QMC: A Model Checker for Quantum Systems

Simon J. Gay[1][*], Rajagopal Nagarajan[2][*], and Nikolaos Papanikolaou[2][*]

[1] Department of Computing Science, University of Glasgow
simon@dcs.gla.ac.uk
[2] Department of Computer Science, University of Warwick
{biju,nikos}@dcs.warwick.ac.uk

1 Introduction

The novel field of quantum computation and quantum information has been growing at a rapid rate; the study of quantum information in particular has led to the emergence of communication and cryptographic protocols with no classical analogues. Quantum information protocols have interesting properties which are not exhibited by their classical counterparts, but they are most distinguished for their applications in cryptography. Notable results include the unconditional security proof [1] of quantum key distribution [2]. This result in particular is one of the reasons for the widespread interest in this field, and it demonstrates an achievement not possible in classical cryptographic systems. Furthermore, the implementation of quantum cryptography has been demonstrated in non-laboratory settings and is already an important practical technology. Implementations of quantum cryptography have already been commercially launched and tested by a number of companies including MagiQ, Id Quantique, Toshiba, and NEC. The unconditional security of quantum key distribution protocols does not automatically imply the same degree of security for actual systems, of course; this justifies the need for systems modelling and verification in this setting.

The benefits of automated verification techniques are well known for classical communication protocols, especially in the cryptographic setting. Our research programme is to apply these techniques to *quantum* protocols with the expectation of gaining corresponding benefits. Our earlier work involved applying probabilistic model–checking techniques to such protocols [3, 4]; it became clear that existing techniques are not satisfactory for analysing quantum systems. Today, while simulation tools for quantum systems abound, to our knowledge no other authors have developed a tool directly aimed at verification. In this paper we describe just such a tool, named QMC (Quantum Model Checker); it allows for automated verification of properties of quantum protocols. Properties to be verified by QMC are expressed using the logic QCTL [5], a logic designed specifically for reasoning about quantum systems. QMC enables the modelling of systems which can be expressed within the quantum *stabilizer formalism*, which is known to be simulable in polynomial time in accordance with the Gottesman–Knill theorem [6]. The systems expressible in this formalism are restricted, in the sense that the set of operations which they can perform is not universal for quantum computation. Nevertheless, stabilizers are sufficient to describe a number of systems of practical interest. We assume that the reader has an understanding of the basics of quantum computation [7].

2 Tool Description

QMC is a command–line tool, implemented in Java 5, which simulates all possible executions of a quantum protocol model, and then checks that the model satisfies any number of specification formulae supplied by the user. Both the model and its properties are placed in a single input file. We will now discuss the modelling language of QMC, the simulation of models, the logic used for specifying properties, and the algorithms used for model–checking. Then we will discuss a concrete example, namely a model of quantum coin-flipping [2], the QMC source for which is listed in the appendix.

Modelling Language. Protocols are modelled using a high-level, concurrent programming language developed especially for QMC, with influences from Pascal and the language PROMELA [8]. It is an imperative language with lightweight concurrency, typed variables (which may be local to a process or shared) including channels, and commands for allocating and manipulating the qubits in a global quantum state. Qubit variables are pointers to the individual quantum bits (qubits) in the global state. The language has guarded commands, which are used to express looping and nondeterministic choices.

QMC has a statically typed language, and variables of different types must be declared at the beginning of a protocol model (in which case they become shared variables), or at the beginning of the process in which they are used (in which case they become local to that process). The classical data types include integers (type **integer**), bits (type **bool**), and floating–point numbers (type **real**). There is a channel type corresponding to each of these base types (e.g. **channel of integer**).

Qubit variables are references to qubits in QMC's internal quantum state. After declaration, these variables have no value, and must be initialized with the newqubit expression. A process can apply any one of the *Clifford group operators* (see [7]): Hadamard (**had**), Controlled–not (**cnot**), Phase (**ph**) to a given qubit, as well as performing quantum measurement with respect to the computational basis. These are the operations allowed in the quantum stabilizer formalism.

Simulating QMC *programs.* As in PROMELA, any command in QMC is either *executable* or not from the current state. QMC's process scheduler interleaves executable statements from all process declarations in the input, resolving all non–deterministic choices. Quantum measurement is also treated as a source of non-determinism; a qubit in a general stabilizer state always produces one of two possible outcomes with equal probability, and we treat these equiprobable outcomes like a non–deterministic choice. Consequently, we do not analyse the probabilities of different execution paths; this is an area for future work. The scheduler ensures finiteness of models by requiring that all loops are executed for a maximum of 100 times, and by ending simulation if there are only non-executable processes remaining. We are currently developing an operational semantics for the QMC language. Execution of a QMC program produces a tree of execution paths, in which the branching arises from non–determinism.

Specification Logic. The properties of a given protocol model are expressed using the logic QCTL [5], which is a temporal logic especially designed for reasoning about quantum systems based on CTL [9]. In particular, QCTL adds the usual CTL modalities to exogenous quantum propositional logic (EQPL [10]). This logic allows us to

reason about the evolution of the global quantum state as a given protocol model is executed. It also enables reasoning about the states of classical variables and, hence, measurement outcomes. The basic syntax of this logic is stated below for completeness; the reader is referred to [5] for details.

$$\alpha = \bot \mid \mathsf{qb} \mid \alpha \Rightarrow \alpha \qquad \qquad \text{(classical formulae)}$$
$$t = x \mid (t+t) \mid (tt) \mid \mathrm{Re}(\lvert \top \rangle_A) \mid \mathrm{Im}(\lvert \top \rangle_A) \mid \int \phi \qquad \text{(terms)}$$
$$\gamma = (t \leq t) \mid \bot \mid (\alpha \sqsupset \alpha) \qquad \qquad \text{(quantum formulae)}$$
$$\theta = \gamma \mid \theta \sqsupset \theta \mid (\mathsf{EX}\theta) \mid (\mathsf{E}[\theta \mathsf{U}\theta]) \mid (\mathsf{AF}\theta) \qquad \text{(temporal formulae)}$$

Interesting state formulas are those of the form $\int \phi \leq a$ and $[\mathsf{qb}_i, \mathsf{qb}_j]$, where ϕ is a base formula (e.g. $\neg\mathsf{qb}_0$) and $\mathsf{qb}_i, \mathsf{qb}_j, \mathsf{qb}_0$ are qubit variables. The first formula states that, the probability of formula ϕ being satisfied in the current state is less than or equal to a. The formula $[\mathsf{qb}_i, \mathsf{qb}_j]$ states that, in the current state, the qubits corresponding to the variables qb_i and qb_j are *not entangled* with the rest of the quantum system. In order to evaluate such a formula, QMC analyses the entanglement of the current quantum state.

Verification Algorithms. QMC implements algorithms for evaluating EQPL formulas over stabilizer states, which are represented internally using a matrix representation (see [11]). In order to check the truth of a particular formula, its truth needs to be determined for all possible valuations; the tool automatically extracts all valuations from the internal representation. More interestingly, the tool has been designed to explore all possible executions of a particular protocol arising from different measurement outcomes and non–deterministic choices.

Entanglement formulae may be checked without converting the internal representation to the set of all valuations. In fact, it is possible to determine whether a list of qubits $q_i, q_{i+1}, \ldots$ constitutes a partition of the global quantum state at any point during execution using the *entanglement normal forms* developed by Audenaert and Plenio [12]. Temporal formulae are checked by traversing the tree produced by the execution of a protocol model.

An Example. In the appendix, we include a listing of the QMC model for the quantum coin–flipping protocol due to Bennett and Brassard [2]. Quantum coin–flipping enables two users, Alice and Bob, to establish a common random bit x through the transmission of a single qubit q and its measurement. The protocol relies on the principle that, if Alice and Bob use compatible bases for preparation and for measurement of this qubit, their bit values will be guaranteed to match by the laws of quantum mechanics. Incompatible bases will produce a matching bit value only with probability $\frac{1}{2}$, although in this case the protocol is repeated and the bit discarded. There are various possible attacks that may be performed by an enemy, which would enable him or her to compromise the final bit value. It is such attacks that we would like to investigate using the QMC tool on this and related quantum protocols. The basic property which needs to be checked at the end of the protocol is that the bit values of Alice and Bob (i.e. the variables Alice.result and Bob.result) do indeed match, and this is only true if the measurement basis $\hat{b}$ chosen by Bob matches the preparation basis b of Alice. Also, if Alice's and Bob's bits and bases are in agreement, the protocol should not abort; this is expressed as a temporal property.

3   Conclusion and Future Work

We have described QMC, a model-checking tool for quantum protocols. As far as we know, it is the first dedicated verification tool (as opposed to simulation systems) for quantum protocols. QMC allows the modelling and verification of properties of protocols expressible in the quantum stabilizer formalism. The logic for expressing properties is QCTL. We have considered a simple example illustrating the input language of the tool.

It is significant to note that the restriction to quantum stabilizer states allows QMC to simulate protocols efficiently, although it limits the expressive power of the tool. There are many quantum protocols which involve quantum states that fall outside the scope of the stabilizer formalism. We are currently investigating ways of approximating such states using techniques due to Bravyi and Kitaev [13]. Using such techniques it will be possible to obtain an implementation which is powerful enough for the analysis of general quantum protocols and the detection of potential flaws in their design.

References

[1] Mayers, D.: Unconditional security in quantum cryptography. Journal of the ACM **48**(3) (2001) 351—406
[2] Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: Proceedings of International Conference on Computers, Systems and Signal Processing. (December 1984)
[3] Nagarajan, R., Gay, S.J.: Formal verification of quantum protocols. arXiv:quant-ph/0203086 (2002)
[4] Gay, S.J., Nagarajan, R., Papanikolaou, N.: Probabilistic model–checking of quantum protocols. DCM 2006: Proceedings of the 2nd International Workshop on Developments in Computational Models (2006) arXiv:quant-ph/0504007.
[5] Baltazar, P., Chadha, R., Mateus, P., Sernadas, A.: Towards model-checking quantum security protocols. In P. Dini *et al.*, ed.: Proceedings of the First Workshop on Quantum Security: QSec'07, IEEE Press (2007)
[6] Gottesman, D.: The Heisenberg representation of quantum computers. In Corney, S., Delbourgo, R., Jarvis, P., eds.: Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics, International Press (1999)
[7] Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
[8] Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Pearson Education (2003)
[9] Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite state concurrent system using temporal logic specifications: a practical approach. In: POPL '83: Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, New York, NY, USA, ACM (1983) 117–126
[10] Mateus, P., Sernadas, A.: Weakly complete axiomatization of exogenous quantum propositional logic. Information and Computation **204**(5) (2006) 771—794
[11] Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. Physical Review A **70**(52328) (2004)
[12] Audenaert, K., Plenio, M.: Entanglement on mixed stabiliser states: Normal forms and reduction procedures. New Journal of Physics **7**(170) (2005)
[13] Bravyi, S., Kitaev, A.: Universal quantum computation with ideal Clifford gates and noisy ancillas. Physical Review A **71** (2005) 1–14

Appendix:
QMC Model of the Quantum Coin-Flipping Protocol

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

process Alice;
var x, b, g, result : bool; q: qubit;
begin
    q := newqubit;
    {Choose one of the four BB84 states and prepare qubit q}
    if
    □ true →x:=false; b:=false;
    □ true →x:=false; b:=true; had q;
    □ true →x:=true; b:=false; X q;
    □ true →x:=true; b:=true; X q; had q;
    fi
    AtoB!q;
    B2A?g;
    A2B!b;
    A2B!x;
    result := ((not (b and g)) and (b or g));
end;

process Bob;
var g, x_hat, b_hat, x, b, result : bool; rq: qubit;
    abort: bool;          {Bob aborts the protocol}
    dontknow: bool;       {Bob cannot determine Alice's honesty}
begin
    AtoB?rq;
    if
    □ true →g:=false; B2A!g;
    □ true →g:=true; B2A!g;
    fi
    {Select random measurement basis and measure}
    if
    □ true →b_hat:=false;
    □ true →b_hat:=true; had rq;
    fi
    x_hat:=meas rq;
    {Receive original b and x}
    A2B?b;
    A2B?x;
    {Compare bases and bits}
    if
    □ ((b=b_hat) and (not (x=x_hat))) →abort:=true;
    □ (not (b=b_hat)) →dontknow:=true;
    fi
    result := ((not (b and g)) and (b or g));
end;
endprogram.
 finalstateproperty (Alice. result == Bob.result);
 property (AG (((b==b_hat) and (x==x_hat)) imp (abort==false)))
```