{m+1} := q_1; \ldots, c_{m+n} := q_n; \\ &c1!c_1; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\ \mid &c1?x_1; \ldots cm?x_m; c(m+1)?y_1; \ldots c(m+n)?y_n; \\ &c_{\text{ack}}!1; \textsc{Tproc}[\![R]\!], \kappa, (\sigma_Q, \sigma_R), |\psi\rangle)\end{aligned}$$

Let $C_Q = \mathcal{T}(C_C)$. The first transitions from this QMC configuration are the

assignments to temporary variables $(c_1, \ldots, c_{m+n})$ prior to sending.

$$
\begin{aligned}
C_Q \longrightarrow_Q\ & (c_2 := v_2; \ldots, c_m := v_m; c_{m+1} := q_1; \ldots, c_{m+n} := q_n; \\
& c1!c_1; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\
& \mid c1?x_1; \ldots cm?x_m; c(m+1)?y_1; \ldots c(m+n)?y_n; c_{\text{ack}}!1; \\
& \textsc{Tproc}[\![R]\!], \kappa, (\sigma_Q[c_1 \mapsto v_1], \sigma_R), |\psi\rangle) \\
\longrightarrow_Q^*\ & (c_{m+1} := q_1; \ldots, c_{m+n} := q_n; \\
& c1!c_1; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\
& \mid c1?x_1; \ldots cm?x_m; c(m+1)?y_1; \ldots c(m+n)?y_n; c_{\text{ack}}!1; \\
& \textsc{Tproc}[\![R]\!], \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m], \sigma_R), |\psi\rangle) \\
\longrightarrow_Q^*\ & (c1!c_1; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\
& \mid c1?x_1; \ldots cm?x_m; c(m+1)?y_1; \ldots c(m+n)?y_n; c_{\text{ack}}!1; \\
& \textsc{Tproc}[\![R]\!], \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m] \\
& [q_1 \mapsto \mathsf{null}][c_{m+1} \mapsto 1] \cdots [q_n \mapsto \mathsf{null}][c_{m+n} \mapsto n], \sigma_R), |\psi\rangle)
\end{aligned}
$$

The ordering of the next sequence of transitions, in which one process sends and the other receives the values, is non-deterministic due to the possible interleavings. We show one possible execution and note that all executions will arrive at the same final configuration.

$$
\begin{aligned}
\longrightarrow_Q\ & (c2!c_2; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\
& \mid c1?x_1; \ldots cm?x_m; c(m+1)?y_1; \ldots c(m+n)?y_n; c_{\text{ack}}!1; \\
& \textsc{Tproc}[\![R]\!], \kappa[c1 \mapsto v_1], (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m] \\
& \quad [q_1 \mapsto \mathsf{null}][c_{m+1} \mapsto 1] \cdots [q_n \mapsto \mathsf{null}][c_{m+n} \mapsto n], \sigma_R), |\psi\rangle) \\
\longrightarrow_Q\ & (c2!c_2; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\
& \mid c2?x_2; \ldots cm?x_m; c(m+1)?y_1; \ldots c(m+n)?y_n; c_{\text{ack}}!1; \\
& \textsc{Tproc}[\![R]\!], \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m][q_1 \mapsto \mathsf{null}][c_{m+1} \mapsto 1] \\
& \quad \cdots [q_n \mapsto \mathsf{null}][c_{m+n} \mapsto n], \sigma_R[x_1 \mapsto v_1]), |\psi\rangle) \\
\longrightarrow_Q^*\ & (c(m+1)!c_{m+1}; \ldots; c(m+n)!c_{m+n}; c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \\
& \mid c(m+1)?y_1; \ldots c(m+n)?y_n; c_{\text{ack}}!1; \textsc{Tproc}[\![R]\!], \\
& \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m][q_1 \mapsto \mathsf{null}][c_{m+1} \mapsto 1] \\
& \quad \cdots [q_n \mapsto \mathsf{null}][c_{m+n} \mapsto n], \sigma_R[x_1 \mapsto v_1] \cdots [x_m \mapsto v_m]), |\psi\rangle) \\
\longrightarrow_Q^*\ & (c_{\text{ack}}?\text{ack}; \textsc{Tproc}[\![Q]\!] \mid c_{\text{ack}}!1; \textsc{Tproc}[\![R]\!], \\
& \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m][q_1 \mapsto \mathsf{null}] \cdots [q_n \mapsto \mathsf{null}], \\
& \sigma_R[x_1 \mapsto v_1] \cdots [x_m \mapsto v_m][y_1 \mapsto 1] \cdots [y_n \mapsto n]), |\psi\rangle)
\end{aligned}
$$

The final transitions are for the message acknowledgement, preventing Q from proceeding until the communication is complete.

$$\begin{aligned}\longrightarrow_Q^* \ & (\textsc{Tproc}[\![Q]\!] \mid \textsc{Tproc}[\![R]\!], \\ & \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m][q_1 \mapsto \mathsf{null}] \cdots [q_n \mapsto \mathsf{null}][\mathrm{ack} \mapsto 1], \\ & \sigma_R[x_1 \mapsto v_1] \cdots [x_m \mapsto v_m][y_1 \mapsto 1] \cdots [y_n \mapsto n]), |\psi\rangle)\end{aligned}$$

Finally, let us rename the variables in process R (we can take the substitution of names inside $\textsc{Tproc}[\![R]\!]$) to give

$$\begin{aligned}C'_Q = \ & (\textsc{Tproc}[\![Q]\!] \mid \textsc{Tproc}[\![R\{\widetilde{v}, \widetilde{q}/\widetilde{x}, \widetilde{y}\}]\!], \\ & \kappa, (\sigma_Q[c_1 \mapsto v_1] \cdots [c_m \mapsto v_m][q_1 \mapsto \mathsf{null}] \cdots [q_n \mapsto \mathsf{null}][\mathrm{ack} \mapsto 1], \\ & \sigma_R[v_1 \mapsto v_1] \cdots [v_m \mapsto v_m][q_1 \mapsto 1] \cdots [q_n \mapsto n]), |\psi\rangle)\end{aligned}$$

Due to type preservation, we have that $\mathcal{T}_\sigma(q_1, \ldots, q_r, Q)$ does not include qubits $\widetilde{q}$, which are instead provided by $\mathcal{T}_\sigma(q_1, \ldots, q_r, R)$. We have

$$\begin{aligned}\mathcal{T}(C'_C) = \ & (\textsc{Tproc}[\![Q]\!] \mid \textsc{Tproc}[\![R\{\widetilde{v}, \widetilde{q}/\widetilde{x}, \widetilde{y}\}]\!], \kappa, \\ & (\sigma_Q[q_1 \mapsto \mathsf{null}] \cdots [q_n \mapsto \mathsf{null}], \sigma_R[q_1 \mapsto 1] \cdots [q_n \mapsto n]), |\psi\rangle)\end{aligned}$$

The temporary variables $c_1, \ldots, c_m$ do not appear in Q, hence we can conclude that $\mathcal{T}(C'_C) = C'_Q$. □

7 CONCLUSIONS AND FUTURE WORK

Process calculus and model checking are two formal methods that have proved successful for the verification of classical systems. We believe that similar benefits are likely to be realised through the application of these techniques to quantum communication systems.

In this paper we have described an approach to the verification of quantum protocols using both process calculus and model checking. The language CQP extends the classical π-calculus with the inclusion of primitives for quantum information, and the QMC tool uses the stabilizer formalism to offset the complexity of simulating quantum computations on classical computers.

The major advantage of our approach is in the combination of techniques, enabling us to reap the benefits of both CQP and QMC. In Section 6 we defined a translation from CQP to the modelling language of QMC which facilitates the application of process calculus techniques and model checking

using a single specification. We have formally specified and proved the requirements for the semantic correctness of this translation (Theorem 1). In Section 6.2 we demonstrated the translation by converting a CQP model of quantum teleportation into QMC.

Further work will include the development of techniques for equational reasoning about quantum processes in CQP. Some authors of this paper have made significant progress towards a congruence relation for quantum processes, which will mark a significant step in the development of quantum process calculus. These techniques can then be used in conjunction with model checking in QMC, via this translation, to verify properties of quantum protocols, and eventually, practical quantum cryptography and communication systems.

Both CQP and QMC provide scope for extending their individual functionality, for example, incorporating complex data structure and recursion. In particular, support for arbitrary quantum operators in QMC would allow us to consider attacks on cryptographic protocols that fall outside the stabilizer formalism.

ACKNOWLEDGEMENTS

The authors acknowledge support from the EPSRC grant no. EP/E00623X/1 *(Semantics of Quantum Computation)*. The second author is partially supported by the EPSRC grant no. EP/F020813/1 *Quantum Computation: Foundations, Security, Cryptography and Group Theory*. The third author would like to acknowledge the support from the *Warwick Institute for Financial Computing* and from the Czech Grant Agency, grant no. GA201/07/0603. The fourth and fifth authors were partially supported by the EU Sixth Framework Programme (Project SecoQC: *Development of a Global Network for Secure Communication based on Quantum Cryptography*).

REFERENCES

[1] Scott Aaronson and Daniel Gottesman. (2004). Improved simulation of stabilizer circuits. *Physical Review A (Atomic, Molecular, and Optical Physics)*, 70(5):052328.

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

[3] C. H. Bennett and G. Brassard. (December 1984). Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, Bangalore, India.

[4] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. (Mar 1993). Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, 70(13):1895–1899.

[5] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. (2000). *Model Checking*. MIT Press.

[6] Artur K. Ekert. (Aug 1991). Quantum cryptography based on bell's theorem. *Phys. Rev. Lett.*, 67(6):661–663.

[7] C. Elliott. (July 2004). Quantum cryptography. *IEEE Security & Privacy*, 2(4):57–61.

[8] E. Allen Emerson. (1990). *Temporal and modal logic*, volume B: Formal Models and Semantics, pages 995–1072. MIT Press.

[9] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying, (2006). Probabilistic bisimilarities between quantum processes. arXiv:cs.LO/0601014.

[10] Simon Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou, (2005). Probabilistic model–checking of quantum protocols.

[11] Simon J. Gay. (2006). Quantum programming languages: survey and bibliography. *Math. Struct. in Comp. Sci.*, 16(4):581–600.

[12] Simon J. Gay and Rajagopal Nagarajan. (2005). Communicating Quantum Processes. In *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 145–157, New York, NY, USA. ACM Press.

[13] Simon J. Gay and Rajagopal Nagarajan. (2006). Types and Typechecking for Communicating Quantum Processes. *Math. Struct. in Comp. Sci.*, 16(3):375–406.

[14] Simon J. Gay, Nikolaos Papanikolaou, and Rajagopal Nagarajan. (July 2008). QMC: a model checker for quantum systems. In *CAV 2008: In Proceedings of the 20th International Conference on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 543–547. Springer-Verlag.

[15] Daniel Gottesman. (1998). The heisenberg representation of quantum computers. In *International Conference on Group Theoretic Methods in Physics*.

[16] Jozef Gruska. (1999). *Quantum Computing*. McGraw Hill.

[17] M. Kwiatkowska, G. Norman, and D. Parker. (September 2001). PRISM: Probabilistic symbolic model checker. In P. Kemper, editor, *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 7–12. Available as Technical Report 760/2001, University of Dortmund.

[18] Marie Lalire, (2005). A probabilistic branching bisimulation for quantum processes. arXiv:quant-ph/0508116.

[19] Marie Lalire. (2006). Relations among quantum processes: bisimilarity and congruence. *Math. Struct. in Comp. Sci.*, 16(3):407–428.

[20] P. Mateus and A. Sernadas. (2006). Weakly complete axiomatization of exogenous quantum propositional logic. *Information and Computation*, 204(5):771—794.

[21] Dominic Mayers. (2001). Unconditional security in quantum cryptography. *J. ACM*, 48(3):351–406.

[22] Rajagopal Nagarajan and Simon J. Gay, (2002). Formal verification of quantum protocols. arXiv:quant-ph/0203086.

[23] Michael A. Nielsen and Isaac L. Chuang. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press.

[24] Hanne Riis Nielson and Flemming Nielson. (1999). Semantics with applications: A formal introduction. Revised edition; original published by John Wiley & Sons, 1992.

[25] Nikolaos Papanikolaou. (2009). *Model Checking Quantum Protocols*. PhD thesis, Department of Computer Science, University of Warwick.

[26] David Park. (1981). Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK. Springer-Verlag.

[27] Eleanor Rieffel and Wolfgang Polak, (1998). An introduction to quantum computing for non-physicists. arXiv:quant-ph/9809016v2.

[28] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. (2001). *Modelling and Analysis of Security Protocols*. Addison Wesley.

[29] P. W. Shor. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *SFCS '94: Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Washington, DC, USA. IEEE Computer Society.

[30] W. K. Wootters and W. H. Zurek. (1982). A single quantum cannot be cloned. *Nature*, 299:802—803.

[31] Mingsheng Ying, Yuan Feng, and Runyao Duan, (Jul 2007). An algebra of quantum processes. `http://arxiv.org/abs/0707.0330v1`.

[32] Mingsheng Ying, Yuan Feng, Runyao Duan, and Zhengfeng Ji. (2009). An algebra of quantum processes. *ACM Trans. Comput. Logic*, 10(3):1–36.