Verifying Quantum Communication Protocols with Ground Bisimulation

Xudong Qin
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
steven_qxd@126.com

Yuxin Deng
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
yxdeng@sei.ecnu.edu.cn

Abstract

One important application of quantum process algebras is to formally verify quantum communication protocols. With a suitable notion of behavioural equivalence and a decision method, one can determine if the specification of a protocol is consistent with an implementation. Ground bisimulation is a convenient behavioural equivalence for quantum processes because of its associated coinduction proof technique. We exploit this technique to design and implement an on-the-fly algorithm to check if two given processes in quantum CCS are equivalent, which enables us to develop a tool that can verify interesting quantum protocols such as the BB84 quantum key distribution scheme.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Operational semantics

Keywords and phrases Quantum process algebra, Bisimulation, Verification, Quantum communication protocols

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Process algebras provide a useful formal method for specifying and verifying concurrent systems. Their extensions to the quantum setting have also appeared in the literature. For example, Jorrand and Lalire [22, 25] defined the *Quantum Process Algebra* (QPAlg) and presented a branching bisimulation to identify quantum processes with the same branching structure. Gay and Nagarajan [18] developed *Communicating Quantum Processes* (CQP), for which Davidson [9] established a bisimulation congruence. Feng et al. [13] have proposed a quantum variant of Milner's CCS [27], called qCCS, and a notion of probabilistic bisimulation for quantum processes, which is then improved to be a general notion of bisimulation that enjoys a congruence property [15]. Later on, motivated by [29], Deng and Feng [12] defined an open bisimulation for quantum processes that makes it possible to separate ground bisimulation and the closedness under super-operator applications, thus providing not only a neater and simpler definition, but also a new technique for proving bisimilarity. In order to avoid the problem of instantiating quantum variables by potentially infinitely many quantum states, Feng et al. [14] extended the idea of symbolic bisimulation [20] for value-passing CCS and provided a symbolic version of open bisimulation for qCCS. They also proposed an algorithm for checking symbolic ground bisimulation.

In the current work, we consider the ground bisimulation proposed in [12]. We put forward an on-the-fly algorithm to check if two given processes in qCCS with fixed initial quantum states are ground bisimilar. The algorithm is simpler than the one in [14] because the initial quantum states are determined for the former but can be parametric for the latter. Therefore, it is easier to implement. Moreover, in many applications, we are only interested in the correctness of a quantum protocol with a predetermined input of quantum states. This is especially the case in the design stage of a protocol or in the debugging of a program.

The new algorithm is obtained by adapting the on-the-fly algorithm for checking probabilistic bisimulations [11], which in turn has its root in similar algorithms for checking classical bisimulations [17, 20]. The basic idea is as follows. A quantum process with an initial quantum state forms a configuration. We describe the operational behavour of a configuration as a probabilistic labelled transition system (pLTS), where probabilistic transitions arise naturally because measuring a quantum system can entail a probability distribution of post-measurement quantum systems. The notion of ground bisimulation is a strengthening of probabilistic bisimulation by imposing some constraints on quantum variables and the environment states of processes. Therefore, the skeleton of the algorithm for quantum ground bisimulation resembles to that for probabilistic bisimulation. We have developed a tool that can check if two given configurations are ground bisimilar. It is useful to validate whether the specification of a protocol is equivalent to an implementation. We have conducted experiments on a few interesting quantum protocols including super-dense coding, teleportation, secret sharing, and in particular the BB84 quantum key distribution protocol [5].

Other related work Ardeshir-Larijani et al. [3] proposed a quantum variant of CCS to describe quantum protocols. The syntax of that variant is similar to qCCS but its semantics is very different. The behaviour of a concurrent process is a finite tree and an interleaving is a path from the root to a leaf. By interpreting an interleaving as a superoperator [30], the semantics of a process is a set of superoperators. The equivalence checking between two processes boils down to the equivalence checking between superoperators, which is accomplished by using the stabiliser simulation algorithm invented by Aaronson and Gottesman [1]. Ardeshir-Larijani et al. have implemented their approach in an equivalence checker in Java and verified several quantum protocols from teleportation to secret sharing. However, they are not able to handle the BB84 quantum key distribution protocol because its correctness cannot be specified as an equivalence between interleavings. Our approach is based on ground bisimulation and keeps all the branching behaviour of a concurrent process. Our algorithm of checking ground bisimulations is influenced by the on-the-fly algorithm of Hennessy and Lin for value-passing CCS [20] and inspired by the probabilistic bisimulation checking algorithm of Baier et al. [4].

Kubota et al. [24] implemented a semi-automated tool to check a notion of symbolic bisimulation and used it to verify the equivalence of BB84 and another quantum key distribution protocol based on entanglement distillation [31]. There are two main differences between their work and ours. (1) Their tool is based on equational reasoning and thus requires a user to provide equations while our tool is fully automatic. (2) Their semantic interpretation of measurement is different and entails a kind of linear-time semantics for quantum processes that ignores the timepoints of the occurrences of probabilistic branches. However, we use a branching-time semantics. For instance, the occurrence of a measurement before or after a visible action is significant for our semantics but not for the semantics proposed in [24].

Besides equivalence checking, based on either superoperators or bisimulations as mentioned above, model checking is another feasible approach to verify quantum protocols. For instance, Gay et al. developed the QMC model checker [19]. Feng et al. implemented the tool QPMC [16] to model check quantum programs and protocols. There are also other approaches for verifying quantum systems. Abramsky and Coecke [2] proposed a categorical semantics for quantum protocols. Quantomatic [23] is a semi-automated tool based on graph rewriting. Ying [33] established a quantum Hoare logic, which has been implemented in a theorem

prover [26].

The rest of the paper is structured as follows. In Section 2 we recall the syntax and semantics of the quantum process algebra qCCS. In Section 3 we present an algorithm for checking ground bisimulations. In Section 4 we report the implementation of the algorithm and some experimental results on verifying a few quantum communication protocols. Finally, we conclude in Section 5 and discuss some future work.

2 Quantum CCS

We introduce a quantum extension of classical CCS (qCCS) which was originally studied in [13, 32, 15]. Three types of data are considered in qCCS: as classical data we have `Bool` for booleans and `Real` for real numbers, and as quantum data we have `Qbt` for qubits. Consequently, two countably infinite sets of variables are assumed: $cVar$ for classical variables, ranged over by $x, y, ...$, and $qVar$ for quantum variables, ranged over by $q, r, ...$ We assume a set Exp, which includes $cVar$ as a subset and is ranged over by $e, e', \ldots,$ of classical data expressions over `Real`, and a set of boolean-valued expressions $BExp$, ranged over by $b, b', \ldots,$ with the usual boolean constants `true`, `false`, and operators $\neg$, $\wedge$, $\vee$, and $\rightarrow$. In particular, we let $e \bowtie e'$ be a boolean expression for any $e, e' \in Exp$ and $\bowtie \in \{>, <, \geq, \leq, =\}$. We further assume that only classical variables can occur freely in both data expressions and boolean expressions. Two types of channels are used: $cChan$ for classical channels, ranged over by $c, d, ...,$ and $qChan$ for quantum channels, ranged over by $\underline{c}, \underline{d}, ...$ A relabelling function f is a map on $cChan \cup qChan$ such that $f(cChan) \subseteq cChan$ and $f(qChan) \subseteq qChan$. Sometimes we abbreviate a sequence of distinct variables $q_1, ..., q_n$ into $\tilde{q}$.

The terms in qCCS are given by:

$$\begin{aligned} P, Q \quad ::= \quad & \mathbf{nil} \mid \tau.P \mid c?x.P \mid c!e.P \mid \underline{c}?q.P \mid \underline{c}!q.P \mid \mathcal{E}[\tilde{q}].P \mid M[\tilde{q}; x].P \mid \\ & P + Q \mid P \,||\, Q \mid P[f] \mid P\backslash L \mid \mathbf{if}\ b\ \mathbf{then}\ P \mid A(\tilde{q}; \tilde{x}) \end{aligned}$$

where f is a relabelling function and $L \subseteq cChan \cup qChan$ is a set of channels. Most of the constructors are standard as in CCS [27]. We briefly explain a few new constructors. The process $\underline{c}?q.P$ receives a quantum datum along quantum channel $\underline{c}$ and evolves into P, while $\underline{c}!q.P$ sends out a quantum datum along quantum channel $\underline{c}$ before evolving into P. The symbol $\mathcal{E}$ represents a trace-preserving super-operator applied on the systems $\tilde{q}$. The process $M[\tilde{q}; x].P$ measures the state of qubits $\tilde{q}$ according to the observable M and stores the measurement outcome into the classical variable x of P.

Free classical variables can be defined in the usual way, except for the fact that the variable x in the quantum measurement $M[\tilde{q}; x]$ is bound. A process P is closed if it contains no free classical variable, i.e. $fv(P) = \emptyset$.

The set of free quantum variables for process P, denoted by $qv(P)$ can be inductively defined as in Figure 1. For a process to be legal, we require that

1. $q \notin qv(P)$ in the process $\underline{c}!q.P$;
2. $qv(P) \cap qv(Q) = \emptyset$ in the process $P \,||\, Q$;
3. Each constant $A(\tilde{q}; \tilde{x})$ has a defining equation $A(\tilde{q}; \tilde{x}) := P$, where P is a term with $qv(P) \subseteq \tilde{q}$ and $fv(P) \subseteq \tilde{x}$.

The first condition says that a quantum system will not be referenced after it has been sent out. This is a requirement of the quantum no-cloning theorem. The second condition says that parallel composition $||$ models separate parties that never reference a quantum system simultaneously.

$$
\begin{array}{rclcrcl}
qv(\mathbf{nil}) & = & \emptyset & & qv(\tau.P) & = & qv(P) \\
qv(c?x.P) & = & qv(P) & & qv(c!e.P) & = & qv(P) \\
qv(\underline{c}?q.P) & = & qv(P) - \{q\} & & qv(\underline{c}!q.P) & = & qv(P) \cup \{q\} \\
qv(\mathcal{E}[\tilde{q}].P) & = & qv(P) \cup \tilde{q} & & qv(M[\tilde{q};x].P) & = & qv(P) \cup \tilde{q} \\
qv(P+Q) & = & qv(P) \cup qv(Q) & & qv(P \parallel Q) & = & qv(P) \cup qv(Q) \\
qv(P[f]) & = & qv(P) & & qv(P\backslash L) & = & qv(P) \\
qv(\mathbf{if}\ b\ \mathbf{then}\ P) & = & qv(P) & & qv(A(\tilde{q};\tilde{x})) & = & \tilde{q}.
\end{array}
$$

Figure 1 Free quantum variables

Throughout the paper we implicitly assume the convention that processes are identified up to α-conversion, bound variables differ from each other and they are different from free variables.

Before introducing the operational semantics of qCCS processes, we review the model of probabilistic labelled transition systems (pLTSs). Later on we will interpret the behaviour of quantum processes in terms of pLTSs because quantum measurements give rise to probability distributions naturally.

We begin with some notations. A (discrete) probability distribution over a set S is a function $\Delta : S \to [0,1]$ with $\sum_{s\in S} \Delta(s) = 1$; the support of such a Δ is the set $\lceil \Delta \rceil = \{\, s \in S \mid \Delta(s) > 0 \,\}$. The point distribution $\overline{s}$ assigns probability 1 to s and 0 to all other elements of S, so that $\lceil \overline{s} \rceil = \{s\}$. We only need to use distributions with finite supports, and let $Dist(S)$ denote the set of finite support distributions over S, ranged over by Δ, Θ etc. If $\sum_{k\in K} p_k = 1$ for some collection of $p_k \geq 0$, and the Δ_k are distributions, then so is $\sum_{k\in K} p_k \cdot \Delta_k$ with $(\sum_{k\in K} p_k \cdot \Delta_k)(s) = \sum_{k\in K} p_k \cdot \Delta_k(s)$.

▶ **Definition 1.** *A* probabilistic labelled transition system *is a triple* $\langle S, \mathsf{Act}, \rightarrow \rangle$*, where* S *is a set of states,* Act *is a set of actions, and* $\rightarrow \subseteq S \times \mathsf{Act} \times Dist(S)$ *is the transition relation.*

We often write $s \xrightarrow{\alpha} \Delta$ for $(s, \alpha, \Delta) \in \rightarrow$. In pLTSs we not only consider relations between states, but also relations between distributions. Therefore, we need the lifting operations below [11].

▶ **Definition 2.** *Let* $\mathcal{R} \subseteq S \times S$ *be a relation between states. Then* $\mathcal{R}^\circ \subseteq Dist(S) \times Dist(S)$ *is the smallest relation that satisfies the two rules: (i)* $s\ \mathcal{R}\ s'$ *implies* $\overline{s}\ \mathcal{R}^\circ\ \overline{s'}$*; (ii)* $\Delta_i\ \mathcal{R}^\circ\ \Theta_i$ *for all* $i \in I$ *implies* $(\sum_{i\in I} p_i \cdot \Delta_i)\ \mathcal{R}^\circ\ (\sum_{i\in I} p_i \cdot \Theta_i)$ *for any* $p_i \in [0,1]$ *with* $\sum_{i\in I} p_i = 1$*, where* I *is a finite index set.*

We now give the semantics of qCCS. For each quantum variable q we assume a 2-dimensional Hilbert space $\mathcal{H}_q$. For any nonempty subset $S \subseteq qVar$ we write $\mathcal{H}_S$ for the tensor product space $\bigotimes_{q\in S} \mathcal{H}_q$ and $\mathcal{H}_{\overline{S}}$ for $\bigotimes_{q\notin S} \mathcal{H}_q$. In particular, $\mathcal{H} = \mathcal{H}_{qVar}$ is the state space of the whole environment consisting of all the quantum variables, which is a countably infinite dimensional Hilbert space.

Let P be a closed quantum process and ρ a density operator on $\mathcal{H}$,[1] the pair $\langle P, \rho \rangle$ is called a *configuration*. We write Con for the set of all configurations, ranged over by $\mathcal{C}$ and $\mathcal{D}$. We interpret qCCS with a pLTS whose states are all the configurations definable in the language, and whose transitions are determined by the rules in Figure 2; we have omitted the

[1] As $\mathcal{H}$ is infinite dimensional, ρ should be understood as a density operator on some finite dimensional subspace of $\mathcal{H}$ which contains $\mathcal{H}_{qv(P)}$.

(Tau)
$$\langle \tau.P, \rho\rangle \xrightarrow{\tau} \langle P, \rho\rangle$$

(C-Outp)
$$\frac{v = [\![e]\!]}{\langle \underline{c}!e.P, \rho\rangle \xrightarrow{\underline{c}!v} \langle P, \rho\rangle}$$

(Q-inp)
$$\frac{r \notin qv(\underline{c}?q.P)}{\langle \underline{c}?q.P, \rho\rangle \xrightarrow{\underline{c}?r} \langle P[r/q], \rho\rangle}$$

(Q-Com)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{\underline{c}?r} \langle P_1', \rho\rangle \qquad \langle P_2, \rho\rangle \xrightarrow{\underline{c}!r} \langle P_2', \rho\rangle}{\langle P_1 \mid\mid P_2, \rho\rangle \xrightarrow{\tau} \langle P_1' \mid\mid P_2', \rho\rangle}$$

(Meas)
$$\frac{M = \sum_{i\in I} \lambda_i E^i \qquad p_i = tr(E^i_{\tilde{q}}\rho)}{\langle M[\tilde{q}; x].P, \rho\rangle \xrightarrow{\tau} \sum_{i\in I} p_i \langle P[\lambda_i/x], E^i_{\tilde{q}}\rho E^{i\dagger}_{\tilde{q}}/p_i\rangle}$$

(Int)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{\alpha} \Delta \qquad qbv(\alpha) \cap qv(P_2) = \emptyset}{\langle P_1 \mid\mid P_2, \rho\rangle \xrightarrow{\alpha} \Delta \mid\mid P_2}$$

(Rel)
$$\frac{\langle P, \rho\rangle \xrightarrow{\alpha} \Delta}{\langle P[f], \rho\rangle \xrightarrow{f(\alpha)} \Delta[f]}$$

(Cho)
$$\frac{\langle P, \rho\rangle \xrightarrow{\alpha} \Delta \qquad [\![b]\!] = \mathtt{true}}{\langle \mathbf{if}\ b\ \mathbf{then}\ P, \rho\rangle \xrightarrow{\alpha} \Delta}$$

(C-Inp)
$$\frac{v \in \mathtt{Real}}{\langle c?x.P, \rho\rangle \xrightarrow{c?v} \langle P[v/x], \rho\rangle}$$

(C-Com)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{c?v} \langle P_1', \rho\rangle \qquad \langle P_2, \rho\rangle \xrightarrow{c!v} \langle P_2', \rho\rangle}{\langle P_1 \mid\mid P_2, \rho\rangle \xrightarrow{\tau} \langle P_1' \mid\mid P_2', \rho\rangle}$$

(Q-Outp)
$$\langle \underline{c}!q.P, \rho\rangle \xrightarrow{\underline{c}!q} \langle P, \rho\rangle$$

(Oper)
$$\langle \mathcal{E}[\tilde{q}].P, \rho\rangle \xrightarrow{\tau} \langle P, \mathcal{E}_{\tilde{q}}(\rho)\rangle$$

(Sum)
$$\frac{\langle P_1, \rho\rangle \xrightarrow{\alpha} \Delta}{\langle P_1 + P_2, \rho\rangle \xrightarrow{\alpha} \Delta}$$

(Res)
$$\frac{\langle P, \rho\rangle \xrightarrow{\alpha} \Delta \qquad cn(\alpha) \cap L = \emptyset}{\langle P\backslash L, \rho\rangle \xrightarrow{\alpha} \Delta\backslash L}$$

(Cons)
$$\frac{\langle P[\widetilde{v}/\widetilde{x}, \tilde{r}/\tilde{q}], \rho\rangle \xrightarrow{\alpha} \Delta \qquad A(\widetilde{x}, \tilde{q}) := P}{\langle A(\widetilde{v}, \tilde{r}), \rho\rangle \xrightarrow{\alpha} \Delta}$$

Figure 2 Operational semantics of qCCS. Here in rule *(C-Outp)*, $[\![e]\!]$ is the evaluation of e, and in rule *(Meas)*, $E^i_{\tilde{q}}$ denotes the operator E^i acting on the quantum systems $\tilde{q}$.

obvious symmetric counterparts to the rules *(C-Com)*, *(Q-Com)*, *(Int)* and *(Sum)*. The set of actions Act takes the following form, consisting of classical/quantum input/output actions.

$$\{c?v, c!v \mid c \in cChan, v \in \mathtt{Real}\} \quad \cup \quad \{\underline{c}?r, \underline{c}!r \mid \underline{c} \in qChan, r \in qVar\}$$

We use $cn(\alpha)$ for the set of channel names in action α. For example, we have $cn(\underline{c}?x) = \{\underline{c}\}$ and $cn(\tau) = \emptyset$.

In the first eight rules in Figure 2, the targets of arrows are point distributions, and we use the slightly abbreviated form $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ to mean $\mathcal{C} \xrightarrow{\alpha} \overline{\mathcal{C}'}$.

The rules use the obvious extension of the function $\mid\mid$ on terms to configurations and distributions. To be precise, $\mathcal{C} \mid\mid P$ is the configuration $\langle Q \mid\mid P, \rho\rangle$ where $\mathcal{C} = \langle Q, \rho\rangle$, and $\Delta \mid\mid P$ is the distribution defined by:

$$(\Delta \mid\mid P)(\langle Q, \rho\rangle) \stackrel{def}{=} \begin{cases} \Delta(\langle Q', \rho\rangle) & \text{if } Q = Q' \mid\mid P \text{ for some } Q' \\ 0 & \text{otherwise.} \end{cases}$$

Similar extension applies to $\Delta[f]$ and $\Delta\backslash L$.

▶ **Definition 3** ([12]). *A relation $\mathcal{R} \subseteq Con \times Con$ is a* ground simulation *if $\mathcal{C}\ \mathcal{R}\ \mathcal{D}$ implies that $qv(\mathcal{C}) = qv(\mathcal{D})$, $\mathrm{env}(\mathcal{C}) = \mathrm{env}(\mathcal{D})$, and*

- *whenever $\mathcal{C} \xrightarrow{\alpha} \Delta$, there is some distribution Θ with $\mathcal{D} \stackrel{\hat{\alpha}}{\Longrightarrow} \Theta$ and $\Delta\ \mathcal{R}^\circ\ \Theta$.*

A relation $\mathcal{R}$ is a ground bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are ground simulations. We denote by $\sim$ the largest ground bisimulation, called* ground bisimilarity.

3 Algorithm

In this section, we present an on-the-fly algorithm to check if two configurations are ground bisimilar. For convenience, we will only consider pLTSs with finite tree structures. On the one hand, this makes the algorithm easier to describe and analyse. On the other hand, our main motivation of this work is to verify quantum communication protocols and, to the best of our knowledge, almost all of them can be specified by qCCS processes without recursion. Other verification tools such as those in [24, 3] also adopt this design choice; they disallow recursion in their modelling language. Modifying the algorithm to deal with pLTSs with loops is possible, with an increased (but still polynomial) time complexity.

In Algorithm 1, the main function is **Bisim**(t, u). It initialises the start state pair (t, u), the set W for visited state pairs, which is empty initially, and then searches for a bisimulation based on that initialisation. The algorithm keeps updating three sets: the above mentioned W, the set N for non-bisimilar state pairs and the set B for bisimilar state pairs. The function **Match**(t, u, W) invokes a depth-first traversal to match a pair of states (t, u) with all their possible behaviors. There are three possibilities that two states are deemed non-bisimilar: (1) one state has a transition that the other cannot match, (2) they do not have the same set of free quantum variables, or (3) the density operators corresponding to their quantum registers are different. If one of the three cases takes place, we add the state pair into N. Otherwise these two states are bisimilar, then we add the state pair into B.

An auxiliary function **Act**(t, u) is called in **Match** to discover the next action that both t and u can perform. If the two states have no more action to do then the function returns an empty set. If only one of them has no more action to do then the two states are immediately declared to be non-bisimilar.

The other set W is updated in function **MatchAction**(γ, t, u, W). This function discovers next pairs of states or distributions, depending on the type of transitions, and recursively invokes the function **Match** or **MatchDistribution**. The current state pair is added to W when the new functions are invoked.

The function **MatchDistribution**(Δ, Θ, R) is called if we need to compare a pair of state distributions instead of a single pair of states. It returns a boolean value indicating whether the distributions are equivalent. In order to do so, it compares each pair of states from the supports of the two distributions. After checking the bisimilarity of these state pairs, the function generates an equivalence relation of the state pairs not contained in the set N for non-bisimilar state pairs. Another auxiliary function **Check**(Δ, Θ, R) is used to check whether Δ and Θ are related by the lifted relation R°. Technically, we take advantage of a nice property of the lifting operation: $\Delta\ R^\circ\ \Theta$ if and only if the maximum flow in an appropriately constructed network is 1 [4, 11]. There are standard algorithms for computing the maximum flow in a network; see e.g. [8]. Besides the lifting condition, we check the disjunction of the returning boolean values from function **Match**.

Now let us prove the termination and correctness of the algorithm.

▶ **Theorem 4** (Termination). *Given two states t and u from two loop-free pLTSs,* ***Bisim(t,u)*** *always terminates.*

Proof. In the absence of loops in the pLTSs, the termination of the algorithm is easy to see. Starting from the initial pair of states, the next action to perform will be detected in

Algorithm 1 Bisim(t,u)

Require: Two pLTSs with initial states t and u.

Ensure: A boolean value θ indicating if the two pLTSs are bisimilar, a set N of non-bisimilar state pairs and a set B of bisimilar state pairs.

function Bisim(t, u)
  return Match(t,u,W)

function Match(t, u, W) $\triangleright$ $t = \langle P, \rho\rangle$ *and* $u = \langle Q, \sigma\rangle$
  if $t, u \in W$ **then**
    $\theta := \mathtt{tt}$
  else
    for $\gamma \in \mathtt{Act}(t, u)$ **do**
      $(\theta_\gamma, N_\gamma, B_\gamma)$:=**MatchAction**$(\gamma,t,u,W)$
    θ:=$\bigwedge_\gamma \theta_\gamma \wedge qv(P) = qv(Q) \wedge tr_{\overline{qv(P)}}(\rho) = tr_{\overline{qv(P)}}(\sigma)$
    $N = \bigcup_\gamma N_\gamma$
    $B = \bigcup_\gamma B_\gamma$
    if $\theta = \mathtt{ff}$ **then** N:=$N \cup \{(t, u)\}$
    else if $\theta = \mathtt{tt}$ **then** B:=$B \cup \{(t, u)\}$
  return (θ, N, B)

function MatchAction(γ, t, u, W)
  switch γ **do**
    case $c!$
      for $t \xrightarrow{c!e_i} t_i$ **do**
        for $u \xrightarrow{c!e'_j} u_j$ **do**
          $(\theta_{ij}, N_{ij}, B_{ij}) :=$ **Match**$(t_i, u_j, W \cup \{(t, u)\}$
      return $(\bigwedge_i(\bigvee_j(\theta_{ij} \wedge e_i = e'_j)) \wedge \bigwedge_j(\bigvee_i(\theta_{ij} \wedge e_i = e'_j)), \bigcup_{ij} N_{ij}, \bigcup_{ij} B_{ij})$
    case τ
      for $t \xrightarrow{\tau} \Delta_i$ **do**
        for $u \xrightarrow{\tau} \Theta_j$ **do**
          $(\theta_{ij}, N_{ij}, B_{ij}) :=$ **MatchDistribution**$(\Delta_i, \Theta_j, W \cup \{(t, u)\}$
      return $(\bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij}), \bigcup_{ij} N_{ij}, \bigcup_{ij} B_{ij})$
    otherwise
      for $t \xrightarrow{\gamma} t_i$ **do**
        for $u \xrightarrow{\gamma} u_j$ **do**
          $(\theta_{ij}, N_{ij}, B_{ij}) :=$ **Match**$(t_i, u_j, W \cup \{(t, u)\}$
      return $(\bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij}), \bigcup_{ij} N_{ij}, \bigcup_{ij} B_{ij})$

function MatchDistribution(Δ, Θ, W)
  for $t_i \in \lceil\Delta\rceil$ and $u_j \in \lceil\Theta\rceil$ **do**
    $(\theta_{ij}, N_{ij}, B_{ij}) :=$ **Match**(t_i, u_j, W)
  R:=$\{(t_i, u_j) | (t_i, u_j) \notin \bigcup_{ij} N_{ij}\}^*$
  return (**Check**$(\Delta,\Theta,R) \wedge \bigvee_{ij} \theta_{ij}, \bigcup_{ij} N_{ij}, \bigcup_{ij} B_{ij}$)

function **Match**. Then it invokes function **MatchAction** to find the next new pair of states and recursively call function **Match** to check them. Each time function **MatchAction** calls function **Match** it adds the current state pair into W at the same time. If we reach the leaf nodes, there is no more action, we only compare the quantum variables used and the state of quantum registers. After that, the function terminates, so do the calls to the other functions. Moreover, if there still exist actions enabled in one pLTS but not in the other, then the two pLTSs are not bisimilar and then the whole algorithm terminates. ◀

▶ **Theorem 5** (Correctness). *Given two states t and u from two pLTSs, $\boldsymbol{Bisim}(t, u)$ returns true if and only if they are ground bisimilar.*

Proof. The proof of the correctness is similar to that in [20]. Since our algorithm is not symbolic, our treatement of boolean constraints is easier. On the other hand, we need to deal with probability distributions and have an extra procedure **MatchDistribution** to check if two distributions are related by a lifted relation. The detailed proof is provided in Appendix A. ◀

At the end of this section, we analyse the time complexity of the algorithm.

▶ **Theorem 6** (Complexity). *Let the number of nodes reachable from t and u is n. The time complexity of function $\boldsymbol{Bisim}(t, u)$ is $O(n^5/log\ n)$.*

Proof. The number of state pairs is at most n^2. When a state pair (t, u) is examined, each transition of t is compared with all transitions of u with the same action. Since the pLTSs are assumed to be finite trees, the number of comparisons of transitions does not exceed some constant. Each comparison may call the function **Check** at most once, which requires time $O(n^3/log\ n)$ if we use the maximum network flow algorithm in [8]. As a result, the execution time of $\mathbf{Bisim}(t, u)$ is in $O(n^5/log\ n)$. ◀

4 Implementation and Experiments

In this section, we report on an implementation of our approach and provide the experimental results of verifying several quantum communication protocols.

4.1 Implementation

We have implemented a ground bisimulation checker based on Algorithm 1 in Python 3.7. The workflow of our tool is sketched in Figure 3. The tool consists of a pLTS generation module and a bisimulation checking module, devoted to modeling and verification, respectively. The input of this tool is a specification and an implementation of a quantum protocol, both described as qCCS processes, the definition of user-defined operators (some of them are shown in Appendix C), as well as an initialisation of classical and quantum variables. Unlike classical variables, the initialisation of all quantum variables, deemed as a quantum register, is accomplished at the same time so to allow for superposition states. The final output of the tool is a result indicating whether the specification and the implementation, under the same initial states, are bisimilar, together with a set of bisimilar state pairs and a set of non-bisimilar state pairs.

The pLTS generation module acts as a preprocessing unit before the verification task. It first translates the input qCCS processes into two abstract syntax trees (ASTs) by a parser. Then the ASTs are transformed into two pLTSs according to the operational semantics given in Figure 2, using the user-defined operators and the initial values of variables. The

bisimulation checking module implements the ground bisimilarity checking algorithm we defined in the last section. It checks whether the initial states of the two generated pLTSs are bisimilar.

The tool is available in [**?**], where we also provide all the examples for the experiments to be discussed in Section 4.3.

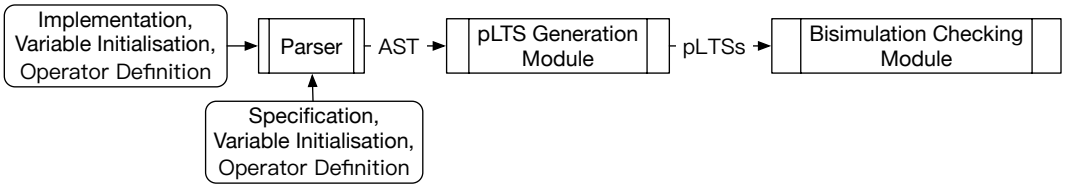

Figure 3 Verification workflow.

4.2 Example: BB84 Quantum Key Distribution Protocol

To illustrate the use of our tool, we formalise the BB84 quantum key distribution protocol. Our formalisation follows [12], where a manual analysis of the protocol is provided. Now we perform automatic verification via the ground bisimulation checker. More examples are given in Appendix B.

The BB84 protocol provides a provably secure way to create a private key between two partners with a classical authenticated channel and a quantum insecure channel between them. The protocol does not make use of entangled states. It ensures its security through the basic property of quantum mechanics: if the states to be distinguished are not orthogonal, such as $|0\rangle$ and $|+\rangle$, then information gain about a quantum state is only possible at the expense of changing the state. Let the sender and the receiver be *Alice* and *Bob*, respectively. The basic BB84 protocol with a sequence of qubits $\tilde{q}$ with size n goes as follows:

(1) *Alice* randomly generates two sequences of bits $\tilde{B}_a$ and $\tilde{K}_a$ using her qubits $\tilde{q}$.

(2) *Alice* prepares the state of $\tilde{q}$, such that the ith bits of $\tilde{q}$ is $|x_y\rangle$ where x and y are the ith bits of $\tilde{B}_a$ and $\tilde{K}_a$, and respectively, $|0_0\rangle = |0\rangle$, $|0_1\rangle = |1\rangle$, $|1_0\rangle = |+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|1_1\rangle = |-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$.

(3) *Alice* sends her qubits $\tilde{q}$ to *Bob*.

(4) *Bob* randomly generates a sequence of bits $\tilde{B}_b$ using his qubits $\tilde{q'}$.

(5) *Bob* measures the ith qubit of $\tilde{q}$ he received from *Alice* according to the basis determined by the ith bit of $\tilde{B}_b$. Respectively, the basis is $\{|0\rangle, |1\rangle\}$ if it is 0 and $\{|+\rangle, |-\rangle\}$ if it is 1.

(6) *Bob* sends his choice of measurements $\tilde{B}_b$ to *Alice*, and after receiving the information, *Alice* sends her $\tilde{B}_a$ to *Bob*.

(7) *Alice* and *Bob* match two sequences of bits $\tilde{B}_a$ and $\tilde{B}_b$ to determine at which positions the bits are equal. If the bits match, they keep the corresponding bits of $\tilde{K}_a$ and $\tilde{K}_b$. Otherwise, they discard them.

After the execution the basic BB84 protocol, the remaining bits of $\tilde{K}_a$ and $\tilde{K}_b$ should be the same, provided that the communication channels are prefect and there is no eavesdropper.

Then we consider the case that there exists an eavesdropper called *Eve* taking part in the communication. *Alice* and *Bob* also have more behaviours to detect *Eve*. In the BB84 protocol with an eavesdropper, let $\tilde{K'}_a$ and $\tilde{K'}_b$ to be the remaining bits of $\tilde{K}_a$ and $\tilde{K}_b$ with size k. Then *Alice*, *Bob* and *Eve* proceed as follows:

(1) *Alice* randomly chooses $\lceil k/2 \rceil$ bits of $\tilde{K'}_a$, denoted by $\tilde{K''}_a$ and sends it to *Bob* together with the indexes of the chosen bits.

Program	Variables	Bisim	Impl	Spec	N	B	Sec
Super-dense coding 1	$q_1 = \|0\rangle$ $q_2 = \|0\rangle$ $x = 1$	Yes	15	15	0	11	11
Super-dense coding 2	$q_1 = \|0\rangle$ $q_2 = \|0\rangle$ $x = 5$	No	5	12	-	-	0.2
Super-dense coding (modified)	$q_1 = \|0\rangle$ $q_2 = \|0\rangle$ $x = 5$	Yes	15	15	0	11	11
Teleportation 1	$q_1 = \|1\rangle$ $q_2 = \|0\rangle$ $q_3 = \|0\rangle$	Yes	33	15	0	22	19
Teleportation 2	$q_1 = \frac{1}{\sqrt{2}}\|0\rangle + \frac{1}{\sqrt{2}}\|1\rangle$ $q_2 = \|0\rangle$ $q_3 = \|0\rangle$	Yes	33	15	0	22	19
Teleportation 3	$q_1 = \frac{\sqrt{3}}{2}\|0\rangle + \frac{1}{2}\|1\rangle$ $q_2 = \|0\rangle$ $q_3 = \|0\rangle$	Yes	33	15	0	22	19
Secret Sharing 1	$q_1 = \|1\rangle$ $q_2 = \|0\rangle$ $q_3 = \|0\rangle$ $q_4 = \|0\rangle$	Yes	102	26	0	65	62
Secret Sharing 2	$q_1 = \frac{1}{\sqrt{2}}\|0\rangle + \frac{1}{\sqrt{2}}\|1\rangle$ $q_2 = \|0\rangle$ $q_3 = \|0\rangle$ $q_4 = \|0\rangle$	Yes	102	26	0	65	66
Secret Sharing 3	$q_1 = \frac{\sqrt{3}}{2}\|0\rangle + \frac{1}{2}\|1\rangle$ $q_2 = \|0\rangle$ $q_3 = \|0\rangle$ $q_4 = \|0\rangle$	Yes	102	26	0	65	58
BB84	$q1 = \|0\rangle$ $q2 = \|0\rangle$	Yes	151	131	304	414	1371
BB84 (with eavesdropper)	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $q3 = \|0\rangle$	No	1243	763	-	-	56367
BB84 (with eavesdropper & modified)	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $q3 = \|0\rangle$	Yes	1179	779	17272	12294	1585740

Table 1 Experimental results. The columns headed by **Impl** and **Spec** show the numbers of nodes contained in the generated pLTSs of the implementations and specifications, respectively. Column **N** shows the sizes of the sets of non-bisimilar state pairs and Column **B** shows the sizes of the sets of bisimilar state pairs. Column **Sec** shows the time cost of the verification in milliseconds.

(2) After receiving the information from *Alice*, *Bob* chooses $\lceil k/2 \rceil$ bits of $\tilde{K'}_b$ according to the indexes he received, denoted by $\tilde{K''}_b$ and sends it back to *Alice*.
(3) *Alice* and *Bob* match two sequences of bits $\tilde{K''}_a$ and $\tilde{K''}_b$. If two sequences match, then they have not detected the eavesdropper and the remaining substrings of $\tilde{K'}_a$ and $\tilde{K'}_b$ are used as the secure key. Otherwise, they detect *Eve* and the protocol halts without generating any secure keys.

Implementation. For simplicity, we assume that the sequence $\tilde{q}$ consists of only one qubit. This is enough to reflect the essence of the protocol. The other qubits used below are auxiliary qubits for the operation *Ran*.

$$Alice \stackrel{def}{=} Ran[q_1; B_a].Ran[q_1; K_a].Set_{K_a}[q_1].H_{B_a}[q_1].\underline{A2B}!q_1.$$
$$b2a?B_b.a2b!B_a.key_a!cmp(K_a, B_a, B_b).\mathbf{nil};$$
$$Bob \stackrel{def}{=} \underline{A2B}?q_1.Ran[q_2; B_b].M_{B_b}[q_1; K_b].b2a!B_b.$$
$$a2b?B_a.key_b!cmp(K_b, B_a, B_b).\mathbf{nil};$$
$$BB84 \stackrel{def}{=} (Alice||Bob) \setminus \{a2b, b2a, \underline{A2B}\}$$

where there are several special operations:

- $Ran[q; x] = Set_+[q].M_{0,1}[q; x].Set_0[q]$, where Set_+ (resp.Set_0) is the operation which sets a qubit it applies on to $|+\rangle$ (resp.$|0\rangle$), $M_{0,1}[q; x]$ is the quantum measurement on q according to the basis $\{|0\rangle, |1\rangle\}$ and stores the result into x.
- $Set_K[q]$ sets the qubit q to the state $|K\rangle$.
- $H_B[q]$ applies H or does nothing on the qubit q depending on whether the value of B is 1 or 0.
- $M_B[q; K]$ is the quantum measurement on q according to the basis $\{|+\rangle, |-\rangle\}$ or $\{|0\rangle, |1\rangle\}$ depending on whether the value of B is 1 or 0.
- $cmp(x, y, z)$ returns x if y and z match, and ϵ, meaning it is empty, if they do not match.

Specification. The specification can be defined as follows using the same operations:

$$BB84_{spec} \stackrel{def}{=} Ran[q_1; B_a].Ran[q_1; K_a].Ran[q_2; B_b]$$
$$.(key_a!cmp(K_a, B_a, B_b).\mathbf{nil}||cmp(K_a, B_a, B_b).\mathbf{nil}).$$

Input. For the implementation $BB84$, we need to declare the following variables and operators in the input attached to it.

- The classical bits are named B_a, K_a for *Alice* and B_b, K_b for *Bob*.
- The qubits are declared together as a vector $|q_1, q_2\rangle$. The vector always needs an initial value. We can set it to be $|00\rangle$ in this example.

When modelling the protocol, we use several operators. They should be defined and their definitions are part of the input.

- The operator *Ran* involves two operators Set_+, Set_0 and a measurement $M_{0,1}$.
- Set_K needs Set_0 and Set_1.
- H_B requires the Hadamard gate H.
- M_B uses the measurement $M_{+,-}$.

The function *cmp* is treated as an in-built function, so there is no need to define it in the input.

For the specification $BB84_{spec}$, we only declare the classical bits B_a, B_b, K_a, qubits q_1, q_2 and the operator Ran. The variables and operators declared here are the same as those in the input of the implementation.

Output. Taking the input discussed above, the tool first generates two pLTSs, with over 130 states for each of them, and then runs the ground bisimulation checking algorithm. As we can see from the third last row in Table 1, our tool confirms that $\langle BB84, \rho_0\rangle \sim \langle BB84_{spec}, \rho_0\rangle$, where ρ_0 denotes the initial state of the quantum register, thus the implementation is faithful to the specification. In the output of the tool, there is an enumeration of 304 pairs of non-bisimilar states and 414 pairs of bisimilar states. The pLTSs and the state pairs can be found in [**?**].

Implementation with an Eavesdropper. We proceed to model the protocol with an eavesdropper. For that purpose, we extend the processes $Alice$ and Bob with two processes for eavesdropper detection.

$$\begin{aligned} Alice' \stackrel{def}{=} & key_a?K'_a.Pstr_{K'_a}[q_1;x].a2b!x.a2b!SubStr(K'_a,x).b2a?K''_b.\\ & (\textbf{if } SubStr(K'_a,x) = K''_b \textbf{ then } key'_a!RemStr(K'_a,x).\textbf{nil}\\ & \textbf{else } alarm_a!0.\textbf{nil});\\ Bob' \stackrel{def}{=} & key_b?K'_b.a2b?x.a2b?K''_a.b2a!SubStr(K'_b,x).\\ & (\textbf{if } SubStr(K'_b,x) = K''_a \textbf{ then } key'_b!RemStr(K'_b,x).\textbf{nil}\\ & \textbf{else } alarm_b!0.\textbf{nil}) \end{aligned}$$

where there are three more special operations:

- $Pstr$ is a measurement similar to Ran; it randomly generates the value of x.
- $SubStr(K,x)$ returns the substring of K at the index specified by x.
- $RemStr(K,x)$ returns the remaining substring of K by deleting $SubStr(K,x)$.

Then we define the eavesdropper as follows.

$$Eve \stackrel{def}{=} \underline{A2E}?q_1.Ran[q_3;B_e].M_{B_e}[q_1;K_e].Set_{K_e}[q_1].H_{B_e}[q_1].\underline{E2B}!q_1.key_e!K_e.\textbf{nil}.$$

With the participation of Eve, we adjust the communication of $Alice$ and Bob:

$$Alice \longrightarrow Alice[f_a], Bob \longrightarrow Bob[f_b]$$

where $f_a(\underline{A2B}) = \underline{A2E}$, and $f_b(\underline{A2B}) = \underline{E2B}$.

Next, we introduce a test process $Test$ before arriving at the new implementation $BB84'$:

$$\begin{aligned} Test \stackrel{def}{=} & key'_a?x.key'_b?y.key'_e?z.\\ & (\textbf{if } x \neq y \textbf{ then } fail!0.\textbf{nil}\\ & + \textbf{ if } x = y \textbf{ then } key_e!z.skey!x.\textbf{nil});\\ BB84' \stackrel{def}{=} & (Alice||Bob||Alice'||Bob'||Eve||Test) \setminus C \end{aligned}$$

where $C = \{a2b, b2a, key_a, key_b, \underline{A2E}, \underline{E2B}, alarm_a, alarm_b\}$.

Specification with an Eavesdropper. Taking into account the presence of an eavesdropper, we define the new specification as follows:

$$
\begin{aligned}
BB84'_{spec} \overset{def}{=} & Ran[q_1; B_a].Ran[q_1; K_a].Ran[q_3; B_e].Ran'_{B_a,B_e,K_a}[q_1; K_e].Ran[q_2; K_b]. \\
& Ran'_{B_e,B_b,K_e}[q_1; K_b].Pstr[q_1; x]. \\
& (\textbf{if } K_{ab} = K_{ba} \textbf{ then } key_e!K_e.skey!RemStr(K_{ab}, x).\textbf{nil} \\
& + \textbf{if } K_{ab} \neq K_{ba} \textbf{ then} \\
& (\textbf{if } K^x_{ab} \neq K^x_{ba} \textbf{ then } alarm_a!0.\textbf{nil}||alarm_b!0.\textbf{nil} \\
& + \textbf{ if } K^x_{ab} = K^x_{ba} \textbf{ then } fail!0.\textbf{nil}))
\end{aligned}
$$

where $K_{ab} = cmp(K_a, B_a, B_b)$, $K_{ba} = cmp(K_b, B_a, B_b)$, $K^x_{ab} = SubStr(K_{ab}, x)$, $K^x_{ab} = SubStr(K_{ba}, x)$. And similar to Ran, $Ran'_{x,y,z}[q; v]$ is a special measurement that randomly generates the value of v if x and y do not match and give v the value of z if they match.

As to the input, we need to declare more variables for $BB84'$ as there are more roles involved in the communication.

- The classical bits named K'_a, K''_a for *Alice* are used for storing the result of processing the sequence. The bits K'_b, K''_b for *Bob* do the same work.
- The classical bits for *Eve* are named B_e and K_e.
- The classical bits named x, y, z are declared to store the remaining string.
- The qubits are declared together to be a longer vector $|q_1, q_2, q_3\rangle$. We set it to be $|000\rangle$ in this example.

Similar to the function *cmp*, the functions *SubStr* and *RemStr* are already declared inside the tool.

For the specification $BB84'_{spec}$, we declare the bits K_{ab}, K_{ba}, K^x_{ab} and K^x_{ba}. Certainly, we still need to declare B_e and K_e.

We see from the second last row in Table 1 that in this case our tool gives a negative verification result, i.e. $\langle BB84', \rho_0\rangle \not\sim \langle BB84'_{spec}, \rho_0\rangle$. In other words, the implemention $BB84'$ is unsatisfactory. After carefully examining the behaviour of the process $BB84'$, we find that the problem lies in the fact that *Alice*′ and *Bob*′ have excessive freedom to trigger alarms. *Alice* and *Bob* will declare an alarm if their measurement methods are not matched. The parallelism between the final test process and them leads to a process that continues exhibiting undesirable actions. This is not what the specification exactly describes. The implementation has more behaviours than what the specification requires. Some modification is needed to make it behave the same as the specification.

Improved BB84 Protocol with an Eavesdropper. We fix the problem discussed above by requiring *Alice*′ and *Bob*′ to communicate with *Test* before the latter triggers an alarm, thus we move the delaration of alarms to the test process. *Alice*′ and *Bob*′ only send messages

when they use different measurements. The following three processes are updated:

$$Alice' \stackrel{def}{=} key_a?K'_a.Pstr_{K'_a}[q_1;x].a2b!x.a2b!SubStr(K'_a,x).b2a?K''_b.$$
$$(\textbf{if } SubStr(K'_a,x) = K''_b \textbf{ then } key'_a!RemStr(K'_a,x).\textbf{nil}$$
$$\textbf{else } msg_a!0.\textbf{nil});$$
$$Bob' \stackrel{def}{=} key_b?K'_b.a2b?x.a2b?K''_a.b2a!SubStr(K'_b,x).$$
$$(\textbf{if } SubStr(K'_b,x) = K''_a \textbf{ then } key'_b!RemStr(K'_b,x).\textbf{nil}$$
$$\textbf{else } msg_b!0.\textbf{nil});$$
$$Test \stackrel{def}{=} key'_a?x.key'_b?y.key'_e?z.$$
$$(\textbf{if } x \neq y \textbf{ then } fail!0.\textbf{nil} \ + \ \textbf{if } x = y \textbf{ then } key_e!z.skey!x.\textbf{nil})$$
$$+ \ msg_a?x.msg_b?y.key'_e?z.alarm!0.\textbf{nil}.$$

The last row in Table 1 tells us that the above modification of the implementation is indeed correct.

4.3 Experimental Results

We conducted experiments on four quantum communication protocols and a few variants of them. Table 1 provides a summary of our experimental results obtained on a macOS machine with an Intel Core i7 2.5 GHz processor and 16GB of RAM. In each case, we report the final outcome (whether an implementation is equivalent to its specification), the number of nodes in two pLTSs, the numbers of non-bisimilar and bisimilar state pairs in N and B, repectively, as well as the verification time of our ground bisimulation checking algorithm (excluding the pLTS generation part).

In Table 1 there are cases where negative verification results are obtained. One of them is already discussed in Section 4.2. We now comment on the other case. We verify the super-dense coding protocol with two different initial values of variable x in the first two rows. If $x = 1$, we can check that the implementation behaves the same as the specification. However, if $x = 5$, the behaviour of the implementation differs from that of the specification, and they are even not trace equivalent. See Appendix B.1 for more discussion and the improvement of the formal modelling.

Not all the cases in Table 1 give the size of the set N of non-bisimilar state pairs, as the bisimulation checking algorithm may immediately terminate once a negative verification result is obtained, i.e. the two initial states are not bisimilar.

5 Conclusion and Future Work

We have presented an on-the-fly algorithm to check ground bisimulation for quantum processes in qCCS. Based on the algorithm, we have developed a tool to verify quantum communication protocols modelled as qCCS processes without recursion. We have carried out experiments on several non-trivial quantum communication protocols from super-dense coding to key distribution and found the tool helpful.

As to future work, a couple of interesting problems remain to be addressed. For example, the behavioural equivalence considered in the current work is a strong notion of ground bisimulation because all actions are visible. In practical verifications, it is common to introduce invisible actions in implementations. Then it is more appropriate to equate an implementation with a specification with respect to a weak notion of bisimulation that

abstracts away invisible actions. Another problem with the current work is to compare quantum processes with predetermined states of quantum registers. However, there are occasions where one would expect two processes to be equivalent for arbitrary initial states. It is infeasible to enumerate all those states. Then the symbolic bisimulations proposed in [14] will be useful. We are considering to implement the algorithm for symbolic ground bisimulation, and then tackle the more challenging symbolic open bisimulation, both proposed in that work.

References

1. Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(052328), 2004.
2. Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science*, pages 415–425. IEEE Computer Society, 2004.
3. Ebrahim Ardeshir-Larijani, Simon J. Gay, and Rajagopal Nagarajan. Automated equivalence checking of concurrent quantum systems. *ACM Trans. Comput. Log.*, 19(4):28:1–28:32, 2018.
4. Christel Baier, Bettina Engelen, and Mila E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60(1):187–231, 2000.
5. C. H. Bennett and G. Brassard. Quantum cryptography: Public-key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computer, Systems and Signal Processing*, pages 175–179, 1984.
6. C.H. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, and W. Wootters. Teleporting an unknown quantum state via dual classical and EPR channels. *Physical Review Letters*, 70:1895–1899, 1993.
7. C.H. Bennett and S.J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.
8. Joseph Cheriyan, Torben Hagerup, and Kurt Mehlhorn. Can A maximum flow be computed on o(nm) time? In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 235–248. Springer, 1990.
9. T. A. S. Davidson. *Formal Verification Techniques using Quantum Process Calculus*. PhD thesis, University of Warwick, 2011.
10. Y. Deng, R. van Glabbeek, M. Hennessy, and C. Morgan. Testing finitary probabilistic processes (extended abstract). In *Proc. CONCUR'09*, volume 5710 of *LNCS*, pages 274–288. Springer, 2009.
11. Yuxin Deng. *Semantics of Probabilistic Processes: An Operational Approach*. Springer, 2015.
12. Yuxin Deng and Yuan Feng. Open bisimulation for quantum processes. In *Proceedings of the 7th IFIP International Conference on Theoretical Computer Science*, volume 7604 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 2012.
13. Y. Feng, R. Duan, Z. Ji, and M. Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, 2007.
14. Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic*, 15(2):1–32, 2014.
15. Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for quantum processes. In *Proc. POPL'11*, pages 523–534. ACM, 2011.
16. Yuan Feng, Ernst Moritz Hahn, Andrea Turrini, and Lijun Zhang. QPMC: A model checker for quantum programs and protocols. In *Proceedings of the 20th International Symposium on Formal Methods*, volume 9109 of *Lecture Notes in Computer Science*, pages 265–272. Springer, 2015.

17 Jean-Claude Fernandez and Laurent Mounier. Verifying bisimulations "on the fly". In *Proceedings of the 3rd International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*, pages 95–110. North-Holland, 1990.

18 S. J. Gay and R. Nagarajan. Communicating quantum processes. In J. Palsberg and M. Abadi, editors, *Proc. POPL'05*, pages 145–157, 2005.

19 Simon J. Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou. QMC: A model checker for quantum systems. In *Proceedings of the 20th International Conference on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 543–547. Springer, 2008.

20 Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theor. Comput. Sci.*, 138(2):353–389, 1995.

21 Mark Hillery, Vladimír Bužek, and André Berthiaume. Quantum secret sharing. *Physical Review A*, 59(3):1829, 1999.

22 Philippe Jorrand and Marie Lalire. Toward a quantum process algebra. In *Proceedings of the 1st Conference on Computing Frontiers*, pages 111–119. ACM, 2004.

23 Aleks Kissinger. *Pictures of Processes: Automated Graph Rewriting for Monoidal Categories and Applications to Quantum Computing*. PhD thesis, University of Oxford, 2011.

24 Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, and Hideki Sakurada. Semi-automated verification of security proofs of quantum cryptographic protocols. *Journal of Symbolic Computa*, 73:192–220, 2016.

25 Marie Lalire. Relations among quantum processes: Bisimilarity and congruence. *Mathematical Structures in Computer Science*, 16(3):407–428, 2006.

26 Tao Liu, Yangjia Li, Shuling Wang, Mingsheng Ying, and Naijun Zhan. A theorem prover for quantum hoare logic and its applications. *CoRR*, abs/1601.03835, 2016.

27 R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

28 M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2000.

29 Davide Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Informatica*, 33(1):69–97, 1996.

30 Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

31 P.W. Shor and J. Preskill. Simple proof of security of the bb84 quantum key distribution protocol. *Physical Review Letters*, 85(2):441–444, 2000.

32 M. Ying, Y. Feng, R. Duan, and Z. Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic*, 10(3):1–36, 2009.

33 Mingsheng Ying. *Foundations of Quantum Programming*. Morgan Kaufmann, 2016.

A Correctness of Algorithm 1

In this section we give a detailed proof of the correctness of the algorithm. To simplify the presentation, we use $\mathtt{R}(t, u, W, N, B)$ to mean the following condition is satisfied:

- If $(t', u') \in N \vee (t', u') \in B \wedge t' \xrightarrow{\alpha} t'' \wedge u' \xrightarrow{\alpha'} u'', (t', u') \notin \{(t, u)\} \cup W \wedge$:
  - if $\alpha \equiv c!e \wedge \alpha' \equiv c!e'$ then
    - $e = e' \wedge (t'', u'') \notin W \wedge (t'', u'') \in B \implies t'' \sim u''$.
    - $e \neq e' \vee ((t'', u'') \notin W \wedge (t'', u'') \in N) \implies t'' \not\sim u''$.
  - let $t'' \equiv \Delta'$ and $u'' \equiv \Theta'$, if $\alpha \equiv \tau \wedge \alpha' \equiv \tau$, then $\forall t'_i \in \lceil \Delta' \rceil, u'_j \in \lceil \Theta' \rceil$
    - $(t'_i, u'_j) \notin W \wedge (t'_i, u'_j) \in B \implies t'_i \sim u'_j$.
    - $(t'_i, u'_j) \notin W \wedge (t'_i, u'_j) \in N \implies t'_i \not\sim u'_j$.
  - otherwise $\alpha = \alpha'$, then
    - $(t'', u'') \notin W \wedge (t'', u'') \in B \implies t'' \sim u''$.
    - $(t'', u'') \notin W \wedge (t'', u'') \in N \implies t'' \not\sim u''$.

▶ **Lemma 7.** *If* $N_1 \cap N_2 = \emptyset \wedge B_1 \cap B_2 = \emptyset$ *then* $\mathtt{R}(t,u,W,N_1,B_1)$ *and* $\mathtt{R}(t,u,W,N_2,B_2)$ *implies* $\mathtt{R}(t,u,W,N_1 \cup N_2, B_1 \cup B_2)$.

Proof. Straightforward from the definition of R. ◀

We define the verification conditions of our three matching functions.

▶ **Definition 8.** ***Match***(t,u,W) *is true if the following conditions are satisfied:*

- *(C1)* $W \cap N = \emptyset \wedge W \cap B = \emptyset \wedge N \cap B = \emptyset$ *and*
  - *if* $(t,u) \in W$*, then* $(t,u) \notin N \wedge (t,u) \in B$*,*
  - *if* $(t,u) \notin W$*, then either* $\theta = true \wedge (t,u) \in B$ *or* $\theta = false \wedge (t,u) \in N$*.*
- *(C2)* $\mathtt{R}(t,u,W,N,B)$*.*

Let ***Bisim***$(t,u) =$***Match***$(t,u,\emptyset)$*.*

▶ **Definition 9.** ***MatchAction***(γ,t,u,W) *is true if all the following conditions are satisfied:*

- *(M1)* $W \cap N = \emptyset \wedge (t,u) \notin W \wedge (t,u) \notin N \wedge (t,u) \notin B$*.*
- *(M2)* $\mathtt{R}(t,u,W,N,B)$*.*
- *(M3)* $\forall t \xrightarrow{\alpha} t', \exists u \xrightarrow{\alpha'} u'$*,* $(t',u') \notin \{(t,u)\} \cup W$ *and*
  - *if* $\alpha \equiv c!e \wedge \alpha' \equiv c!e'$ *then*
    - $e = e' \wedge (t',u') \notin W \wedge (t',u') \in B \implies t' \sim u'$*.*
    - $e \neq e' \vee ((t',u') \notin W \wedge (t',u') \in N) \implies t' \not\sim u'$*.*
  - *let* $t' \equiv \Delta$ *and* $u' \equiv \Theta$*, if* $\alpha \equiv \tau \wedge \alpha' \equiv \tau$*, then* $\forall t_i \in \lceil \Delta \rceil, u_j \in \lceil \Theta \rceil$
    - $(t_i,u_j) \notin W \wedge (t_i,u_j) \in B \implies t_i \sim u_j$*.*
    - $(t_i,u_j) \notin W \wedge (t_i,u_j) \in N \implies t_i \not\sim u_j$*.*
  - *otherwise* $\alpha = \alpha'$*, then*
    - $(t',u') \notin W \wedge (t',u') \in B \implies t' \sim u'$*.*
    - $(t',u') \notin W \wedge (t',u') \in N \implies t' \not\sim u'$*.*

▶ **Definition 10.** ***MatchDistribution***(Δ,Θ,W) *is true if the following conditions are satisfied:*

- *(D1)* $W \cap N = \emptyset$*,* $\forall t_i \in \lceil \Delta \rceil, u_j \in \lceil \Theta \rceil$*,* $(t_i,u_j) \notin W$ *and* $\exists (t_i,u_j) \in B$*.*
- *(D2) Let* $t \xrightarrow{\alpha} \Delta, u \xrightarrow{\alpha'} \Theta$*,* $\mathtt{R}(t,u,W,N,B)$*.*

▶ **Proposition 11.** *Let* ***MatchAction***$_\gamma(\gamma,t,u,W)$ *be the execution of* ***MatchAction*** *with action* γ*. If* ***MatchAction***$_\gamma(\gamma,t,u,W)$ *is true for each action* γ *then* ***Match***(t,u,W) *is also true, and it returns* $\theta = \bigwedge_\gamma \theta_\gamma$ *and* $N = \bigcup_\gamma N_\gamma$*.*

Proof. The only time point that (t,u) is added into W is during the execution of **MatchAction**, then according to the Definition 9, we have $W \cap N = \emptyset$ and $W \cap B = \emptyset$. Since the verified pLTS is a finite tree, if they reach the leaf states of the pLTSs, there should be $\theta = true$, $N = \emptyset$ and $B = \{(t,u)\}$, at the same time $(t,u) \notin W \wedge (t,u) \notin N \wedge (t,u) \in B$. Furthermore, we have $t \sim u$ in such case. According to the structure of the function, (t,u) will be added into N if θ is false and B if θ is true. As a result, $C1$ is satisfied.

From conditions (M2) and (M3), $\mathtt{R}(t,u,W,N_\gamma,B_\gamma)$ exists. According to Lemma 7, we have the condition that $\mathtt{R}(t,u,W,\bigcup_\gamma N_\gamma,\bigcup_\gamma B_\gamma)$, and then $C2$ is satisfied. ◀

▶ **Proposition 12.** *Suppose* $(t,u) \notin W$*. Assume that* ***Match***$(t_i,u_j,W \cup \{(t,u)\})$ *is true for all actions* $\gamma \neq \tau$ *when there exist transitions* $(t \xrightarrow{\gamma} t_i,\ u \xrightarrow{\gamma} u_j)$ *or* ***MatchDistribution***$(\Delta_i,\Theta_j,$ $W \cup \{(t,u)\})$ *is true for all actions* $\gamma = \tau$ *when there exist transitions* $(t \xrightarrow{\tau} \Delta_i, u \xrightarrow{\tau} \Theta_j)$*. Then* ***MatchAction***$(\gamma,t,u,W \cup \{(t,u)\})$ *is true and* $\theta = \bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij})$*,* $N = \bigcup N_{ij}$*.*

Proof. From the structure of **MatchAction**, (t, u) does not exist in W, and (t, u) can not be added into N or B here. So the first condition is satisfied.

To show $(M2)$ and $(M3)$, we first consider the case where (t_i, u_j) are already the leaves of the finite trees. If $\theta_{ij} = qv(t_i) = qv(u_j) \wedge (tr_{\overline{qv(u_i)}}\rho_i) = tr_{\overline{qv(u_j)}}(\sigma_j)$ is *true*, we have $(t_i, u_j) \in B_{ij}$ and $N_{ij} = \emptyset$. As there is no more outgoing transitions, we have $t_i \sim u_j$.

If it is not the leaf node, by $(C2)$, we have $\mathtt{R}(t_i, u_j, \{(t, u)\} \cup W, N_{ij}, B_{ij})$. Since $B = \bigcup_{ij} B_{ij}$, we get that $(t_i'', u_j'') \notin W \wedge (t_i'', u_j'') \in B$ implies $t_i'' \sim u_j''$. By the definition of ground bisimulation, we get that $(t_i', u_j') \notin W \wedge (t_i', u_j') \in B$ implies $t_i' \sim u_j'$, so $(M2)$ is satisfied.

If θ is true, since $\theta = \bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij})$, there exists θ_{ij} that is *true*, then there is some $(t_i, u_j) \in B_{ij}$. Similarly, by the definition of ground bisimulation, $(M3)$ is also satisfied.

The final case we need to consider is the distribution (Δ, Θ) instead of a node. If θ is true, then the θ_{ij} returned from **Check** should also be *true*. So the function **Match** must return *true*, which means that $(t_i, u_j) \in B$ implies $t_i \sim u_j$.

The pair $(t_i, u_j) \in N$ is similar with the pair $(t_i, u_j) \in B$ while it implies that $t_i \not\sim u_j$. ◀

▶ **Proposition 13.** *Suppose* $\forall t_i \in \lceil\Delta\rceil, u_j \in \lceil\Theta\rceil, (t_i, u_j) \notin W$. *If* $\textbf{\textit{Match}}(t_i, u_j, W)$ *is true then* $\textbf{\textit{MatchDistribution}}(\Delta, \Theta, W)$ *is true where* Δ *and* Θ *satisfy the lifting condition,* $\theta = \textbf{\textit{Check}}(\Delta, \Theta, R) \wedge \bigvee_{ij} \theta_{ij}$ *and* $N = \bigcup_{ij} N_{ij}$.

Proof. According to the verification conditions of **March**, all the $\mathbf{Match}(t_i, u_j, W)$ have been finished before we get R and call **Check**. If $\Delta \sim \Theta$, then we have $\exists (t_i, u_j) \in B$ implies that $t_i \sim u_j$. ◀

Proof of Theorem 5. From the verification condition of **Match**, we have that if $\mathbf{Bisim}(t, u) = \mathbf{Match}(t, u, \emptyset)$ returns $(true, N, B)$, we guarantee the bisimilarity $t \sim u$. ◀

B Examples

B.1 Super-dense Coding Protocol

Super-dense coding is proposed by Bennett and Wiesner in 1992 [7]. It is a quantum communication protocol allowing two classical bits to be encoded in one qubit during a transmission, so it needs only one quantum channel. Such advantage is based on the use of a maximally entangled state, EPR state. An EPR state can be transformed into all the four kinds of EPR states through a 1-qubit operation, and these EPR states are mutually orthogonal.

Protocol. We suppose the sender and the receiver of the communication are *Alice* and *Bob*, then the protocol goes as follows:

(1) *Alice* and *Bob* prepare an EPR state $|\beta_{00}\rangle_{q_1,q_2}$ together. Then they share the qubits, *Alice* holding q_1 and *Bob* holding q_2.

(2) If *Alice* wants to send value $x \in \{0, 1, 2, 3\}$, she applies the corresponding Pauli operation σ^x on her qubit q_1.

(3) *Alice* sends the qubit q_1 to *Bob*.

(4) *Bob* applies a controlled-not operation on q_1, q_2 and a Hadamard operation on q_1 to remove the entanglement.

(5) *Bob* measures q_1 and q_2 to get the value x.

After the execution of the protocol above, *Bob* gets the value x which *Alice* wants to send. Note that x could be represented in a 2-bit string. The protocol exactly transmits two classical bits of information by sending one qubit from *Alice* to *Bob*.

Implementation. We model the super-dense coding protocol in qCCS as follows:

$$Alice \stackrel{def}{=} \underline{c}_A?q_1.\sum_{0\le i\le 3}(\textbf{if } x=i \textbf{ then } \sigma^i[q_1].\underline{e}!q_1.\textbf{nil});$$

$$Bob \stackrel{def}{=} \underline{c}_B?q_2.\underline{e}?q_1.CN[q_1,q_2].H[q_1].M[q_1,q_2;x].d!x.\textbf{nil};$$

$$EPR \stackrel{def}{=} Set^{\Psi}[q_1,q_2].\underline{c}_B!q_2.\underline{e}_A!q_1.\textbf{nil};$$

$$Sdc \stackrel{def}{=} c?x.(Alice||Bob||EPR)\setminus\{\underline{c}_A,\underline{c}_B,\underline{e}\}$$

where CN is the controlled-not operation and H is the Hadamard operation, Set^{Ψ} is the operation transforming all the inputs into an EPR state $|\beta_{00}\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$, its operation elements are $\{|\beta_{00}\rangle\langle 00|, |\beta_{00}\rangle\langle 01|, |\beta_{00}\rangle\langle 10|, |\beta_{00}\rangle\langle 11|\}$, and σ^i are Pauli operators where $\sigma^0 = I, \sigma^1 = X, \sigma^2 = Z, \sigma^3 = Y$. The element set of measurement M is $\{|00\rangle\langle 00|, |01\rangle\langle 01|, |10\rangle\langle 10|, |11\rangle\langle 11|\}$.

Specification. The specification of the protocol can be defined as:

$$Sdc_{spec} \stackrel{def}{=} c?x.\tau^{11}.\sum_{i=0}^{3}(\textbf{if } x=i \textbf{ then } Set^i[q_1,q_2].d!x.\textbf{nil})$$

where Set^i is the operation of transforming the current state into the state decided by the value of i like Set^{Ψ}. Here we have inserted some harmless τ-transitions (τ^{11} stands for a series of 11 τ-transitions) because in ground bisimulations τ-actions are not abstracted away.

Input. Associated with the specification Sdc are the following variables and operators that need to be declared.

- The classical bit is named x. It stores the value *Alice* wants to send. We test the program with different values of x.
- The qubits are declared as a vector $|q_1, q_2\rangle$. They are used for generating the EPR state here. Without loss of generality, we set them to be $|00\rangle$.
- The operation of transforming $|00\rangle$ into the EPR state is defined as Set^{Ψ}.
- The controlled-not operation is defined as CN.
- The Hadamard operation is defined as H.
- The Pauli operations are defined as $\sigma^0, \sigma^1, \sigma^2$ and σ^3.
- The measurement is defined as M with its operation elements.

For the specification Sdc_{spec}, we declare the following set of variables and operators.

- The classical bit named x is still required to store the value *Alice* wants to send.
- The qubits are declared as a vector $|q_1, q_2\rangle$. We set them to be $|00\rangle$.
- The operation of transforming an arbitrary state into $|00\rangle$ (resp. $|01\rangle$, $|10\rangle$, $|11\rangle$) is defined as Set^0 (resp. Set^1, Set^2, Set^3).

We see from the first two lines of Table 1 that not all the inputs can get a positive verification result. In the case $x = 1$, we can check that $\langle Sdc, \rho_0\rangle \sim \langle Sdc_{spec}, \rho_0\rangle$, where ρ_0 is the initial state of the quantum variables. In the case $x = 5$, none of the four branches can be chosen, then the tool find that $\langle Sdc, \rho_0\rangle \not\sim \langle Sdc_{spec}, \rho_0\rangle$. To correct it, some modifications are needed on both the implementation and the specification.

Improved Super-dense Coding Protocol. We improve the above qCCS programs by considering the case $i \neq 1, 2, 3, 4$. In such case we send an alarming message and skip all the rest of operations. The new specification Sdc' is defined below:

$$Alice' \stackrel{def}{=} \underline{c}_A?q_1.(\sum_{0\leq i\leq 3}(\textbf{if } x = i \textbf{ then } \sigma^i[q_1].\underline{e}!q_1.\textbf{nil}) + \textbf{if } \neg \bigvee_{0\leq i\leq 3} x = i \textbf{ then } c_C!msg.\textbf{nil});$$

$$Bob' \stackrel{def}{=} \underline{c}_B?q_2.(\underline{e}?q_1.CN[q_1,q_2].H[q_1].M[q_1,q_2;x].d!x.\textbf{nil} + c_C?msg.\tau^8.d!x.\textbf{nil});$$

$$EPR \stackrel{def}{=} Set^{\Psi}[q_1,q_2].\underline{c}_B!q_2.\underline{c}_A!q_1.\textbf{nil};$$

$$Sdc' \stackrel{def}{=} c?x.(Alice||Bob||EPR) \setminus \{\underline{c}_A, \underline{c}_B, c_C, \underline{e}\}.$$

We adjust the specification to add a new branch, so as to get Sdc'_{spec}:

$$Sdc'_{spec} \stackrel{def}{=} c?x.\tau^{11}.(\sum_{i=0}^{3}(\textbf{if } x = i \textbf{ then } Set^i[q_1,q_2].d!x.\textbf{nil}) + \textbf{if } \neg \bigvee_{0\leq i\leq 3} x = i \textbf{ then } Set^{\Psi}[q_1,q_2].d!x.\textbf{nil}).$$

The improved modelling processes are indeed bisimilar, as can be seen in Table 1.

B.2 Quantum Teleportation Protocol

Quantum teleportation [6] is one of the most important protocols in quantum information theory. It teleports an unknown quantum state by only sending classical information, so it just requires a classical communication channel. It makes use of a maximally entangled state.

Protocol. Let the sender and the receiver be *Alice* and *Bob*, respectively. The quantum teleportation protocol goes as follows:

(1) *Alice* and *Bob* prepare an EPR state $|\beta_{00}\rangle_{q_2,q_3}$ together. Then they share the qubits, *Alice* holding q_2 and *Bob* holding q_3.

(2) To transmit qubit q_1, *Alice* applies a CN operation on q_1 and q_2 followed by a H operation on q_1.

(3) *Alice* measures q_1 and q_2 and sends the outcome x to *Bob*.

(4) When *Bob* receives x, he applies the corresponding σ^x operation on his qubit q_3 to recover the original state of q_1.

After the execution, $Bob's$ qubit q_3 has the same state as the qubit q_1.

Implementation. We provide below an implementation of the quantum teleportation protocol:

$$Alice \stackrel{def}{=} \underline{c}_A?q2.CN[q_1,q_2].H[q_1].M[q_1,q_2;x].Set^{\Psi}[q_1,q_2].e!x.\textbf{nil};$$

$$Bob \stackrel{def}{=} \underline{c}_B?q_3.e?x.\sum_{0\leq i\leq 3}(\textbf{if } x = i \textbf{ then } \sigma^i[q_3].\textbf{nil});$$

$$EPR \stackrel{def}{=} Set^{\Psi}[q_1,q_2].\underline{c}_A!q_2.\underline{c}_B!q_3.\textbf{nil};$$

$$Tel \stackrel{def}{=} (Alice||Bob||EPR) \setminus \{\underline{c}_A, \underline{c}_B, e\}$$

where the operators used here are all already declared before.

Specification. The specification of the protocol can also be described in qCCS. To show the correctness of Tel, it suffices to prove that Tel is bisimilar to a swap operation between the first and the thrid qubits, that is $SWAP_{1,3}[q_1, q_3]$. The specification is thus easy.

$$Tel_{spec} \stackrel{def}{=} \tau^{13}.SWAP[q_1, q_3].\mathbf{nil}.$$

Input. Some of the operators defined in the input are the same as those in the super-dense coding example, so we do not repeat them. We need to declare the following variables associated with the implementation.

- The classical bits x is declared here to store the measurement result of qubits q_1, q_2.
- The qubits are declared together as a vector $|q_1, q_2, q_3\rangle$. The first qubit q_1 is what *Alice* wants to teleport. The values of the last two qubits are arbitrary as they will be transformed into an EPR state later. We set the quantum register to be $|\psi 00\rangle$ and test the processes with different values of $|\psi\rangle$.

The specification Tel_{spec} declares the same set of variables. And only one operation $SWAP_{1,3}$ is defined in the input.

We see from Table 1 that Tel and Tel_{spec} hebave the same with all the three different values of q_1.

B.3 Quantum Secret Sharing Protocol

Quantum secret sharing protocol is proposed by Hillery et al. [21]. The problem involves an agent *Alice* sending information to other two agents *Bob* and *Charlie*, one of whom is dishonest. It is a classical method which is known as secret sharing that *Alice* splits the information into two parts, then *Bob* and *Charlie* need to collaborate to get the complete information. The idea is to let the honest one keep the dishonest one from misbehaving. A quantum version of it can be realised by a three-qubit maximally entangled state called GHZ state, which has similar property as EPR states.

Protocol. The protocol goes as follows:

(1) *Alice*, *Bob* and *Charlie* prepare an GHZ state $(|000\rangle + |111\rangle)/\sqrt{2}_{q_2,q_3,q_4}$ together prior to the following execution. Then they share the qubits, *Alice* holding q_2, *Bob* holding q_3 and *Charlie* holding q_4.

(2) *Alice* entangles q_1 and q_2 by applying a CN operation followed by a H operation on q_1.

(3) *Alice* measures q_1 and q_2 separately and sends the outcomes m and n to *Charlie*.

(4) *Bob* also measures q_3 and sends the outcome o to *Charlie*.

(5) Upon receiving the bits m, n and o, *Charlie* retrieves the state through applying Pauli operations X or Z on q_4 according to the values of these bits.

After the execution, $Charlie's$ qubit q_4 has the same state as the qubit q_1.

Implementation. The quantum secret sharing protocol can be encoded in qCCS as follows:

$$Alice \stackrel{def}{=} \underline{c}_A?q2.CN[q_1,q_2].H[q_1].M[q_1;m].M[q_2;n].e!m.f!n.\mathbf{nil};$$

$$Bob \stackrel{def}{=} \underline{c}_B?q_3.H[q_3].M[q_3;o].g!o.\mathbf{nil});$$

$$Charlie \stackrel{def}{=} \underline{c}_C?q_4.e?m.f?n.g?o.$$
$$\mathbf{if}\ o=1\ \mathbf{then}\ Z[q_4].\mathbf{if}\ m=1\ \mathbf{then}\ X[q_4].\mathbf{if}\ n=1\ \mathbf{then}\ Z[q_4].\mathbf{nil});$$

$$GHZ \stackrel{def}{=} Set^{GHZ}[q_2,q_3,q_4].\underline{c}_A!q_2.\underline{c}_B!q_3.\underline{c}_C!q_4.\mathbf{nil};$$

$$QSS \stackrel{def}{=} (Alice||Bob||Charlie||GHZ) \setminus \{\underline{c}_A,\underline{c}_B,\underline{c}_C,e,f,g\}$$

where the operators used are all already declared before except that Set^{GHZ} is the operation transforming all the input into a GHZ state.

Specification. To show the above implementation is correct, we prove that QSS is bisimilar to a swap operation between the first and the fouth qubits, that is $SWAP_{1,4}[q_1,q_4]$. The specification can be written as follow:

$$QSS_{spec} \stackrel{def}{=} \tau^{24}.SWAP[q_1,q_4].\mathbf{nil}.$$

Input. In the input of the implementation, we also need to define the Clifford operations and Pauli operations presented before. Other variables and operations are declared as follows.

- The classical bits m, n and o are declared to store the measurement result of qubits q_1, q_2 and q_3.
- The qubits are declared together as a vector $|q_1,q_2,q_3,q_4\rangle$. Similar to the teleportation example, the values of the last three qubits are arbitrary as they will be transformed into a GHZ state later. And the first qubit q_1 will be set to several different values to test the implementation.
- The operation of transforming an arbitrary state into the GHZ state is defined as Set^{GHZ}.

The specification process declares the same set of variables. The only operation required is defined as $SWAP_{1,4}$.

We can see from Table 1 that the implementation behaves the same as the specification for all the three different values of q_1.

C Operators

We have frequently used the Hadamard operator, the controlled-not operator and the Pauli operators in our examples. Their matrix representations are listed below.

$$H=\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix} \qquad CN=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

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

The quantum operations Set^{Ψ}, Set^{GHZ}, Set^{i} are defined using the operator-sum representation [28] with a set of Kraus operators.

$$Set^{\Psi}: \ \{\frac{|00\rangle+|11\rangle}{\sqrt{2}}\langle 00|,\ \frac{|00\rangle+|11\rangle}{\sqrt{2}}\langle 01|,\ \frac{|00\rangle+|11\rangle}{\sqrt{2}}\langle 10|,\ \frac{|00\rangle+|11\rangle}{\sqrt{2}}\langle 11|\}$$

$$Set^{GHZ}: \{\frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 000|,\ \frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 001|,\ \frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 010|,$$
$$\frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 011|,\ \frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 100|,\ \frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 101|,$$
$$\frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 110|,\ \frac{|000\rangle+|111\rangle}{\sqrt{2}}\langle 111|\}$$

$$Set^{0}: \{|00\rangle\langle 00|,\ |00\rangle\langle 01|,\ |00\rangle\langle 10|,\ |00\rangle\langle 11|\}$$

$$Set^{1}: \{|01\rangle\langle 00|,\ |01\rangle\langle 01|,\ |01\rangle\langle 10|,\ |01\rangle\langle 11|\}$$

$$Set^{2}: \{|10\rangle\langle 00|,\ |10\rangle\langle 01|,\ |10\rangle\langle 10|,\ |10\rangle\langle 11|\}$$

$$Set^{3}: \{|11\rangle\langle 00|,\ |11\rangle\langle 01|,\ |11\rangle\langle 10|,\ |11\rangle\langle 11|\}$$